Master's thesis

2019

Bendik Skarre Abrahamsen

Master's thesis

NTNU
Norwegian University of
Science and Technology
Faculty of Natural Sciences
Department of Physics

NTNU
Norwegian University of
Science and Technology

Bendik Skarre Abrahamsen

Cluster Analysis of Multiparametric Magnetic Resonance Images of Rectal Cancer

June 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Cluster Analysis of Multiparametric Magnetic Resonance Images of Rectal Cancer

Bendik Skarre Abrahamsen

Applied Physics and Mathematics
Submission date: June 2019
Supervisor: Kathrine Røe Redalen

Norwegian University of Science and Technology
Department of Physics

Abstract:
Purpose: Cluster analysis performed on images obtained with functional magnetic resonance imaging (MRI) has been proposed as a method of partitioning heterogenous tumour volumes into more homogenous subvolumes that exhibit similar physical characteristics. The aim of this study is to examine the validity of clustering solutions based on multiparametric MRI (mpMRI) of patients with rectal cancer and to examine whether any of the obtained subvolumes are related to the progression free survival (PFS).
Materials and methods: Cluster analysis using k-means and Gaussian mixture models (GMM) was performed on a dataset containing T2 weighed (T2w) MRI, diffusion weighted images (DWI) with 6 different diffusion b-values from $0\,\mathrm{s\,mm^{-2}}$-$1000\,\mathrm{s\,mm^{-2}}$, apparent diffusion coefficient (ADC) parameter maps and intravoxel incoherent motion (IVIM) parameter maps of a patient cohort consisting of 54 patients with rectal cancer. Principal component analysis was performed to reduce the dimensionality of the dataset prior to clustering. The optimal number of clusters for the k-means algorithm was found using the Calinski-Harabasz (CH) index, Davies-Bouldin (DB) index, DB* index, DB** index and the Silhouette coefficient. In the GMM clustering solution the optimal number of components was found using the Bayesian information criterion (BIC).

For the survival analysis the patient cohort was partitioned into groups CRT or No CRT depending on whether or not they were to receive preoperative chemoradiotherapy (CRT) resulting in groups, from now on referred to as CRT and No CRT, containing 24 and 30 patients, respectively. Logrank tests using the median cluster (k-means)/component (GMM) volume within a treatment group as a cutoff and univariate Cox regression using the cluster volume as a risk factor was used to associate the obtained clusters (k-means)/components (GMM) to PFS. The Bonferroni corrections for multiple comparisons was used to correct the significance threshold according to the number of clusters tested for association with PFS. This corrections was made for the k-means and GMM clustered dataset separately.
Results: The optimal number of clusters in the k-means cluster analysis and components in the GMM cluster analysis was found the be two and nine respectively. The GMM clustering solution was found to be stable towards repeats using different randomly sampled subsets of the full dataset.

One of the clusters in the k-means partitioned dataset was found to be associated with PFS (logrank: p-value < 0.02, univariate Cox: p-value < 0.006) for the No CRT group. In the GMM partitioned dataset two components were found to be associated with the PFS by using the logrank test (p-value $= 0.001$ and p-value $= 0.001$) in the No CRT group. These components were found not to be significantly associated to the PFS at the Bonferroni corrected significance level in the Cox regression. However, a component not significantly associated with the PFS in the logrank test was found to be significantly associated with PFS in the univariate Cox regression (p-value < 0.002). A greater relative risk was shown to be associated with an increase in volume of this component as compared to a similar volume increase in the unpartitioned tumour volume.

No clusters from the k-means partitioned dataset or components in the GMM partitioned dataset were found to be associated with PFS in the CRT group.

Conclusion: Cluster analysis based on multiparametric MRI (mpMRI) and derived parameter maps of patients with rectal cancer was shown to successfully partition the tumour volumes into subvolumes associated with PFS for patients that did not receive CRT.

Sammendrag:
Formål: Klyngeanalyse utført på funksjonelle magnetresonans (MR) bilder har blitt foreslått som en metode for å dele heterogene tumorvolum inn i mer homogene subvolum. Målet med denne studien er å undersøke validiteten av klynger oppnådd ved klyngeanalyse av multiparametriske MR-bilder av pasienter med rektalkreft. Det ønskes også å undersøke hvorvidt noen av de resulterende subvolumene kan relateres til progresjonsfri overlevelse (PFS)
Materialer og metoder: Klyngeanalyse ved bruk av k-means og Gaussian Mixture Models (GMM) ble utført på et datasett bestående av T2-vektede MR-bilder, diffusjonsvektede MR-bilder med seks forskjellige diffusjons b-verdier fra $0\,\mathrm{s\,mm^{-2}}$-$1000\,\mathrm{s\,mm^{-2}}$, den såkalte apparente diffusjonskoeffisient (ADC) parameterkart og intravoxel inkoherent bevegelse (IVIM) parameterkart for 54 pasienter gitt diagnosen rektalkreft. Prinsipalkomponentanalyse ble utført for å redusere antallet dimensjoner i datasettet før klyngeanalyse ble gjennomført. Det optimale antallet klynger ble funnet ved bruk av Calinski-Harabasz (CH) indeksen, Davies-Bouldin (DB) indeksen, DB* indeksen, DB** indeksen og Silhuett koeffisienten. For GMM ble det optimale antallet komponenter funnet ved Bayes informasjonskriterium (BIC).

I overlevelsesanalysen ble pasientgruppen delt i to etter hvorvidt hver pasient mottok preoperativ kjemoradioterapi (CRT) eller ikke. De resulterende gruppene vil fra nå av bli omtalt som CRT- og ikke CRT gruppen. CRT gruppen bestod av 24 pasienter, mens ikke CRT gruppen bestod av 30 pasienter. Det ble utført logrank tester der pasienter innen hver gruppe ble inndelt etter median klyngevolum (k-means)/komponentvolum (GMM). Univariat Cox regresjon ble utført ved bruk av klyngevolum (k-means)/komponentvolum (GMM) som risikofaktor. For begge metoder var klinisk endepunkt definert ved PFS. Bonferronikorreskjon ble utført for å korrigere signifikansnivået for gjentatt testing. Korreksjonen ble utført for datasettet inndelt ved k-means klyngeanalyse og datasettet inndelt ved GMM klyngeanalyse hver for seg.
Result: Det optimale antallet klynger for k-means klyngeanalyse og komponenter for GMM klyngeanalyse var to og ni. Komponentene funnet ved GMM klyngeanalyse ble funnet å være stabile ved gjentatt inndeling av forskjellige datasett laget ved å trekke tilfeldige voxeler ut fra det fulle datasettet.

En av klyngene som ble funnet i datasettet inndelt ved bruk av k-means klyngeanalyse var relatert til PFS (logrank: p-verdi < 0.02, univariat Cox: p-verdi < 0.006) i ikke CRT pasientgruppen. I datasettet inndelt ved GMM klyngeanalyse ble det funnet to komponenter som var signifikant relatert til PFS ved bruk av logrank test (p-verdi $= 0.001$ og p-verdi $= 0.001$) i ikke CRT gruppen. Disse komponentene ble derimot funnet å ikke være signifikant relatert til PFS ved det Bonferronikorrigerte signifikansnivået når de ble undersøkt med univariat Cox regresjon. Én komponent som ikke var signifikant relatert til PFS ved logrank test var signifikant relatert til PFS ved univariat Cox regresjon (p-verdi < 0.002). En større relativ risiko var knyttet til økende volum av denne komponenten sammenlignet med en tilsvarende økning i det totale tumorvolumet.

Ingen klynger (k-means) eller komponenter (GMM) var signifikant relatert

til PFS i CRT pasientgruppen.
Konklusjon: Klyngeanalyse ble benyttet på multiparametriske MR-bilder av pasienter med rektalkreft og parameterkart utledet ved bruk av disse. Dette gav tumorvolum inndelt i subvolum der noen av disse var relatert til PFS for pasienter som ikke mottok preoperativ CRT.

Contents

1 Introduction

Rectum and rectosigmoid cancer together was the seventh most frequent type of cancer in Norway in 2016 for and it constituted 4.6% and 3.7% of all new cancer incidences for males and females respectively. This adds up to a total of 1325 new cases diagnosed in 2017 [1].

The Norwegian guidelines for diagnosis and treatment of rectal cancer have since 2003 recommended obligatory preoperative magnetic resonance examination. The current national staging guidelines for colorectal cancer states that patients that are diagnosed with rectal cancer should have an magnetic resonance (MR) examination of the rectum performed to stage the cancer and determine the spread [2]. MR images have bee shown to shown to provide excellent anatomical information about the rectum and mesorectal fascia [3].

In addition to providing accurate anatomical images MRI also has the possibility acquiring functional images. Functional images, like dynamic contrast-enhanced (DCE) MRI and diffusion weighted imaging (DWI), can depict important biologic features like vascularity and cellularity and have become important in the search for new biomarkers [4]. Image contrast in DWI is generated by the displacement of water molecules during the acquisition time [5]. This displacement can be due to diffusion, active transport, flow or perfusion [6]. These properties are known to be affected by multiple factors like cell density, vascularity, viscosity of intracellular fluid and cell membrane integrity [7].

Various studies have shown that using DWI in conjunction with standard morphological MRI can aid in assessing the response to radiotherapy. The routine use of DWI for tumour response assessment after chemoradiotherapy is now recommended in the guidelines set fourth by the European Society of Gastrointestinal and Abdominal Radiology(ESGAR) [8].

The apparent diffusion coefficient is a simple mono-exponential model aimed at quantifying the properties expressed by the DWI [9]. In cancer diagnostics the ADC is often interpreted as the cellularity of the tissue [10]. The observed ADC values are known to be sensitive to capillary perfusion [9, 11]. The more complex intravoxel incoherent motion model (IVIM) model was proposed in order to create separate diffusion- and perfusion parameter maps from DWI [11].

Tumour volumes are known be exhibit genetic and microenvironmental heterogeneity between tumours in addition to within a single tumour [12, 13]. Multiple studies have suggested that tumour heterogeneity can become a promising biomarker to differentiate between tumour types, grades, predict treatment outcome or monitor treatment effect [14].

Cluster analysis is a wide range of algorithms or methods whose goal is to discover natural groupings in a set of data [15]. Cluster analysis on multiparametric magnetic resonance images (mpMRI) has been proposed for separating the heterogenous tumour volume into smaller more homogenous subvolumes that exhibit similar physical characteristics. Validation of cluster analysis as an applicable methodology in separation the tumour volume into meaningful subvolumes has come in the form of histological examinations [16] and by the discovery of clusters within the tumour volume associated with local tumour control [17] and local relapse [18, 19].

The aim for this study was to investigate clustering of pretreatment tumour volumes for patients with rectal cancer using T2 weighted images, raw DWI, ADC parameter maps and IVIM parameter maps as input features. The performance and validity of the clustering solutions obtained by two common clustering algorithm, the *k*-means and Gaussian mixture model, was assessed. It was also tested whether any of the volumes of the obtained clusters were better biomarkers for PFS than the total tumour volume.

2 Theory

2.1 Cancer

Cancer can be defined as an abnormal growth of cells caused by multiple changes in the gene expression leading to an imbalance of cell proliferation and cell death and ultimately evolving into a population of cells that is able to invade tissues and metastasize to distant sites. Clinically cancer manifests itself as a large group of diseases that vary considerably across parameters like cellular differentiation, metastatic potential, diagnostic detectability, response to treatment and prognosis [20]. However, from a cell biological point of view, it has been proposed that a small number of molecular, biochemical and cellular traits are shared among all types of human cancer [21]. Observations suggest that human cancers progress through a process driven by stepwise, somatic-cell mutations from a single progenitor cell. Different varieties of mutant cells are produced as the tumour proliferates. Occasionally a mutant cell acquires a selective advantage with respect to the original tumour cells and surrounding normal cells. Selective pressure through the competition of space and resources causes this mutant to be the precursor of a new predominant cell population [22]. This is summarized by Greaves et al [23], as process parallel to Darwanian natural selection, in which cancer clones are asexually reproducing , unicellular quasispecies. Strong evidence suggests tumours may exhibit a branched evolution where several distinct subclonal populations may exist within a single tumour [24, 25, 26]. The observed intratumour heterogeneity can be seen as strong evidence of the mutator phenotype hypothesis [26], which predicts that that cancer development is a random process driven by an elevated spontaneous mutation rate compared to normal human cells [27].

The genetic heterogentity of the tumour tissue combined with spatial differences in environmental stressors leads to development of intratumor phenotypical differences [25].

2.2 Magnetic Resonance Imaging

The section on MRI is adapted from a project thesis written by the author of this thesis with minor changes [28].

A complete description of MRI is outside the scope of this report and the interested reader is referred the abundance of available books and reports on the subject. A short description covering some of the relevant concepts will nonetheless be given. MRI is a non-invasive imaging technique that has been under continuous development and improvement since its discovery and is today used for a wide and steadily growing range of diagnostic applications.

MRI is based on the characteristic magnetic dipole moments, $\vec{m}_p$, shown by nuclei with an uneven number of number of nucleons. The most commonly used in medical imaging in the hydrogen nucleus (^1H). Upon interaction with an external magnetic flux density, $\vec{B}_0$, the magnetic dipole moment of a proton will start to precess about $\vec{B}_0$ with a characteristic frequency, ω_0 proportional to external magnetic flux density

$$\omega_0 = -\gamma B_0. \tag{1}$$

In equation (1) γ is the gyromagnetic ratio which for a simple proton is $\gamma = e/2m$. The precession frequency ω_0 is called the *Larmour* frequency and the equation (1) is called the *Larmour* equation. The magnetic moment vector of the protons will tend to align with the external magnetic field but will due to the thermal energy associated with the absolute temperature be prevented from complete alignment.

The distribution of spins in an external magnetic field will be almost spherical, but slightly skewed towards the external magnetic field [29]. Even though the individual protons are precessing about the external magnetic field, no precession of the macroscopic magnetization vector will be seen. The magnetization vector must be tipped away from the external field direction for the precession to be observed. This is accomplished through the application of a radiofrequency magnetic field for a short time period (called an 'RF pulse'). Application of an RF pulse will act as a torque on the magnetization vector rotating the net magnetization vector about the direction of which the RF pulse is applied. An RF pulse of magnetic flux density $\vec{B}_1$ that is applied for a time interval t will have the effect of creating a flip angle α between $\vec{M}_0$ and $\vec{B}_0$ given by the relation

$$\alpha = \gamma B_1 t. \tag{2}$$

When the magnetization vector $\vec{M}_0$ has been flipped away from alignment with the external magnetic field it will begin to precess. This is due to the torque exerted by the external magnetic field on $\vec{M}_0$. $\vec{M}_0$ will gradually realign with $\vec{B}_0$. The precession of $\vec{M}_0$ around $\vec{B}_0$ will be detectable as an induced voltage in a receiver coil due to the fluctuating magnetic it represents.

Signal decay will primarily be due to two relaxation processes. These are the 'spin-lattice' relaxation and the 'spin-spin' relaxation. The 'spin-lattice' relaxation is relaxation of the signal due to interaction of the spins with their surroundings. This will tend to regrow the magnetization along $\vec{B}_0$ once it is flipped into the transverse plane. The rate at which the regrowth occurs is defined by a time constant T_1. 'Spin-spin' relaxation is the dephasing of clusters of spins constituting the magnetization vector $\vec{M}_0$. Clusters of similarly behaving spins are called spin isochromats. If we presume, as often done in MR, that the external field $\vec{B}_0$ is initially aligned along the z-axis, when $\vec{M}_0$ is flipped into the xy-plane the dephasing will cause the component of $\vec{M}_0$ in the xy-plane to diminish. The 'spin-spin' relaxation is quantified through the time constant T_2.

In MRI the different T_1 and T_2 values of different tissues are utilized to create contrast. Parameters that are used to get different image weightings, that is whether to weight differences in the T1 of tissues or T2 of tissues, are the echo time (TE) and the repetition time (TR). TE is the time between the application of the initial pulse and the maxima of the following echo, and TR is the time between application of excitation pulses. One of the most common

MRI sequences is the Spin Echo (SE) sequence. This sequence consists of 90° excitation pulse repeated every TR. This is followed by a 180° pulse that will have the effect of rephasing the spin isochromats to form an echo at the TE. This can be thought of as hands on a clock. Assume you have three hands centered at 6 o'clock each hand representing an isochromat. One hand will move counterclockwise and one clockwise at a given speed and the last hand will be stationary at 6 o'clock. Applying the 180° pulse will have the effect of moving the stationary hand to 12 o'clock. While the two hands previously diverging from 6 o'clock will now converge towards 12 o'clock. The maxima of the echo will appear after the isochromats have had the same amount of time to dephase as they've had to rephase leading to the 180° pulse placed at TE/2. Excellent animations illustrating this sequence can be found online [30].

Bloch proposed a set of equations famously known as the Bloch equations that are used to calculate the magnetization $\vec{M}_0$ in MRI and nuclear magnetic resonance(NMR) [31]. Solving these for the SE sequence leads to a solution for the transverse magnetization

$$M_{\perp}(T_R, T_E) = M_0(1 - e^{-T_R/\mathrm{T1}})e^{-T_E/\mathrm{T2}}. \tag{3}$$

Equation (3) clearly shows the influence of time constants T1 and T2 and the operational parameters TE and TR on the transverse magnetization. The transverse magnetization is related proportionally to the signal induced in the receiver coil. For $T_E << \mathrm{T2}$ the equation (3) can be approximated as (4) while for $T_R >> \mathrm{T1}$ the equation can be approximated as (5). Thus for T1 weighting of an image a short TE compared to T2 is chosen, while for T2 weighting of an image a long TR compared to T1 is chosen. Plots the equations (4) and (5) for various values of the relaxation constants T1 and T2 respectively are shown in figure 2 and 1.

$$M_{\perp}(T_R) = C(1 - e^{-T_R/\mathrm{T1}}) \tag{4}$$

$$M_{\perp}(T_E) = Ce^{-T_E/\mathrm{T2}} \tag{5}$$

To form an image in using MRI spatial localization of the signal is needed. Magnetic gradients are used to spatially encode each contribution to the signal. A magnetic gradient $G_z(\vec{r})$, here assumed applied along the z-direction, will when applied alter the Larmour frequency at which the affected protons precess according to

$$\omega = \gamma[B_0 + zG_z(\vec{r})]. \tag{6}$$

Efficient excitation of spins will only occur due to RF pulses with frequency matching the Larmour frequency of the affected spins. Thus, only spins that according to (6) has a frequency matching that of the RF pulse will be subject to significant excitation. This is the basis for frequency encoding in MRI. One gradient applied in one direction is used for selecting a slice to excite. This causes only signal from this slice to be received. Another gradient is applied in the

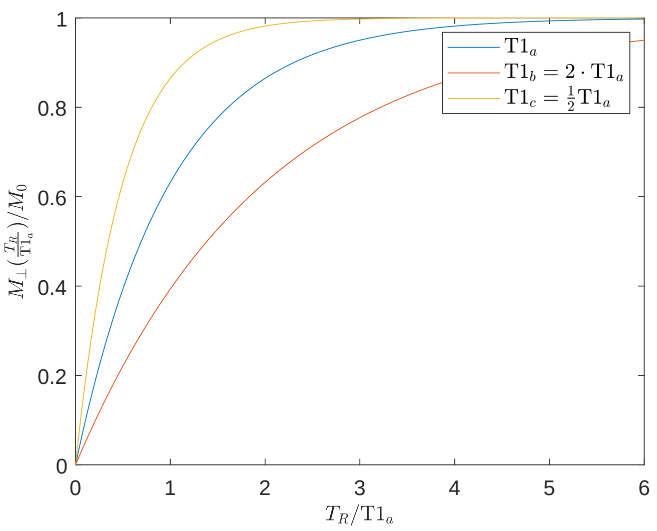


Figure 1: Normalized plot of the transverse magnetization as a function of T_R for different values of the relaxation constant T1.

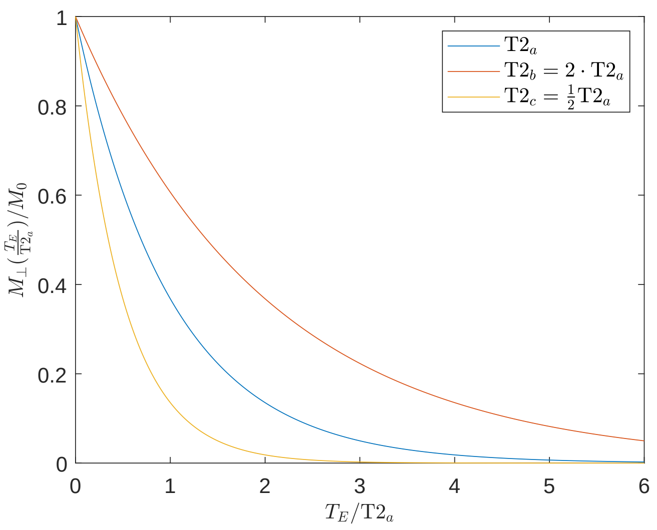


Figure 2: Normalized plot of the transverse magnetization as a function of T_E for different values of the relaxation constant T2.

so called 'frequency-encoding' direction which makes each the signal spatially distinguishable within one direction of the excited slice through their frequency by use of the Fourier transform. A third gradient is applied in the so called 'phase encoding' direction causing is spatially distinct locations of the same precession frequency to be distinguishable, and thus encoding the final of the three spatial degrees of freedom.

As mentioned previously, the rotating magnetic field will induce a current a receiver coil placed outside the patients. The strength of the current induced, and thus the signal intensity, is proportional to the transverse magnetic field. The signal is mapped to a spatial frequency domain called k-space where each axis correspond to the spatial frequency along that direction. As denoted earlier we have a frequency encoding direction and a phase encoding direction. How the frequency encoding direction relates to the k-space is straight forward as spins along the gradient have different frequency according to their position. In the phase encoding direction we encode a phase to the spins at various spatial locations, but a change in phase over distance is just another spatial frequency. Thus the phase encoding direction is often more appropriately called the indirect frequency direction. To create an image from the k-space a 2D-Fourier transform is performed.

2.2.1 Diffusion Weighted Imaging

Diffusion of intra- and extracellular water in tissues will be restricted. This means that if enough time is allowed to pass the mean diffusion distance is less than than what is expected in free diffusion [9]. MRI can be used to measure diffusion of water molecules in body by using a modified T2-weighted spin-echo sequence as proposed by Stejskal and Tanner [32], which uses pair of symmetrically balanced gradients around the 180° pulse. A linearly varying magnetic field along the z-direction added to the static magnetic field B_0 yields an effective magnetic field in the z-direction given by equation (7).

$$B_z(z,t) = B_0 + zG(z,t) \tag{7}$$

where $G_z = \frac{\partial B_z}{\partial z} = G$. The variation in angular velocity due to the additional magnetic field imposed by the magnetic field gradient is shown as the second term of equation (6) written as ω_g in equation (8).

$$\omega_G(z,t) = \gamma z G(z,t) \tag{8}$$

Allowing this gradient to be affect the spins for an amount of time t will cause a phase shift $\phi_G(z,t)$ compared having the gradient off. The resulting phase shift can be calculated as

$$\phi_G(z,t) = -\int_0^t dt' \omega_G(z,t') = -\gamma \int_0^t dt' z(t') G(z,t'). \tag{9}$$

The total phase shift $\phi(z,t)$ can be found adding the phase shift due to the static magnetic field, ϕ_0 to ϕ_G

$$\phi(z,t) = \phi_0(z,t) + \phi_G(z,t) = \gamma B_0 t + \gamma \int_0^t G(z,t') \cdot z(t') dt' \tag{10}$$

For an individual proton moving, the additional phase due to spin displacement in the direction of the gradient will be proportional to the displacement along the direction of the gradient [5]. At a certain TE in the modified Spin-Echo sequence, the total phase shift for a particular spin will thus be equal to

$$\phi(\mathrm{TE}) = \gamma \int_{t_1}^{t_1+\delta} G(z,t') \cdot z(t') dt' - \gamma \int_{t_1+\Delta}^{t_1+\Delta+\delta} G(z,t') \cdot z(t') dt' \tag{11}$$

where δ is the time each motion probing gradient is applied for and Δ is the time between each of the gradients. If no motion along the gradient occur the two terms cancel and no net phase shift is observed. In the case of diffusion each spin will acquire a random displacement causing the phase shift of the individual spins to differ. It can be shown [9] that these random phase shifts accumulated by the individual spins lead to an signal echo attenuation expressed as

$$S(b, \mathrm{TE})_{SE} = S_0 \exp{(-\frac{\mathrm{TE}}{T_2})} \exp{(-b \cdot \mathrm{ADC})} \tag{12}$$

where ADC is the apparent diffusion coefficient and the so called b-value is the diffusion-sensitizing factor that can be calculated as

$$b = \gamma^2 G^2 \delta^2 (\Delta - \frac{\delta}{3}) \tag{13}$$

Now, assuming identical TE for two signal intensity measurements of different diffusion b-value we can calculate the ADC as,

$$\mathrm{ADC} = -\frac{1}{b_1 - b_0} \ln \frac{S(b_1)}{S(b_0)}. \tag{14}$$

The ADC of equation (14) becomes the diffusion coefficient D in voxels where diffusion is the only type of motion [11]. Figure 3 shows a case where good correspondence between the the measured values and the suggested model is observed. A bigger discrepancy is seen in figure 4. As shown in (14), only signal intensities in corresponding voxels imaged with two different b-values are needed to form the ADC-map. However, more b-values gives more robust and accurate estimates, especially where likely ADC values are not known a priori [33].

Generally a range of b-values are used in DWI to study the relative water diffusion of different tissues [34]. At a b-value of $0\,\mathrm{s\,mm^{-2}}$ free water will appear bright due to the fact that this essentially is just a normal T2w sequence. Increasing the b-value to between $50\,\mathrm{s\,mm^{-2}}$ and $100\,\mathrm{s\,mm^{-2}}$ will cause signal from blood is vessels to appear dark due to the large relative motion of blood. Many tumours form highly cellular tissue where water movement is heavily restricted. Therefore, even at high diffusion b-values from 500 to $1000\,\mathrm{s\,mm^{-2}}$ these tissues will remain bright.

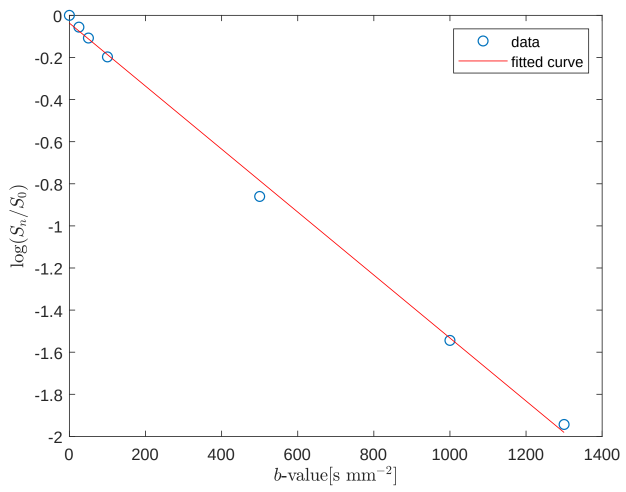


Figure 3: Fit of the logarithm of signal intensity versus b-value for a region of interest in a patient with rectal cancer according to the ADC model shown in equation (14). A good correspondence between model and result is observed.

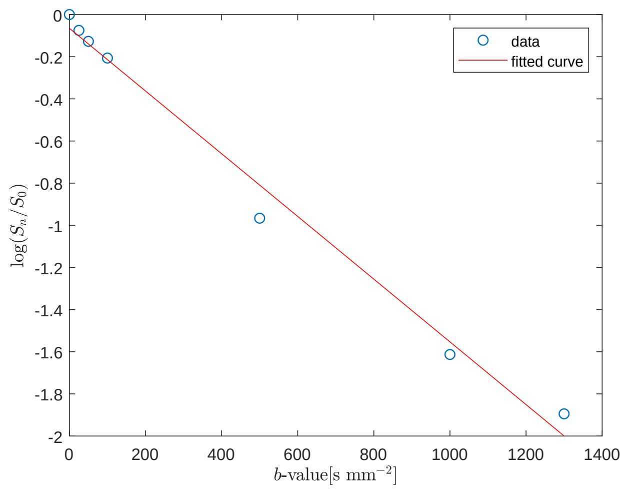


Figure 4: Fit of the logarithm of signal intensity versus b-value for a region of interest in a patient with rectal cancer according to the ADC model shown in equation (14). Sub-optimal correspondence between model and result might warrant use of a more complex model.

2.2.2 Intravoxel Incoherent Motion

As mentioned in the previous section, the ADC becomes the diffusion coefficient, D, in voxels where diffusion is the only type of motion. It has been known for some time that the observed ADC values are sensitive to capillary perfusion [9, 11]. In well perfused tissues (e.g. 0-100 $\mathrm{s\,mm^{-2}}$) the signal attenuation observed in a DWI sequence is due not only to diffusion, but also due to microcirculation within the capillary network as this process also will lead to phase dispersion in a DW-MRI. A method to correct for this motion was by suggested by Le Bihan termed Intravoxel Incoherent Motion(IVIM) [9]. The microcirculatory perfusion can be thought of as a type of ”pseudo-diffusion” as it, in the same way as the diffusion, has no inherent orientation. The rate of motion due to perfusion is greater than that of diffusion. This leads to more a greater displacement of the protons during the DW-MRI sequence resulting in a steeper signal attenuation. The effect will thus mainly be visible for lower b-values, and will be only account for a small correction at higher b-values.

The IVIM model fits the relative signal intensity to a biexponential expression of the form

$$\frac{S(B_n)}{S(B_0)} = f \cdot e^{-b\cdot(D_* + D_{\text{blood}})} + (1-f) \cdot e^{-b\cdot D} \tag{15}$$

where f represents the perfusion fraction, D is the water diffusion coefficient in tissue, D_* is the pseudo-diffusion coefficient representing microcapillary perfusion [33] and D_{blood} is the water diffusion coefficient in blood [35]. Using the improved model on the same regions of interest as shown in figure 3 and 4 yields better fits. The new improved fit of the IVIM model is shown in figures 5 and 6 together with the ADC fit for comparison.

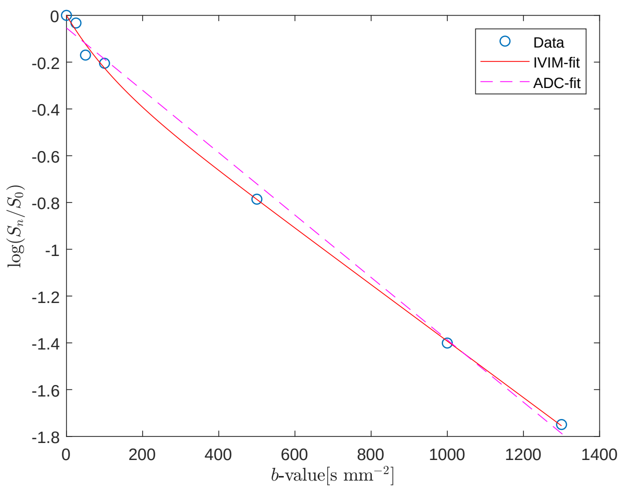


Figure 5: Fit of the logarithm of signal intensity versus b-value for a region of interest in a patient with rectal cancer. The improved fit given by the IVIM model suggested in equation (15) is shown as a solid line whereas the ADC fit is shown as a dashed line. The figure represents the same ROI as figure 3

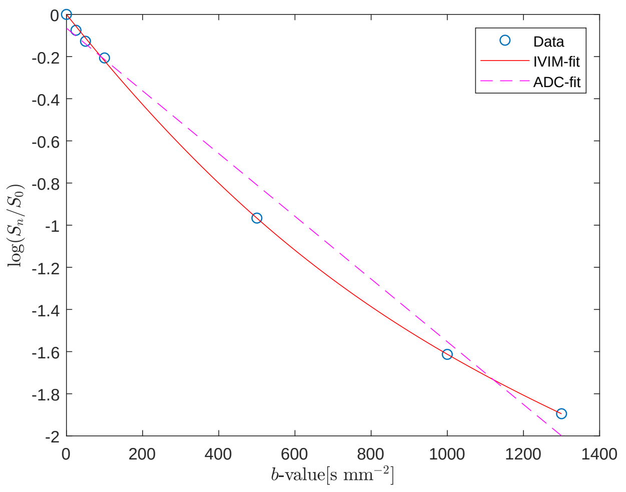


Figure 6: Fit of the logarithm of signal intensity versus b-value for a region of interest in a patient with rectal cancer. The improved fit given by the IVIm model suggested in equation (15) is shown as a solid line whereas the ADC fit is shown as a dashed line. The figure represents the same ROI as figure 4

2.3 Machine Learning

2.3.1 Introduction

Machine learning is steadily finding more and more applications as methods and hardware are developed and improved. Machine learning can be defined as any algorithm improves at performing some task, T, as measured by some performance metric, P, by learning from experience, E [36]. Thus, anything from the rather simple least squares linear regression to the much more complex deep learning methods can be called machine learning. Most machine learning models fall into one of two categories: supervised learning and unsupervised learning. In supervised learning the data is labelled before the model is trained [37]. That is, for each observation of the predictor measurement(s) $x_i, i = 1, \dots, n$ there is an associated response measurement y_i. The goal of supervised learning is to deduce a functional relationship that relates the predictor measurements to the response [38, 37]. The performance of a supervised machine learning algorithm is measured as its generalizations ability, that is, its capability to make correct predictions on previously unseen data [39].

In unsupervised learning we lack the response variable. The goal of an unsupervised machine learning algorithm is to discover interesting aspects of the measurements [40]. One way of doing so is to try to generate labels that organize the data in meaningful way [37]. The accuracy for supervised machine learning algorithms can be assessed having it make predictions on previously unseen data, but where labels are still known. Accuracy can thus be quantified using various performance measures comparing the predicted response from the algorithm to the ground truth. No such ground truth labeling is known when using unsupervised learning, and thus we have no such direct measure of success [41]. This leads the exercise of applying unsupervised machine learning techniques to be more subjective in character [40].

Variables can be classified as either qualitative (also called categorical) and quantitative variables. Quantitative variables take numerical values whereas qualitative variables take on one of K different classes, or categories. It is useful to distinguish classes of supervised models based on the type of their response variable. Regression models map the input space to numerical values, $y_i \in \mathbb{R}$. Classification models, on the other hand, map the input space to one of K pre-defined classes, $y_i \in \mathcal{G}$ [42].

2.3.2 Model selection - The bias variance tradeoff

An important task in machine learning is figuring out which model to use given a certain dataset. Suppose we have a functional relationship between predictor values $\boldsymbol{X_1}, \boldsymbol{X_2}, \dots, \boldsymbol{X_p}$ and the response variable, $\boldsymbol{Y}$, given by

$$\boldsymbol{Y} = f(\boldsymbol{X}) + \epsilon \tag{16}$$

where f is some fixed unknown function representing the systematic information that $\boldsymbol{X}$ yields about $\boldsymbol{Y}$ and ϵ is a random error term that is independent of $\boldsymbol{X}$ and has a mean of zero. With out machine learning algorithm we seek find and estimate for f, denoted $\hat{f}$, that yields accurate predictions on $\boldsymbol{Y}$. Denoting the the predictions on $\boldsymbol{Y}$ by $\hat{Y}$, and since the error term averages to zero, we can predict $\boldsymbol{Y}$ using

$$\hat{\boldsymbol{Y}} = \hat{f}(\boldsymbol{X}). \tag{17}$$

In this equation $\hat{f}$ represents our machine learning algorithm. In a regression setting the most common measure of the goodness of fit is the *mean squared error* (MSE), given by

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2, \tag{18}$$

where $\hat{f}(x_i)$ is the prediction our model $\hat{f}$ gives for the ith prediction. It can be shown that the expected test MSE, $E\left(y_0 - \hat{f}(x_o)\right)^2$ can be decomposed into three terms: the variance of $\hat{f}(x_0)$, the squared bias of $\hat{f}(x_0)$ and the variance of the error term ϵ, as shown in equation (19).

$$E\left(y_0 - \hat{f}(x_o)\right)^2 = \text{Var}(\hat{f}(x_0) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\epsilon) \tag{19}$$

From (19) shows that in order to minimize the mean squared error on the test data we seek a model that simultaneously has low bias and variance. Bias, in this context, is the accuracy of the model across different possible training sets. The variance component refers to the sensitivity of the learning algorithm to small changes in the training data [43]. Even thought the discussion so far has been around the regression setting, the concept of bias-variance trade-off transfers classification with minor adjustments due to the fact that the response no longer is quantitative.

It can be shown that classifier that is associated with the lowest test error rate, on average, is a classifier that assigns each observation to the class that is most likely given its predictor values. This is simply assigning each test observation to class $j \in \mathcal{G}$ such that

$$\arg\max_{j\in\mathcal{G}} \Pr(\boldsymbol{Y} = j | \boldsymbol{X} = x_0). \tag{20}$$

When working with real data however, we do not know the conditional distribution of $\boldsymbol{Y}$ given $\boldsymbol{X}$, and so we cannot compute the Bayes classifier. Many machine learning methods try to estimate the conditional distribution of $\boldsymbol{Y}$ given $\boldsymbol{X}$ through various assumptions. As is summarized *No Free Lunch Theorem* we cannot expect one classification algorithm to outperform another without making prior assumptions on the data. That is there is no "best" algorithm on for all problems and contexts [44].

2.3.3 Principal Component Analysis

Principal Component Analysis (PCA) is an unsupervised dimensionality reduction algorithm in which principal component of the data is calculated and used to understand the data [40]. A variety of dimensionality reduction techniques have been proposed, but PCA remains one of the most widely used [45, 46]. The idea behind PCA is to reduce the number of dimensions of the dataset, while preserving as much 'variability' (i.e statistical information) as possible [45].

Assume we have a dataset with n observation on p numerical variables. This dataset define p n-dimensional vectors $\boldsymbol{X}_1, \boldsymbol{X}_2 \dots \boldsymbol{X}_p$. This can equivalently be represented as the $n \times p$ matrix $\boldsymbol{X}$ whose column vectors $\boldsymbol{X_j}$, $j \in \{1, \dots p\}$ are the measurements on the jth variable. In PCA we seek the linear combination of the columns in the data matrix $\boldsymbol{X}$ that has the largest variance. Mathematically, this can be written as

$$\arg\max_{\boldsymbol{a}}(\mathrm{var}(\boldsymbol{X}\boldsymbol{a})) = \arg\max_{\boldsymbol{a}}(\boldsymbol{a}^T\boldsymbol{\Sigma}\boldsymbol{a}) \tag{21}$$

where $\boldsymbol{X}\boldsymbol{a}$ is matrix notation for the linear combination of columns of $\boldsymbol{X}$, $\sum_{j=1}^{p} a_j \boldsymbol{X}_j$, and $\boldsymbol{a}$ is a coefficient vector with coefficients $a_1, a_2, \dots a_p$ commonly called loadings. $\boldsymbol{\Sigma}$ is the sample covariance matrix of the dataset. For this problem to have a well defined solution an additional restriction must be imposed. The most common restriction involves working with unit norm vectors $\boldsymbol{a}$ such that $\boldsymbol{a}^T\boldsymbol{a} = 1$. Using the method of Lagrange multipliers, denoting the multiplier by λ, the problem is equivalent to finding

$$\arg\max_{\boldsymbol{a}}(\boldsymbol{a}^T\boldsymbol{\Sigma}\boldsymbol{a} - \lambda(\boldsymbol{a}^T\boldsymbol{a} - 1)) \tag{22}$$

Differentiating with respect to $\boldsymbol{a}$ and equating to the null vector yields the simple form of the solution

$$\boldsymbol{\Sigma}\boldsymbol{a} = \lambda\boldsymbol{a}. \tag{23}$$

Thus, since λ is a multiplicative constant, and $\boldsymbol{a}$ is a unit-norm vector, the vector $\boldsymbol{a}$ that maximizes the variance of the linear combination $\boldsymbol{X}\boldsymbol{a}$ are the eigenvectors of the sample covariance matrix. Since the eigenvalues are the variances of the linear combinations defined by the corresponding eigenvector,

$$\mathrm{var}(\boldsymbol{X}\boldsymbol{a}) = \boldsymbol{a}^T\boldsymbol{\Sigma}\boldsymbol{a} = \lambda, \tag{24}$$

we see that specifically the largest eigenvalue, denoted λ_1, and corresponding eigenvector, denoted $\boldsymbol{a_1}$, maximize the variance of the linear combination $\boldsymbol{X}\boldsymbol{a}$.

Using the method of Lagrange multipliers to successively maximize the variance of the linear combination $\boldsymbol{X}\boldsymbol{a}_i$ with the added restriction of orthogonality of different coefficient vectors one can find that the full set of eigenvectors of $\boldsymbol{\Sigma}$ are solutions. These solutions can also be shown to maximize '*uncorrelatedness*' since the covariance between two resulting linear combinations $\boldsymbol{X}\boldsymbol{a}_i$ and $\boldsymbol{X}\boldsymbol{a_k}$ is given by $\boldsymbol{a}_i^T\boldsymbol{\Sigma}\boldsymbol{a}_k = \lambda_k\boldsymbol{a}_i^T\boldsymbol{a}_k = 0$ if $i \neq k$ since $\boldsymbol{a}_i$ and $\boldsymbol{a}_k$ are orthogonal with unit norm. The resulting linear combinations $\boldsymbol{X a_i}$, $i \in \{1, 2, \dots p\}$ are called principal components (PC) of the dataset. The elements of these linear combinations are called PC scores and the elements of the eigenvectors $\boldsymbol{a}_i$, $i \in \{1, 2, ..., p\}$ are called PC loadings.

Since our solutions were the eigenvectors of the sample covariance matrix $\boldsymbol{\Sigma}$, an alternative approach can be employed by the use of the *Singular Value Decomposition* (SVD). This alternative approach is employed in many computer algorithms performing PCA [47, 48]. SVD states that an $n \times m$ matrix $\boldsymbol{A}$ can be factored as

$$\boldsymbol{A} = \boldsymbol{U D V}^T \tag{25}$$

where $\boldsymbol{U}$ is an $n \times n$ orthogonal matrix, $\boldsymbol{V}$ is an $m \times m$ orthogonal matrix and $\boldsymbol{D}$ is an $n \times m$ diagonal matrix with non-negative entries (A $n \times m$ diagonal matrix has $\min(n, m)$ along the diagonal, and all other entries are zero) [49]. Multiplying the matrix $\boldsymbol{A}$ with its transpose from either side we obtain

$$\begin{aligned} \boldsymbol{A A}^T &= \boldsymbol{U D V}^T \boldsymbol{V D}^T \boldsymbol{U}^T = \boldsymbol{U D D}^T \boldsymbol{U}^T \\ \boldsymbol{A}^T \boldsymbol{A} &= \boldsymbol{V D}^T \boldsymbol{U}^T \boldsymbol{U D V}^T = \boldsymbol{V D}^T \boldsymbol{D V}^T. \end{aligned} \tag{26}$$

The right-hand side of these relations describe the eigenvalue decomposition of the left hand sides. We notice that the columns of $\boldsymbol{V}$ are the eigenvectors of $\boldsymbol{A}^T \boldsymbol{A}$, the columns of $\boldsymbol{U}$ are the eigenvectors of $\boldsymbol{A A}^T$ and that the non-zero elements of $\boldsymbol{D}$ are the square roots of the non-zero eigenvalues of $\boldsymbol{A}^T \boldsymbol{A}$ and $\boldsymbol{A A}^T$.

If the PCs are redefined in terms of the centered variables $\boldsymbol{X_j^*} = \boldsymbol{X}_j - \bar{\boldsymbol{X}}_j$, where $\bar{\boldsymbol{X}}_j$ is the expectation value of the variable $\boldsymbol{X}_j$ then the sample covariance matrix gets the simple form

$$(n-1)\boldsymbol{\Sigma} = \boldsymbol{X}^{*T} \boldsymbol{X}^*. \tag{27}$$

The centering does not change the solution as the covariance matrix of the centered solution will remain the same.

The principal components of set of data with p-variables provides the best sequence of linear approximations to that data, out of all ranks $q \leq p$ [41]. Computing the singular value decomposition of the data matrix $\boldsymbol{X}$ we see that by the results obtained in equation (26) the right side of (27) becomes

$$(n-1)\boldsymbol{\Sigma} = \boldsymbol{V D}^2 \boldsymbol{V}^T, \tag{28}$$

and thus the column vectors of $\boldsymbol{V}$ are the eigenvectors of $\boldsymbol{\Sigma}$. Calculations of the PCA has now been reduced to computing the SVD of the centered data matrix $\boldsymbol{X}^*$. Rearranging the entries d_i, $i \in \{1, 2, \dots \min(m, p)\}$ such that $d_1 \geq d_2 \geq \cdots \geq d_{min(m,p)}$ in $\boldsymbol{D}$ and rearranging the columns of $\boldsymbol{U}$ correspondingly the solution remains valid. Equation (23) relates the eigenvalues and eigenvectors of the covariance matrix to the variance of the linear combination $\mathrm{var}(\boldsymbol{X a})$

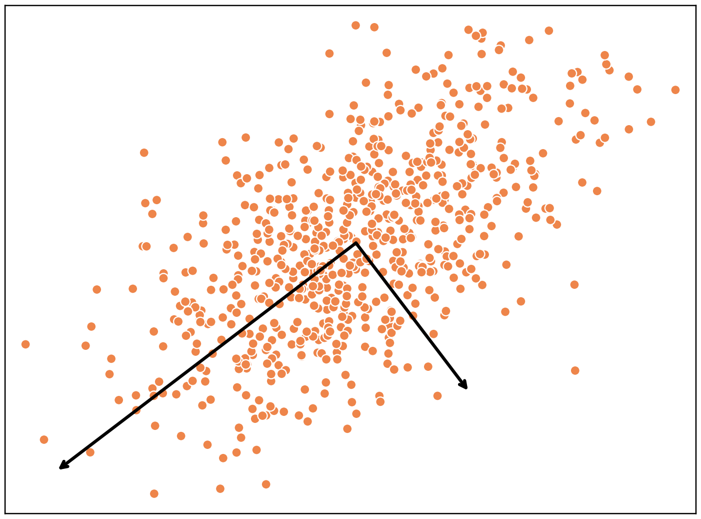


Figure 7: The principal components drawn as vector on samples drawn from a multivariate normal distribution. The first principal component is the longest of the vectors as it points in the direction of greatest variance in our data. The figure inspired by the book [50] .

we seek to maximize. The eigenvalues, λ_i were found to be the square of the diagonal elements of the matrix $\boldsymbol{D}$. Since the elements of the diagonal matrix $\boldsymbol{D}$ now are rearranged into columns of descending value from left to right, and the columns of $\boldsymbol{V}$ is rearranged accordingly, the first eigenvalue will corresponds to the eigenvector maximizing the variance. This eigenvecotor is the first column of the matrix $\boldsymbol{D}$. The preceeding eigenvectors correspond to the vectors that maximize the variance of the linear combination $\text{var}(\boldsymbol{X}\boldsymbol{a}_i)$ under the condition that coefficient vector $\boldsymbol{a_i}$ is orthogonal to all coefficient vectors corresponding to larger λ_i.

Finding the g principal components that captures the most variance in our data matrix now amounts to selecting the first g columns of the matrix $\boldsymbol{D}$. This can be shown to be the best g-dimensional approximation of the original data matrix in terms of least squared differences [45]. The principal components of a synthetic dataset where samples are drawn from a multivariate normal distribution are shown in figure 7. The quality of a PCA solution can be given as the proportion of total variance explained by selected principal component. Using relation between the variance of a principal component and eigenvalues of the covariance matrix from equation (23) the variance explained by the principal component j is $\lambda_j / \sum_{j=1}^{p} \lambda_j$, and the variance explained by the the first g principal components is thus $\sum_{i=1}^{g} \lambda_j / \sum_{j=1}^{p} \lambda_j$

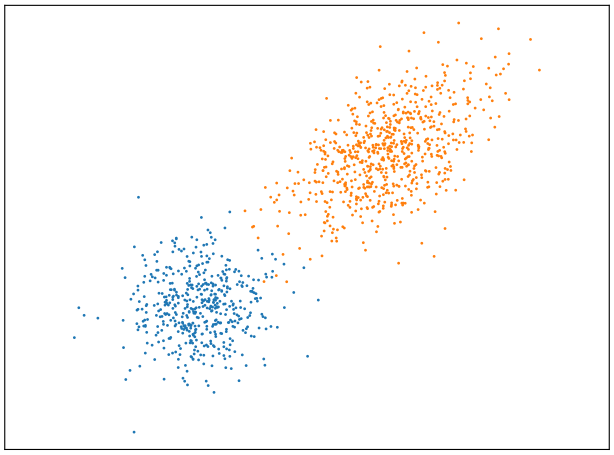


Figure 8: Scatter plot of samples drawn from two multivariate normal distributions with different mean and covariance matrices.

2.3.4 Cluster analysis

Cluster analysis is summarized by Jain [15] as the formal study of methods and algorithms for grouping, or clustering, of objects according to measured or perceived characteristics of similarity. The lack of predetermined labels distinguish clustering from classification. The goal of clustering is not to find and accurate probabilistic description of the data, rather to find hidden data structures or patterns in the data [51].

Clustering is useful if we can find a small number of groups of objects yielding a concise description of the similarities and differences observed in our data set [46]. Numerous methods have been proposed for this task. Since there is no direct measure of success of a clustering algorithm, the performance of a clustering method should perhaps be judged based on its usefulness in the given scenario [46].

Numerous different clustering algorithm have been proposed separating the data based on different measures of similarity. No single one of these algorithms that will perform better than all others across all datasets, and even the same algorithm with different selection of parameters may cause completely different clustering results [51]. Example of two datasets that may require different clustering strategies are shown in figures 8 and 9. In the figure 8 the data samples are drawn from two multivariate normal distributions with different mean and covariance matrix. In the figure 9 positions along the circular arch of a circle is drawn from a uniform distribution. The radii are then drawn from a set containing two radii with equal probability of drawing either one. Some Gaussian noise is superimposed to the radii.

The procedure of evaluating the results of a clustering algorithm is known

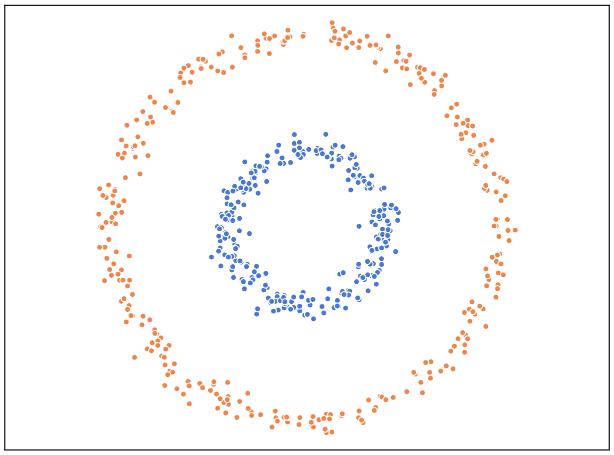


Figure 9: Scatter plot of radially separated clusters. For each point a radius is randomly drawn from a population of two radii at equal probabilities. Gaussian noise is added the the radius value. The placement along the circle arch is drawn from a uniform random distribution.

as cluster validity [52]. When performing clustering validity it is sometimes indicated that there are no justifiable cluster is the dataset. This is of course also an acceptable answer [46]. In the following subsection we will discuss two common clustering algorithms, as well as some criteria used to assess the validity clustering solution given a particular set of parameters.

2.3.5 k-means Clustering

The following section is loosely based on the article *Data clustering: 50 years beyond K-means* by Jain et al. [15].

The k-means clustering algorithm was proposed more than 50 years ago, but is still to this day one of the most widely used clustering algorithms. Its continued success can be attributed to its ease of implementation, simplicity, efficiency and empirical success across a wide range of scientific fields [15]. The algorithm, as the names implies, clusters the data samples into k non-overlapping clusters. Suppose we have a dataset, $\boldsymbol{X}$, consisting of n d-dimensional observations $\boldsymbol{x}_i$, $i \in \{1 \dots n\}$. We let these points be clustered into a set of k clusters C, denoting each cluster by c_j, $j \in \{1, \dots, j\}$. The classic k-means algorithm seeks to minimize the loss function $J(C)$ defined as the sum of the squared distance between each data sample, $\boldsymbol{x}_i$, and the cluster center to to which it is partitioned. Denoting the cluster center of the jth cluster as $\boldsymbol{\mu}_j$ this amounts to minimizing

$$J(C) = \sum_{j=1}^{k} \sum_{x_i \in c_n} ||\boldsymbol{x}_i - \boldsymbol{\mu}_j||^2 \tag{29}$$

The algorithm does so by performing the following sequence of steps:

1. Initialize a k-partition randomly or based on some prior knowledge.
2. Generate a new partition by assigning each sample to its closest cluster center.
3. Compute new cluster centers.
4. Repeat steps 2 and 3 until cluster membership stabilizes.

If convergence is found, the k-means algorithm is not guaranteed to have converged to the global minimum of $J(C)$ as cluster membership may also stabilize at a local minimum of $J(C)$. At this point it is worth noting that since $J(C)$ always decreases with an increasing number of clusters, and is by definition equal to zero when $k = n$, it only makes sense to minimize $J(C)$ for a fixed number of clusters k. The user must thus supply the algorithm with a suitable k, which is the most critical parameter of the algorithm.

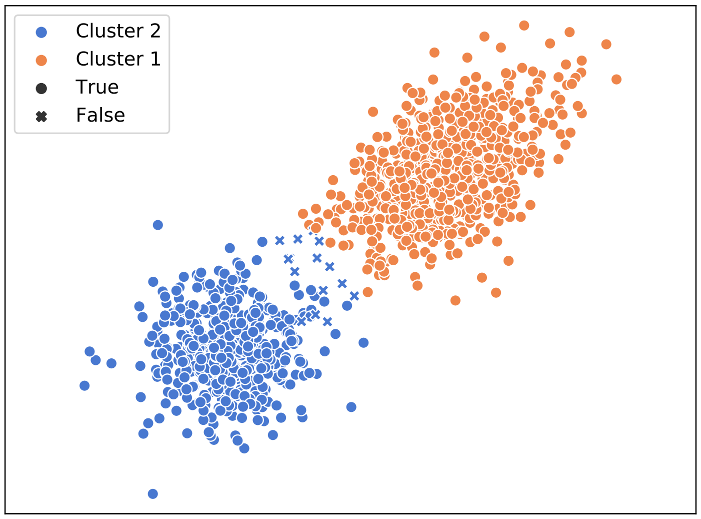
Using the classical k-means algorithm on the synthetic data from figure 8 we can see the effect of selecting k erroneously. Selecting the correct k yields the clustering result shown in figure 10a, where only few samples are assigned to the wrong cluster. Selecting $k = 3$ in yields the clustering shown in figure 10b. A large number of the samples in the cluster that is least spherical are now erroneously assigned.

The loss function defined in equation (29) uses the Euclidean distance between points and their cluster centers. As a consequence, the resulting algorithm will find spherically shaped clusters in the data [15]. A more generalized description of the k-means algorithm that allows for different distance measures to be used in the loss function is presented by Kashima et al. [53].

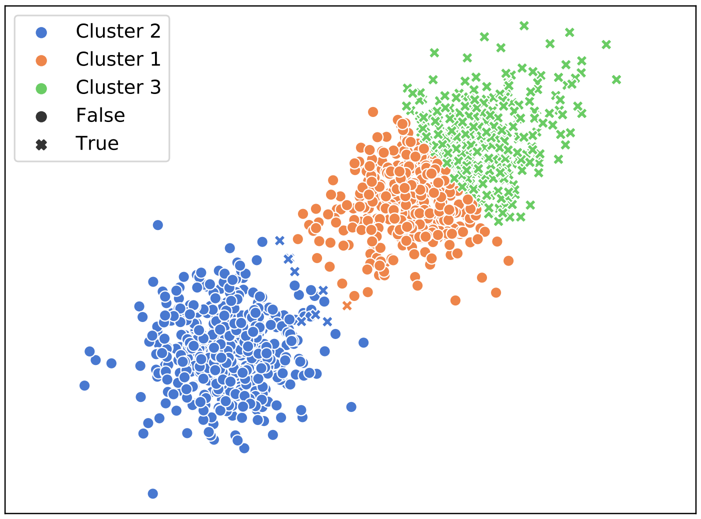
2.3.6 Finite Mixture Densities - Gaussian Mixture Models

The following section is based primarily on the book *Cluster Analysis* by Everitt et al. [46].

The k-mean algorithm considered thus far makes no explicit assumptions on the patterns of the data samples. This does not mean that no such assumptions have been made, but it differentiates them from the family of method discussed in this section. In finite mixture density models a formal statistical model is postulated for the populations from which the samples are drawn. Each sample is assumed drawn form a cluster/sub-population. Each of these clusters are assumed to have different multivariate probability density functions, resulting in what is called finite mixture densities for the population as a whole [46]. The



(a) k is set to 2



(b) k is set to 3

Figure 10: k-means clustering on a synthetic dataset consisting of samples drawn from two multivariate normal distributions with different means and covariance matrices. The colors represent which cluster each sample is assigned to by the k-means algorithm. The marker style represent whether the given assignment is correct.

clustering problem is now estimating the parameters of the probability density functions from which that samples are assumed to have been drawn.

Suppose $\boldsymbol{x}$ is a p-dimensional random variable drawn from a dataset $\boldsymbol{X}$ consisting of the set of clusters $\mathcal{C} = \{c_1, c_2, \ldots c_k\}$. Each cluster is allowed to have its own probability density function (PDF), g_j, $j \in \{1, \ldots, k\}$, parameterized by $\boldsymbol{\theta}_j$. Let the probability that a sample is drawn from cluster c_j, be denoted by p_j. By the law of total probability, the PDF of the random variable $\boldsymbol{x}$ is given by

$$f(\boldsymbol{x}; \boldsymbol{p}, \boldsymbol{\theta}) = \sum_{j \in \mathcal{C}} p_j g_j(\boldsymbol{x}; \boldsymbol{\theta}_j) \tag{30}$$

where $\boldsymbol{p}^T = (p1, p2, \ldots, p_k)$ and $\theta^T = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_k)$.

Finite mixture are suitable models for cluster analysis if we can assume that the dataset is sampled from a set of clusters with different probability distributions. The probability distributions of the cluster may come from the same family and differ only in the parameter values, or be altogether different PDFs.

Denoting the estimated parameters by $\hat{\boldsymbol{p}}_j$ and $\hat{\boldsymbol{\theta}}$ we can by Bayes theorem estimate the posterior probability that a sample belongs to the cluster c_j as

$$\Pr(c_j | \boldsymbol{x}_i) = \frac{\hat{p}_j g_j(\boldsymbol{x}_i; \hat{\boldsymbol{\theta}}_j)}{f(\boldsymbol{x}_i; \hat{\boldsymbol{p}}, \hat{\boldsymbol{\theta}})}. \tag{31}$$

A sample will be assigned to the cluster in $\mathcal{C}$ that maximizes the posterior probability function.

Now suppose that the samples $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_n$ are samples drawn from the PDF in equation (30) with a given set of mixture densities with set parameters. We'll denote this specific model by $\mathcal{M}$. Equation (30) then represents the probability of that one sample $\boldsymbol{x}_i$ is drawn from the model of probabilistic clusters $\mathcal{M}$, $\Pr(\boldsymbol{x}_i | \mathcal{M})$. The samples are assumed to be generated independently and thus the probability that a given set of samples $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_n\}$ is generated by $\mathcal{M}$ is

$$\Pr(\boldsymbol{X} | \mathcal{M}) = \prod_{1=1}^{n} \Pr(\boldsymbol{x}_i | \mathcal{M}) = \prod_{1=1}^{n} \sum_{j=1}^{k} p_j g_j(\boldsymbol{x}_i; \boldsymbol{\theta}_j). \tag{32}$$

It should now be clear that the task of finite mixture densities on the dataset $\boldsymbol{X}$, is to find a set of clusters $\mathcal{C}$, parameterized by $\boldsymbol{\theta}$ and with mixture proportions $\boldsymbol{p}$, that maximizes the probability $\Pr(\boldsymbol{X} | \mathcal{M})$ [54].

The probability function obtained in equation (32) is the *likelihood function* of $\mathcal{M}$ [55]. For machine learning purposes the likelihood function is often converted to the *log-likelihood* function, l, by taking the natural logarithm

$$l(\boldsymbol{p}, \boldsymbol{\theta}) = \sum_{i=1}^{n} \ln \sum_{j=1}^{k} p_j g_j(\boldsymbol{x}_i; \boldsymbol{\theta}_j) = \sum_{i=1}^{n} \ln f(\boldsymbol{x}_i; \boldsymbol{p}, \boldsymbol{\theta}) \tag{33}$$

In cases where the likelihood and log-likelihood functions are too complicated to employ the normal methods of minimization an iterative approach is used. Perhaps the most common iterative approach is know as the *expectation-maximization* (EM) algorithm [46, 56]. The EM algorithm starts by making an initial partition of the dataset as well as making guesses on the additional parameters. Then it iterates trough the following two steps until the clustering converges, or the change in the log-likelihood/likelihood function sufficiently small [54, 57]:

1. Expectation step: each sample is assigned to the cluster for which the posterior probability, given in equation (31), is the largest.

2. Maximization step: the probability estimates from above is used to reestimate the parameters of each cluster such that the expected likelihood is maximized.

If we restrict the PDF of each cluster to a multivariate normal distribution where the jth cluster has a mean vector $\boldsymbol{\mu}_j$ and a convariance matrix $\boldsymbol{\Sigma}_j$ maximum likelihood yields the following parameter estimates

$$\hat{p}_j = \frac{1}{n}\sum_{i=1}^{n} \Pr(c_j|\boldsymbol{x}_i) \tag{34}$$

$$\hat{\boldsymbol{\mu}}_j = \frac{1}{n\hat{p}_j}\sum_{i=1}^{n} \boldsymbol{x}_i \Pr(c_j|\boldsymbol{x}_i) \tag{35}$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{1}{n}(\boldsymbol{x}_i - \boldsymbol{\mu}_j)(\boldsymbol{x}_i - \boldsymbol{\mu}_j)^T \Pr(c_j|\boldsymbol{x}_i) \tag{36}$$

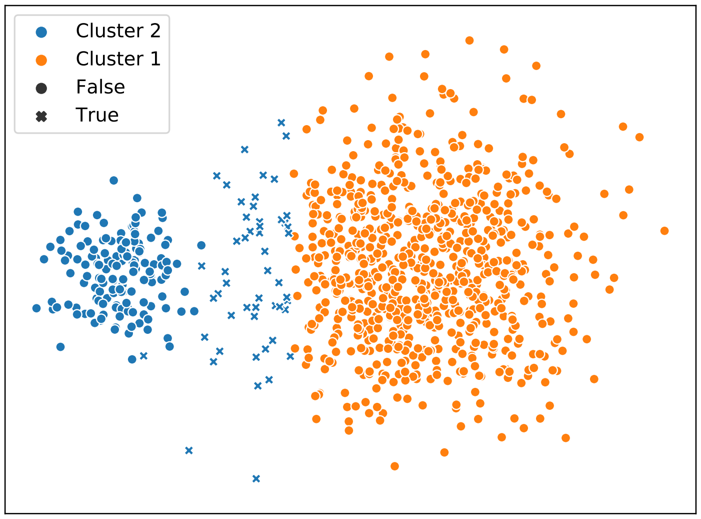
Each of these parameters are assigned for each maximization step. These values are again used to reestimate the posterior probability $\Pr(c_j|\boldsymbol{x}_i)$ before the next iteration in the algorithm.

The expectation maximization algorithm is simple and easy to implement. It should be noted that it doesn't in general converge to a global maximum, and may instead converge to a local maximum [54]. This may be partly offset by running multiple duplicates with different initial parameters and comparing their log-likelihood/likelihood values.

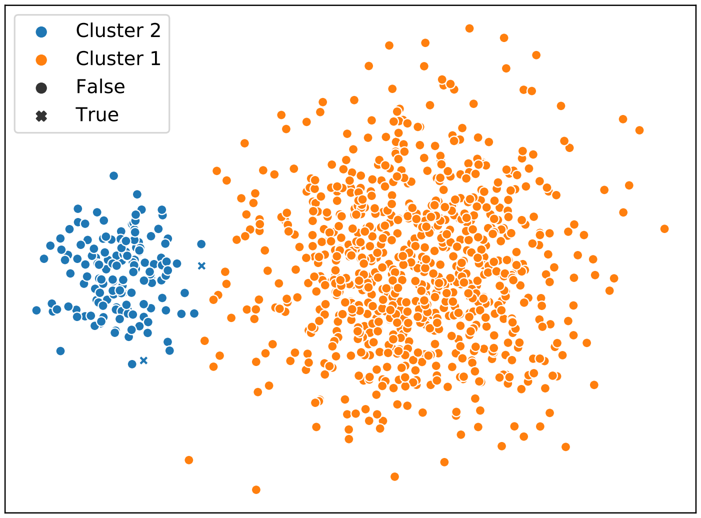
In the following the clusters will be assumed to have multivariate Gaussian/normal densities, and the resulting models will be referred to, as they are in many common software packages, as *Gaussian Mixture Models* (GMM) [58, 59]. Fitting a GMM to the data in figure 8 we obtain the clustering shown in figure 13. Comparing the clustering results to that of the k-means algorithm presented in 10a we see that the GMM performs better in areas where the clusters intersect/overlap. Compared to the k-means algorithm the GMM are more flexible. It can be shown the GMM where each cluster is assumed to have an equal diagonal/spherical covariance matrix $\sigma\boldsymbol{I}$ turn into a probabilistic k-means algorithm [41].

This highlights some key differences between the clusters found by the k-mean algorithm and the GMM. The k-means algorithm will works best with spherical clusters of equal extent. That is, it implicitly assumes that each cluster in the data has the same diagonal covariance matrix and that only the position of the cluster center differs. A comparison of the clustering performance of the k-means algorithm on two clusters drawn from multivariate normal distributions with different covariance matrices $\sigma_1 \boldsymbol{I}$ and $\sigma_2 \boldsymbol{I}$ where $\sigma_1 \neq \sigma_2$ is shown in figure 11. The differences between the algorithms are even more visible when the covariance matrices of the clusters no longer can be assumed to be spherical. An example of this is shown in figure 12. Another notable deficit of the k-means algorithm compared to the GMM is when the density of each cluster is different [60]. This is shown in figure 14 where the covariance matrices of the clusters are equal and spherical, but the probability of the samples being drawn from each of the clusters is different.

Up until this point the notation *clusters* have been used to denote each of the multivariate normal distributions resulting from the GMM clustering solution. In cases where cluster samples cannot be assumed to have purely a multivariate normal PDF, multiple multivariate normal distributions may be needed to represent a cluster adequately [61]. To refer to the multivariate normal distributions as clusters can thus be a bit misleading and lead to an overestimation for the number of clusters in the data. Thus, from here on, each multivariate normal distribution resulting from GMM clustering solution will be referred to as a component rather than a cluster.

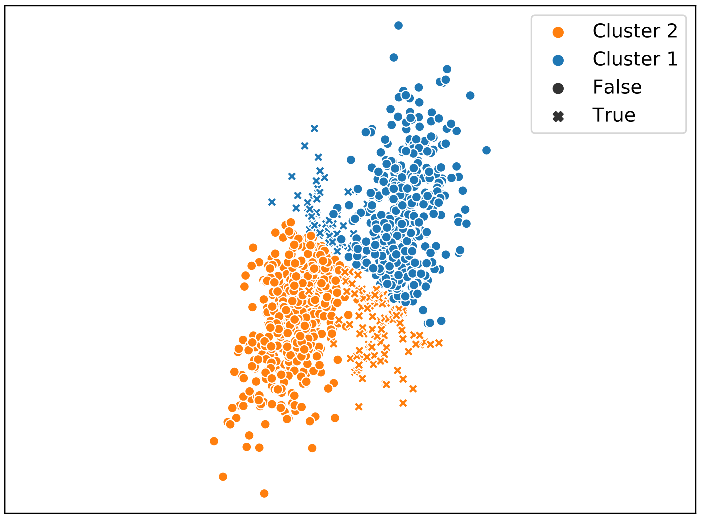


(a) k-means clustering

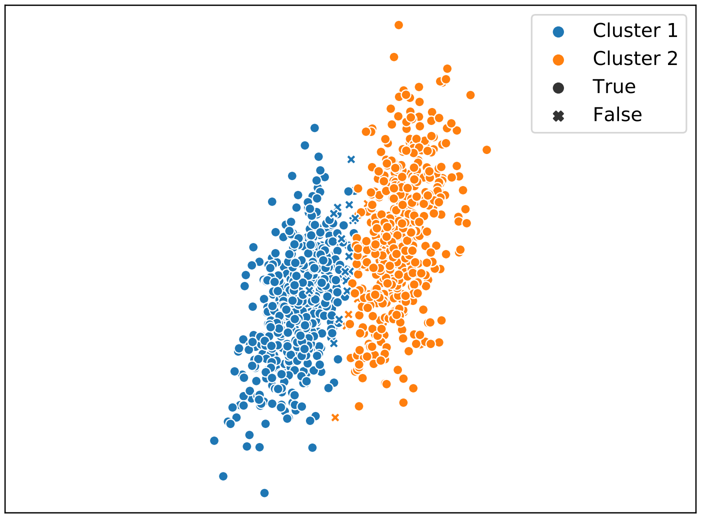


(b) GMM clustering

Figure 11: Clustering with k-means and GMM of samples drawn from two multivariate normal distributions with spherical covariance matrices. The density of samples is made equal within each cluster.



(a) k-means clustering



(b) GMM clustering

Figure 12: Clustering with k-means and GMM of samples drawn from two elongated multivariate normal distributions.

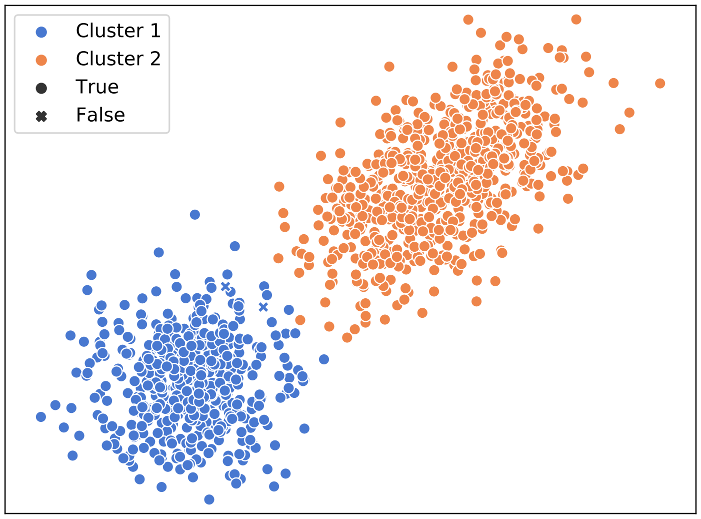
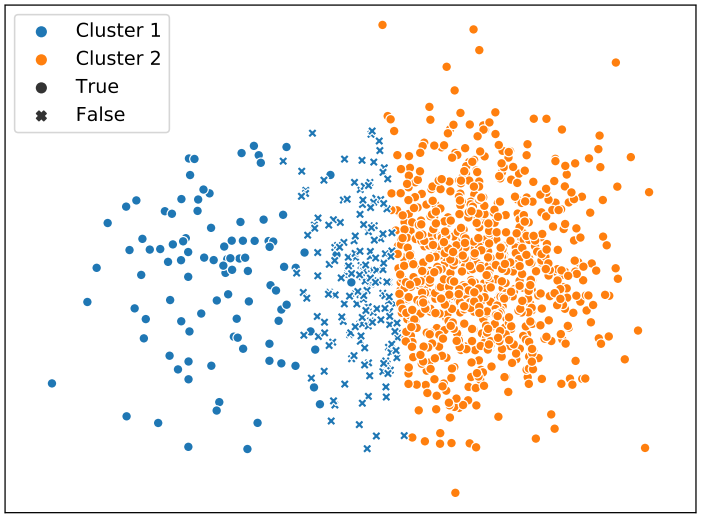
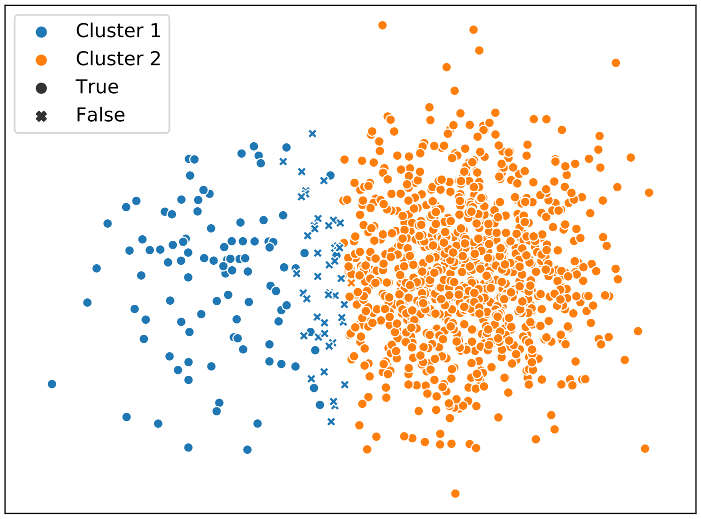


Figure 13: Gaussian mixture model fitted to a dataset drawn from two multivariate normal distributions.



(a) k-means clustering



(b) GMM clustering

Figure 14: Clustering with k-means and GMM of samples drawn from two multivariate normal distributions with equal spherical covariance matrices. The probability of drawing a sample from the orange cluster is greater than that of the blue leading to different sample density within each cluster.

2.4 Cluster validity

2.4.1 Visually uncovering structures in data

The procedure of evaluating the results obtained by a clustering algorithm is know as cluster validity [52]. Both k-means and GMM take as input the number of clusters in the data. This means that they will return a partitioned dataset into the selected number of clusters whether or not there are any reasons to believe there are any clusters present in the data. A strong apriori hypothesis should thus be held that clusters exists in the data for clustering to be effective. Graphical displays of the data can be useful in suggesting that data contain substantial clustering [46]. In the case of multidimensional data effective visualization becomes difficult [52]. A tool for visualizing multidimensional data is the so called scatter plot matrix. This is a matrix p by p grid of scatter plots where in row i column j the scatter plot of the i-th variable is plotted against the j-th. Since a variable plotted against itself give little additional insight the histogram of each variable is often plotted along the diagonal.

When the number of samples in the dataset becomes large, the structures in the data may be difficult to uncover using the simple scatter plot. This is due to the fact that many samples now are likely to overlap making it hard to get a sense of density. This can be solved by the used of *kernel density estimates* (KDE) [62, 63]. The KDE is a non-parametric density estimate. Assuming we have some data samples $X_1, X_2, \dots X_n$ drawn from an unknown PDF, f, the univariate KDE is defined as

$$\hat{f}(x,h) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \tag{37}$$

where h is a real positive number called the bandwidth and K is the so called kernel function. A number of different function can be taken to be kernel functions as long as they satisfy some basic requirements [64, p. 26]. However, the most common kernel the normal distribution

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}. \tag{38}$$

The kernel estimator can understood as the sum of 'bumps' placed at the observations, where the kernel function determines the shapes of the bumps and the bandwidth determined their width [46]. The univariate kernel densities can be generalized to a d-dimensional multivariate kernel density estimator given by

$$\hat{f}(\boldsymbol{x}, \boldsymbol{H}) = n^{-1}\sum_{i=1}^{n} |\boldsymbol{H}^{-1/2}| K(\boldsymbol{H}^{-1/2}(\boldsymbol{x} - \boldsymbol{X}_i)) \tag{39}$$

where $\boldsymbol{H}$ is the $d \times d$ bandwidth matrix or smoothing matrix, and $\boldsymbol{X}_i$ are the d-dimensional observations [64]. There are a number of ways selecting a suitable bandwidth, known as *bandwidth selectors*, for the kernel functions. However, these are outside the scope of this report. A review of the most popular methods are given in the book *Nonparametric Kernel Density Estimation and Its Computational Aspects* by Artur Gramacki [64].

2.4.2 Cluster Validaty Indices (CVI)

Many clustering algorithms are by themselves not able to determine the number of cluster naturally occurring in the data, and therefore the number of clusters must be supplied to the algorithm by the user. However, information about the number of clusters in the data is rarely know a priori. The usual strategy is running the clustering algorithm several times with a different number of proposed clusters and evaluating the clustering solutions obtained [65]. The process of evaluating the results of a clustering algorithm is known by the term cluster validity [52]. Several mathematical criteria, known as *Cluster Validity Indices*, have been proposed as measures of goodness of clustering solutions [66, 67, 68, 69, 70, 71, 72].

Two of the main categories of CVI are internal and external clustering validation [73]. The difference between the two is wether it relies on information not contained in the dataset or not. External criteria rely on a priori knowledge about the clustering structure of the dataset [52]. Internal clustering criteria seek to find the optimal clustering solution without any additional information, or any preconceived knowledge about the structures of the clusters in the dataset. Thus, in situations where no external information is available, internal validation measures are the only options for cluster validation measures [73].

There are two widely accepted criteria for partitioning a dataset used in most CVIs [52]: compactness - a measure of closeness of elements within the same cluster and separability - a measure of how different each cluster are from one another [71]. The best clustering solution for a given dataset will provide clusters where elements within each cluster are as similar as possible, while keeping them as different as possible from elements of a different cluster.

The most common CVIs for crisp/hard clustering, that is clustering where partial cluster membership aren't allowed, are the *Calinski and Harabasz index* (CH) [66], *Davies and Bouldin index* (DB) [68] and the *Silhouette Coefficient* (Sil) [70] [74]. The more recently proposed *Gap Statistic* (GS) [69] has also seen quite frequent use.

A large number of publications have been made on the comparison of CVIs on different synthetic and real datasets [72, 71, 73, 75]. A recent, extensive study conducted by Arbelaitz et al compared the performance of 30 CVIs in 720 synthetic and 20 real datasets using the three clustering algorithms k-means, average-linkage and ward-linkage. In this study it was found that the CVIs could be subdivided into three groups according to performance on the datasets. Amoung other indices, DB, CH and Sil were found in the best performing group. In addition, a variation on the DB index, the DB*, was found to perform well. DB* was proposed by Kim and Ramakrishna in 2005 [71] together with adaptions of a few other commonly seen CVIs. In their work they found that DB* outperformed the traditional DB index. They also proposed a further revision, DB**, which was shown to outperform both the DB and DB* index. The clustering indices were tested on 11 datasets of which seven were synthetic and four real.

Liu et al. [73] compared the performance of 11 CVIs across 5 different aspects: monotonicity, noise, density, subclusters and skewed distributions, and found that the index S_Dbw [76] was the only index that performed well across all aspects [73]. However, in the extensive testing performed by Arbeilaitz et al, it was outperformed by Sil, DB, DB* and CH. It is also quite computationally expensive [71] and thus will no longer be considered in this thesis.

Underneath follows the mathematical definition of some of the better performing aforementioned CVIs. The up arrow (↑) signifies that the criterion is to be maximized and down arrow (↓) signifies that it is to be minimized. The following notation will be used. Consider an F-dimensional dataset $\boldsymbol{X} = \{x_1, x_2 \dots x_N\}$. A cluster solution of $\boldsymbol{X}$ will partitions $\boldsymbol{X}$ into K disjoint clusters: $C = \{c_1, c_2 \dots c_K\}$, where each assigned to exactly one cluster. The centroid of a cluster is its mean vector $\bar{c_k} = 1/|c_k| \sum_{x_i \in c_k} x_i$ and likewise the centroid of the whole dataset is $\bar{\boldsymbol{X}} = 1/N \sum_{x_i \in X} x_i$. The euclidean distance between sample x_i and x_j is denoted by $d(x_i, x_j)$.

Calinski-Harabarsz (CH↑) [66] The Calinski-Harabasz index evaluates the cluster validity using average distances of points within a cluster to its centroid as a measure of compactness, and separation between the cluster centroid to the centroid of the dataset as a measure of separability.

$$\mathrm{CH}(C) = \frac{N-K}{K-1} \frac{\sum_{c_k \in C |c_k| d(\bar{c_k}, \bar{\boldsymbol{X}})}}{\sum_{c_k \in C} \sum_{x_i \in c_k} d(x_i, \bar{c_k})} \tag{40}$$

Davies-Bouldin family of indices (DB, DB*, DB ↓) [68, 71]** The Davies-Bouldin is calculated by for each cluster c_i finding the maximum value of the ratio between the sum of average intracluster distances of c_i and c_j and the distance between the clusters $d(c_i, c_j)$ for all $j \neq i$. The DB index is then found by taking the average of the maximum values of this ratio for each i. That is

$$\mathrm{DB}(C) = \frac{1}{K} \sum_{c_k \in C} \left(\max_{c_l \in C \setminus c_k} \frac{S(c_k) + S(c_l)}{d(\bar{c_k}, \bar{c_l})} \right) \tag{41}$$

where $S(c_k)$ is the average intracluster distance $S(c_k) = 1/|c_k| \sum_{x_i \in c_k} d(x_i, \bar{c_k})$. The DB* index modifies the expression by using the minimum distance between a the cluster center $\bar{c_k}$ and any other cluster center in the ratio

$$\mathrm{DB*}(C) = \frac{1}{K} \sum_{c_k \in C} \left(\frac{\max_{c_l \in C \setminus c_k} \{S(c_k) + S(c_l)\}}{\min_{c_l \in C \setminus c_k} d(\bar{c_k}, \bar{c_i})} \right). \tag{42}$$

The DB** index adds an additional additive term called the $\mathrm{maxDiff}_i(K)$ to the ratio. To calculate the DB** for a clustering solution with K clusters, the clustering solution with $K+1$ clusters must also be found. Denote the average intracluster distance for cluster c_k in a clustering solution with K clusters as $S_{c_k}(K)$. The $\text{maxDiff}_k(K)$ is then defined as

$$\begin{aligned}\text{maxDiff}_k(K) = \max_{K_{max},\ldots,K} \max_{c_l \in C_K \setminus c_k} \{S_{c_k}(K) + S_{c_l}(K)\} - \\ \max_{c_l \in C_{K+1} \setminus c_k} \{S_{c_k}(K+1) + S_{c_l}(K+1)\}\end{aligned} \tag{43}$$

where C_K is the clustering solution with K clusters. The index then defined as

$$\text{DB}^{**}(C_K) = \frac{1}{K} \sum_{c_k \in C_K} \left(\frac{\max_{c_l \in C_K \setminus c_k} \{S_{c_k}(K) + S_{c_l}(K)\} + \text{maxDiff}_k(K)}{\min_{c_l \in C_K \setminus c_k} d(\bar{c_k}, \bar{c_l})} \right) \tag{44}$$

Silhouette coefficient (Sil ↑) [70] The silhouette coefficient uses a cohesion measure based on the average distance between all points in a cluster. As a measure of separability it uses the smallest distance from a sample in one cluster to any sample of any other cluster. Mathematically this is

$$\text{Sil}(C) = \frac{1}{N} \sum_{c_k \in C} \sum_{x_i \in c_k} \frac{b(x_i, c_k) - a(x_i, c_k)}{\max\{a(x_i, c_k), b(x_i, c_k)\}} \tag{45}$$

where

$$a(x_i, c_k) = \frac{1}{|c_k|} \sum_{x_j \in c_k} d(x_i, x_j) \tag{46}$$

$$b(x_i, c_k) = \min_{c_l \in C \setminus c_k} \left\{ \frac{1}{|c_l|} \sum_{x_j \in c_l} d(x_i, x_j) \right\}. \tag{47}$$

It should be noted that Sil is quite computationally expensive having quadratic time complexity in the number of samples as compared to the DB index that is linear in the number of samples [77].

2.4.3 Cluster validity in mixture models

In mixture models clustering using information theoretic methods has become increasingly popular [46] and has become a standard methodology [61]. The two most popular information criteria are the *Akaike information criterea* [78] (AIC) and the *Bayesian information criterion* [79] (BIC) [46]. Multiple studies have found that BIC works well in practice [80, 81], and that usually outperforms the AIC in assessing component membership in both real and synthetic datasets [82, 83].

The BIC is defined as

$$\text{BIC} = 2l(\hat{\boldsymbol{p}}, \hat{\boldsymbol{\theta}}) - d \ln n \tag{48}$$

where l is the maximized value of the log-likelihood function of the mixture model and d is the number of free parameters in the mixture model.

2.5 Survival Analysis

Classical survival analysis focuses on the time elapsed until a single event has occurred for a given patient [84]. Some examples of events, or endpoints, of interest within medical research may be death, occurrence of some disease or relapse of a malignancy. The reason why survival analysis requires special statistical models is that survival data usually include individuals where events have yet to occur [85]. Survival data thus constitute a mixture of complete and incomplete observations. The incomplete observations, where the event of interest has yet to occur, is termed censored survival times [85]. A patient is also considered to be censored if it drops out of the study before the end of the study period [86].

Two time dependent functions are of particular interest when analysis survival data, namely, the survival function, $S(t)$ and the hazard function $h(t)$ [86]. The survival function gives the expected proportion of patients for which the event of interest has yet to occur at time t [84]. Formally, this can be written as

$$S(t) = P(T > t), \tag{49}$$

where T is the random variable denoting the survival time. The survival function can also be interpreted as the probability of surviving until at least time t [86]. The hazard function, $h(t)$, is the conditional probability of dying at time t given that the patient has survived up until that time.

The graph of the empirical survival function can be useful in visualizing survival data. Consider a random sample of survival times $t_1, t_2, \ldots, t_n$. If no censored observations are present for the n observations, the empirical survival function

$$\hat{S}(t) = \frac{n'(t' \geq t)}{n} \quad t \geq 0 \tag{50}$$

where $n'(t' \geq t)$ denotes the number of patients that has survived until at least time t. Some modifications are needed when working with censored survival data. This is due to the fact that the number of survival times equal to or exceeding t is generally not known exactly. The required modification is known as the Kaplan-Meier (KM) estimate and is also called the product-limit (PL) estimate.

The KM estimate gives a non-parametric estimate of the survival curve of survival data containing censored survival times [87]. Suppose we have a random sample of n, survival times $t'_i, i \in \{1, 2 \ldots n\}$ and that there are $k, (k \leq n)$ distinct times at which events occur $t_1 < t_2 < \cdots < t_k$. Denote the number of deaths that occur at time t_j by d_j and the number of patients at risk at time t_j, that is the number of patients that has survived and are uncensored just prior to t_j, by n_j. The KM estimate of S(t) is then defined as

$$\hat{S}(t) = \prod_{j:t_j<t} \frac{n_j - d_j}{n_j}. \tag{51}$$

By convention, censoring times that are recorded to be equal to the time for which an event occurs are considered to happen infinitesimally later. If latest observed time is that of a censored sample, the KM estimate is only defined up until the time of this observation.

Three assumptions are made in the KM estimate: censoring is assumed to be unrelated to the prognosis, the survival probabilities are assumed to be equal for patients recruited throughout the study's inclusion period and that events occur at the recorded times [88]. The third assumption is fulfilled to a lesser and greater degree in different types of events and data. When considering conception data, fulfilment of this assumption is no problem. However, if the considered event is recurrence of some malignancy, the event will only be known to have occurred sometime between the time it is found and the time of the last inspection [89].

Even though the estimated survival curves are a great tool for visually comparing the surviving proportions of patient groups at any specific time, they do not provide a comparison of their the total survival experience [88]. For this purpose, formal methods of hypotheses testing is needed [89]. The most common hypothesis test for comparing the survival of patient groups is the logrank test [88].

The logrank test is a statistical hypothesis test that compares the null hypothesis that the risk of death in two patient groups are the same to the alternative hypothesis that it is different. Under the null hypothesis the number of deaths at a given time is expected to be distributed according to the number of patients at risk within each group at that time.

Suppose we have two patient groups A and B. Denote the total number of deaths at time t_j by d_j. The total number of deaths is distributed between daeths in patient group A, d_{jA} and deaths in patient group B, d_{jB} such that the total nunber of deaths $d_j = d_{jA} + d_{jB}$. In a similar manner we denote the number of patients at risk at time t_j from group A, B and in total by n'_{jA}, n'_{jB} and n'_j respectively. Under the null hypothesis of equal risk of death between the two patient groups d_{jA} and d_{jB} follow a hypergeometric distribution [90]. From the properties of the hypergeometric distribution we thus have that the expectation value and variance of d_{jA} are given by

$$\begin{aligned} E(d_{jA}) &= n'_{jA}\frac{d_j}{n'_j} \\ \mathrm{var}(d_{jA}) &= \frac{d_j(n'_j - d_j)n'_{jA}n'_{jB}}{{n'_j}^2(n'_j - 1)} \end{aligned} \tag{52}$$

By summation over all times of death, t_j, we obtain

$$\begin{aligned} O_A &= \sum_{t_j} d_{jA} \\ E_A &= \sum_{t_j} E(d_{jA}) \\ V_A &= \sum \text{var}(d_{jA}) \end{aligned} \tag{53}$$

where O_A is the observed number of deaths in group A and E_A and V_A are the expected number of deaths and variance in the number of deaths of group A under the null hypothesis. The logrank statistic for equivalence of death rate between two patient groups is

$$\frac{(O_A - E_A)^2}{V_A} \tag{54}$$

which can be shown to be approximately follow a χ^2 distribution with one degree of freedom [85]. The logrank test is based on the same assumptions as the KM estimate. In addition, the logrank is based on the assumption of proportional hazards, which is that the relative risk between the patient groups remains constant over time [91]. If this assumption fails, the power of the logrank test to detect differences of survival between groups may be severely reduced [92].

The most common way of checking the assumptions of proportional hazards is to fit a Cox model with one term representing the group partitioning and another term representing the interaction of the partitioned group and time [91]. In a recent study by Royston et al. [92] that reviewed 50 randomised controlled trials from four leading medical journals with time-to-event outcomes where logrank and Cox proportional hazard models were used, the proportional hazard assumptions were checked and reported in only 28% of the studies.

2.6 Proportional Hazard Models and Cox Regression

In proportional hazard (PH) models the hazard function for the lifetime, T, given a $p \times 1$ vector of fixed covariates, $\boldsymbol{x}$, is

$$h(t|\boldsymbol{x}) = h_0(t) r(\boldsymbol{x}). \tag{55}$$

Here only the form $r(\boldsymbol{x}) = \exp(\boldsymbol{\beta}^T \boldsymbol{x})$ with $\boldsymbol{\beta}$ a $p \times 1$ vector of regression coefficients will be considered. The models are called PH models due to the fact that individuals will have hazard functions that are constant multiples of another [87].

Consider a random sample (t_i, δ_i), $i \in \{1, \ldots, n\}$ with k distinct lifetimes $t_{(1)} < \ldots < t_{(k)}$, and $n-k$ censoring times. δ_i is referred to as censoring indicator or status indicator and is zero if the subject is censored and one otherwise. Let $\mathcal{R}_i = \mathcal{R}(t_{(i)})$ denote the set of patients who are alive and uncensored just prior to time $t_{(i)}$. This if referred to as the risk set. The probability of subject i enduring an event at time t given that an event occurs at time t and that $i \in \mathcal{R}(t)$ is

$$\frac{h(t|\boldsymbol{x}_i)}{\sum_{l\in\mathcal{R}(t)} h(t|\boldsymbol{x}_l)} = \frac{\exp\left(\boldsymbol{\beta}^T\boldsymbol{x}_i\right)}{\sum_{l\in\mathcal{R}(t)} \exp\left(\boldsymbol{\beta}^T\boldsymbol{x}_l\right)}. \tag{56}$$

Based on this intuition Cox [93] proposed the following likelihood estimate for the regression coefficients

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{k} \frac{\exp\left(\boldsymbol{\beta}^T\boldsymbol{x}_{(i)}\right)}{\sum_{l\in\mathcal{R}(t)} \exp\left(\boldsymbol{\beta}^T\boldsymbol{x}_l\right)}. \tag{57}$$

Solving for the regression coefficients can be performed using the maximum likelihood (ML) equation $\boldsymbol{U}(\boldsymbol{\beta}) = \mathbf{0}$ where $\boldsymbol{U}(\boldsymbol{\beta}) = \frac{\partial}{\partial\beta}\log L(\boldsymbol{\beta})$. The calculation of p-values and confidence intervals for the regression coefficients are outside the scope if this thesis. A summary of test statistics that can be used is given by Aalen et al [84].

The hazard ratio, HR, between patients indexed 1 and 2 can now be estimated as

$$\hat{HR} = \frac{\hat{h}(t|\boldsymbol{x}_1)}{\hat{h}(t|\boldsymbol{x}_2)} \tag{58}$$

where $\hat{h}$ is the estimated hazard and $\boldsymbol{x}_i$, $i \in \{1, 2\}$ are the covariate vectors for the considered patients. Since we restricted ourselves to only discussing relative risk functions of the form $\exp\left(\boldsymbol{\beta}^T\boldsymbol{x}\right)$, the estimated HR becomes

$$\hat{HR} = \exp(\boldsymbol{\beta}^T(\boldsymbol{x}_1 - \boldsymbol{x}_2)). \tag{59}$$

The interpretation of HR in terms of $\boldsymbol{\beta}$ is as follows: for a difference of one unit of element j in $\boldsymbol{x}_1$ as compared to $\boldsymbol{x}_2$ when all other elements in the covariate vectors are kept equal, the expected hazard of patient 1 is $\exp\beta_j$ times that of patient 2.

3 Materials and Methods

3.1 OxyTarget study

The patient cohort used in this thesis were participating in a wider study titled ”The OxyTarget study – Functional MRI of Hypoxia-Mediated Rectal Cancer Aggressiveness”. Patient inclusion was in the period from October 2013 and December 2017. All patients treated for rectal cancer at Akershus University Hospital were invited to participate. The study seeks to identify novel imaging biomarkers of hypoxia-induced rectal cancer aggressiveness in order to predict patients with poor response to CRT [94]. A total of 192 patients were enrolled in the study.

3.2 Images & Patients

Of the 192 patients enrolled, this thesis used data from a cohort of 61 patients having matched T2w images, DWI and ROIs where resulting images did not show significant artifacts. All patients had DWI performed for b-values 0, 25, 50, 100, 500 and $1000\,\mathrm{s\,mm^{-2}}$. Two patients where the T2w images and DWI were not made along the same axis were removed. In addition five patients were removed due to alignment issues between the T2w images and the DWI. This is explained in more detail in section 4.1.

The resulting patient cohort had a total of 54 patients. The patient cohort was partitioned according to whether a given patient were to receive preoperative CRT. This was done to keep the treatment within a treatment group as homogeneous as possible and to decrease the effect of treatment differences between patients in the survival analysis. From now on these patient groups will be referred to as *CRT* and *No CRT* and treatment groups will be used to refer to them both. The partitioning resulted in a CRT group of 24 patients and a No CRT group of 30 patients. The patient demographic for each treatment group is summarized in table 1.

MRI was performed on a Philips Achieva 1.5 T system (Philips Healthcare, Best, The Netherlands) using NOVA Dual HP gradients and a five channel cardiac coil with parallel imaging capabilities. In order to reduce bowel movements patients received glucagon ($1\,\mathrm{mg\,ml^{-1}}$, $1\,\mathrm{ml}$ intramuscularly) and Buscopan®($20\,\mathrm{mg\,ml^{-1}}$, $1\,\mathrm{ml}$ intravenously) before and during examination. MRI was performed according to clinical procedure. High-resolution T2w images perpendicular to the tumour axis was used for tumour delineation. The in-plane spatial resolution of the T2w images was 512 by 512 pixels and 128 by 128 pixels for the DWI.

3.3 Pre-Processing

A summary of the workflow adapted for this thesis is shown in figure 15 together with references to where code performing a specific function can be found in the appendix. The MRI images were converted from dicom format to a more

Treatment		
CRT	Number of patients	24
	Male	18
	Female	6
	Number censored	15
	Median age [years]	63
	Age range [years]	(41 - 78)
	Median follow-up [Days]	568
	Follow-up range [Days]	(63 - 1715)
No CRT	Number of patients	30
	Male	17
	Female	13
	Number censored	20
	Median age [years]	70
	Age range [years]	(47 - 84)
	Median follow-up [Days]	674
	Follow-up range [Days]	(93 - 1794)

Table 1: Patient demographic for each of the treatment groups used in this thesis.

convenient format in terms of ease of access in MATLAB® using code described in A.3.1. The delineated tumour volumes were stored as logical masks in NIFTY files. The built in MATLAB® function for reading NIFTY images, `niftiread`, is known to contain a bug. A fix suggested by GITHUB® user Chris Rorden [95] was adapted for use in this thesis.

The spatial resolution and FOV of DWIs and T2w images were unequal. A linear interpolation was performed to achieve DWIs with spatial resolution and spatial position of voxels equal to that of the T2w images. The same operation was performed parameter maps derived from the DWIs. Images were delineated by an expert radiologist. The radiologist's masks was used to extract the tumour voxels used as input for the clustering algorithms.

3.4 Feature Extraction and Feature Generation

The dataset was built extracting the voxel intensity of each voxel in the delineated tumour volumes for each image sequence and parameter map. The data used as input for the model was constructed such that each voxel was represented by its intensity value. It was found in an earlier conducted study that automatic delineation performance using linear discriminant analysis on the described data was better when using the raw DWI images than when using the derived ADC and IVIM images [28]. It was postulated that this might have been due to noise introduced when fitting the DWI images to the ADC and IVIM models. In this study it was thus decided to keep all DWI images in the

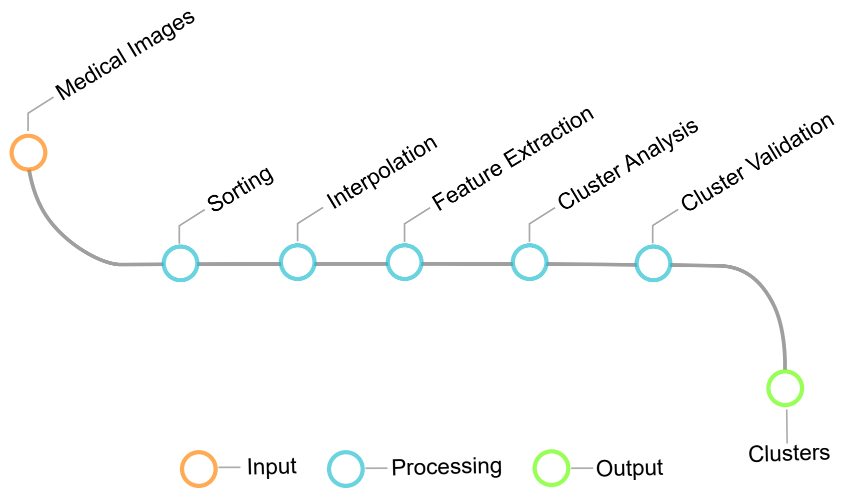


Figure 15: The workflow used in this thesis for the conversion of raw medical images to the obtained clusters. The references in parenthesis are the locations of the corresponding code in the appedix. Raw medical images were first sorted and interpolated to have the same spatial resolution and position of corresponding voxels (A.3.1). Voxel intensities were extracted using the radiologist's mask (A.3.4). Feature by feature z-scoring was performed followed by PCA on the z-scored dataset to reduce the dimensionality (A.4.3). Cluster analysis was then performed with cluster validation to find the optimal number of clusters/components(Clustering: k-means A.4.4, GMM A.4.5). The best performing model was extracted from the cluster validation procedure. This model was used to partition the tumour volume in the obtained clusters used in the survival analysis (k-means: A.4.6, GMM A.4.7).

dataset.

An additional dataset was built where polynomial features were included to attempt to catch some of the evolution of voxel intensity of the tissues with increasing b-value in the DWI images. The polynomial features were built up to the third order using b-values of 25, 50, 100 and $500\,\mathrm{s\,mm^{-2}}$ keeping only the interaction terms. However, this dataset did not yield clusters that were more easily distinguishable and it was thus dropped from further analysis.

The scale of the feature vectors, that is the voxel intensities of each image sequence and parameter map, was found to vary by several orders of magnitude. The k-means algorithm and PCA are both dependent on the scale at which the different feature vectors are measured [96, 40]. The feature vectors were thus rescaled to a mean of zero and standard deviation of one. PCA was performed to reduce the number of features while retaining the maximum amount of variance and the code can be found in section A.4.3 of the appendix. The reduced feature space makes visualization of the dataset easier and more meaningful.

3.5 Visualizing the dataset

Before performing the clustering analysis the dataset was visualized. This was done by the use of histograms, scatterplots and KDE plots of the principal components and the raw data. Visualization was performed to probe for any obvious clusters in the data. Code was written to create a custom plot matrix/grid plot with scatter plots of principal component i and j for $i < j$ and kde plots below the diagonal using principal components i and j for $i > j$. Along the diagonal, the histograms of each principal component was plotted. The function for creating the plot, `kde_scatter_histplot`, can be found in the appendix A.4.1.

3.6 Clustering

Clustering was performed on the reduced feature space by the use of k-means and GMM clustering algorithms. Several CVIs were considered for finding the optimal k. For k-means it was decided to use CH, the DB family of indices and Sil. In section 2.4.2 it was argued that Sil and the DB family of indices was the best performing of these according to literature. The DB and CH has the additional advantage that its time complexity is linear in the number of samples in the dataset, while the Sil is quadratic. With the CH and DB family of indices it is thus possible to use more samples in the clustering evaluation than what is feasible for the silhouette coefficient.

The clustering was performed in the scikit-Learn package (v0.20.3) [97] in python. The CVIs DB* and DB** is not in the base implementation of MATLAB® or the scikit-learn package. A list of python packages and versions used in this thesis can be found in section A.2 fo the appendix. An attempt was made to find maintained packages containing either of the indices in either of the programming languages using the MATLAB® File Exchange [98] and the Python Package Index (PyPI) [99], but no implementations were found. An implementation was thus coded by modifying the DB index of the scikit-learn package as discussed in section 2.4.2. The new implementation of the DB* and DB** indices can be found in the appendix A.4.2 together with a quick performance comparison of the DB, DB* and DB** on datasets with varying numbers of bivariate isotropic Gaussian clusters A.5. The performance comparison showed that DB* on average outperformed DB, as shown by Kim and Ramakrishna [71] and Arbelaitz et al. [65], and that DB** on average outperformed DB and DB* as shown by Kim and Ramakrishna [71].

For each k tested the k-means clustering was run 30 times with different initial centroid locations. This is done because k-means, if it converges, is only guaranteed to converge to a local minimum. A way to overcome this is to run k-means multiple times for each k with different initial partitions and choose the clustering solution with the smallest objective function (29) [15]. The maximal number of iterations was set to 600. For each chosen clustering solution it was checked that convergence was obtained.

Initial centroids were chosen using the k-means++ algorithm [100]. In this algorithm the cluster initial centroids are chosen at random from the data, but are weighted according to the square of the distance from the closest centroid already chosen. In the original article [100], the k-means++ algorithm was shown to consistently outperform the classic k-means algorithm in synthetic and real datasets.

The GMM clustering was performed setting no restriction on the shape of the covariance matrix. k-means was used to set the initial mixture proportions, covariance matrices and mean vectors. The GMM clustering was run with $3 \cdot 10^5$ randomly sampled voxels from the dataset and for a number of components $n_c \in \{1, \dots, 70\}$. For each n_c the GMM was initialized three times. The best initialization was selected. This process was repeated five times using different datasets. To select the optimal number of components the BIC was calculated.

The 5 repeats using differently sampled datasets were also used to assess the reproducibility and validity of the GMM clustering solutions. Each repeat will assign a random identifier or label to each GMM component. To assess which components were most equal between two clustering solutions the Jensen-Shannon (JS) distance was used. The JS divergence, proposed by J. Lin [101], can be considered a symmetric and smoother form of the Kullback-Liebler (KL) divergence [102, 103]. KL, and thus JS, measures the distance between probability distributions [104, 105] and it has been used extensively within the field of bioinformatics [102]. KL divergence, also called the relative entropy [105], between two probability distributions $\boldsymbol{P}$ and $\boldsymbol{Q}$ is defined as

$$D_{\mathrm{KL}}(\boldsymbol{P}||\boldsymbol{Q}) = \sum_{i=1}^{N} P_i \log \frac{P_i}{Q_i} \tag{60}$$

it is non-negative, non-symmetric, that is $D_{KL}(\boldsymbol{P}||\boldsymbol{Q}) \neq D_{KL}(\boldsymbol{Q}||\boldsymbol{P})$ and is zero if and only if $\boldsymbol{P} = \boldsymbol{Q}$. The JS distance between probability distributions

$\boldsymbol{P}$ and $\boldsymbol{Q}$ is defined as the square root of the JS divergence

$$D_{\mathrm{JS}}(\boldsymbol{P}||\boldsymbol{Q}) = \sqrt{\frac{D_{\mathrm{KL}}(\boldsymbol{P}||\boldsymbol{M}) + D_{\mathrm{KL}}(\boldsymbol{Q}||\boldsymbol{M})}{2}} \tag{61}$$

where $\boldsymbol{M} = \frac{1}{2}(\boldsymbol{P} + \boldsymbol{Q})$. The JS distance is non-negative symmetric, always well defined and bounded, is null only when the probability distribution coincide and it satisfies the triangle inequality [106]. It is thus a true metric [106, 107] between probability distributions [106].

The best solution of the five repeats was extracted using the minimum BIC score as a criterion and used as a reference distribution. The JS distance between each component in the reference distribution and each component of the repeat clustering solutions was calculated. Each component label of the repeat cluster solution was mapped to the closest component in the reference distribution in terms of the JS distance.

To measure the agreement between the reference clustering solution and repeat clustering solutions Cohen's κ, Jaccard similarity score and balanced accuracy score was used. These are defined as:

Cohen's Kappa, κ, [108] measures agreement between two raters in classifying N samples into C distinct and exclusive classes. It is defined as

$$\kappa = \frac{p_0 - p_e}{1 - p_e} \tag{62}$$

where p_0 is the relative interrater agreement and p_e is the expected agreement if both raters were to assign labels randomly. $\kappa = 1$ in case the case of perfect interrater agreement, zero if agreement is no better than what would be expected by chance, and negative if interater agreement is worse than what is expected by chance.

Jaccard Index, $\mathcal{J}$, [109] of sets A and B is defined as the intersection of the two sets divided by the union of the two sets. That is

$$\mathcal{J}(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{63}$$

Sets A and B are the labels assigned by the two raters respectively. $\mathcal{J}$ is one for perfect agreement between raters and is zero for perfect disagreement between raters.

Balance Accuracy Score (BAS) [110] is defined as the average accuracy obtained across all labels. It reduces to the conventional accuracy if the raters have equal performance across all labels.

Code written for performing the k-means clustering and GMM clustering can be found in section A.4.4 and A.4.5 of the appendix respectively. Code used to inspect the correspondence between components obtained by GMM clustering with differently sampled datasets can be found in section A.4.7 of the appendix.

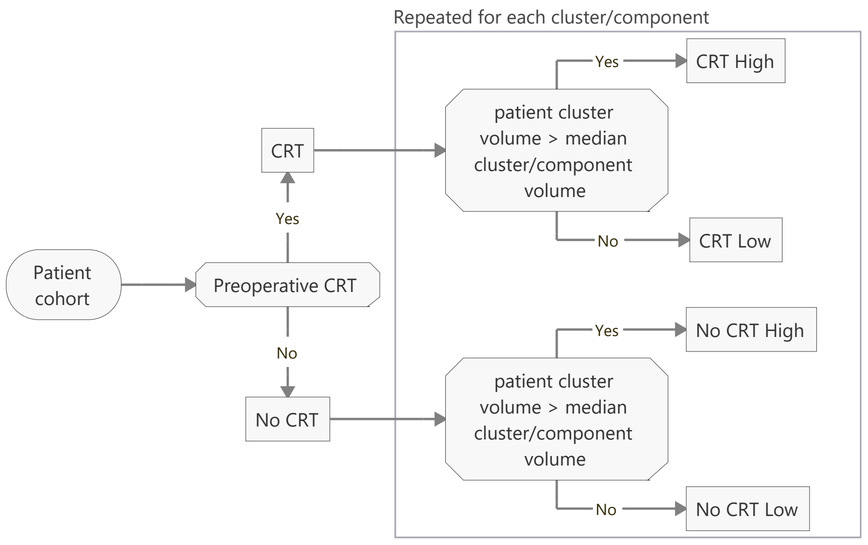

Repeated for each cluster/component
Patient cohort
Preoperative CRT
Yes
No
CRT
No CRT
patient cluster volume > median cluster/component volume
Yes
No
CRT High
CRT Low
patient cluster volume > median cluster/component volume
Yes
No
No CRT High
No CRT Low

Figure 16: The patient cohort was partitioned according to treatment. Kaplan-Meier estimates with logrank tests was used to assess survival differences within each treatment group. Patient within each treatment group was assigned to low or high cluster/component volumes using the median cluster volume within the group as a threshold. This resulted in partitions CRT High and Low

3.7 Survival Analysis

To assess whether a particular cluster could help predict patient outcome the patients were divided into whether they had above or below the median volume of a given cluster. This will be referred to as low or high cluster volume in the *k*-means partitioned volumes and low and high component volumes in GMM partitioned volumes. The partitions of the patient cohort for each cluster/component will be referred to as CRT High, CRT Low, No CRT High and No CRT Low. A diagram of the partitioning scheme is shown in figure 16. The survival analysis was performed using the Kaplan-Meier estimator with logrank test to compare the survival of patients with low and high cluster and component volumes for each component/cluster and treatment group. Progression free survival (PFS) was used as the primary endpoint of the study. Time-to-event was thus considered time to any progression of disease (e.g. local relapse, metastasis) or death to occur. The survival analysis was performed in the lifelines survival package in python [111].

Univariate Cox regression was also performed. This was done using the cluster and component volumes for both treatment groups as covariates. In both logrank tests and Cox regression a significance of $\alpha < 0.05$ was chosen. The significance level was corrected for multiple testing using the Bonferroni adjustment according to the number of partitioned clusters. The Bonferroni adjustments corrects the significance threshold by dividing by the number of separate tests performed [112]. In assessing calculated confidence intervals the notion of compatibility was used as suggested by Amrheim et al [113]. It is suggested to rename confidence intervals to compatibility intervals to highlight the fact that all values within the interval limits are reasonably compatible with the data.

The code used in the survival analysis of the k-means partitioned dataset can be found in section A.4.6 of the appendix. The corresponding code for the GMM partitioned dataset can be found in section A.4.7 of the appendix.

3.8 Software & Code

A more in depth description with regards to the software and packages used in this thesis is given in section A.2 in the appendix. Here, a short description of what each package was used for is given in addition to which version of the package was used. All code used in this thesis can be found in the appendix.

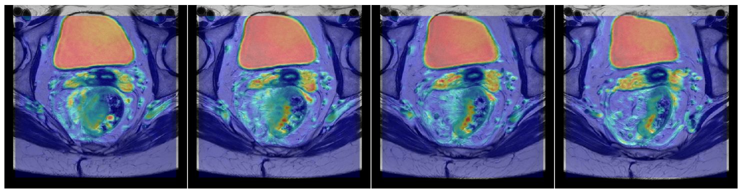


Figure 17: DWI images with b-value of $0\,\mathrm{s\,mm^{-2}}$ superimposed as heatmaps on T2w images. Images are aligned correctly.

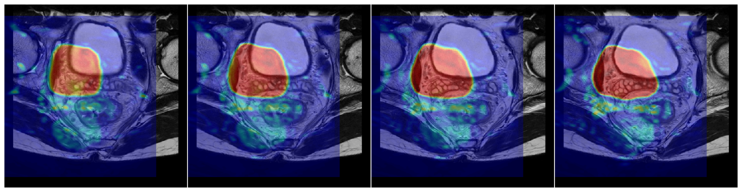


Figure 18: DWI images with b-value of $0\,\mathrm{s\,mm^{-2}}$ superimposed as heatmaps on T2w images. Images are misaligned.

4 Results

4.1 Verification of alignment & Resulting dataset

In interpolating the DWI and the derived parameter maps to attain voxelwise alignment with the T2w images and masks some discrepancies were observed. This caused a subsequent more thorough inspection of the images. Code was written to superimpose the DWI images as heatmaps on the T2w images. The location of the bladder was used as a reference where available due to being bright with clearly defined edges and easily recognizable in the images. An example of correct alignment of DWI and T2w images is presented in figure 17. An examples of an incorrectly aligned image is presented in figure 18. A different alignment issue is presented in figure 19, where the T2w image and DWI do not cover the same FOV.

Alignment issues were observed for a total of five patients. These patients were removed from the study. The resulting dataset contained a total of 54 patients, amoung whom 24 were in the CRT group and 30 in the No CRT group.

4.2 Visualization and pre-processing of the dataset

Prior to clustering analysis the dataset was visulized using various tools. A histogram of the features, that is the voxel intensities of each image sequence, is shown in figure 25. The histogram is made using all the samples avalible in the dataset. From the histogram it is apparent that there are no trivially separable clusters due to a single feature alone. PCA was performed on the dataset and it

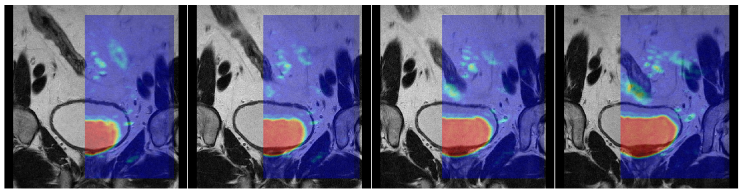


Figure 19: DWI images with b-value of $0\,\mathrm{s\,mm^{-2}}$ superimposed as heatmaps on T2w images. Images are misaligned and do not cover the same FOV.

was observed that a large fraction of the total variance could be explained with relatively few principal components. A pareto chart of the explained variance with each added component can be found in section A.4.3 of the appendix. It was chosen to keep the first 4 principal components that together explained 94.30% of the total variance. The original features were z-scored before the PCA was performed.

The principal components were visualized using the custom plots explained in section 3.5. The resulting plot is found in figure 26. The figure is made using 5000 samples, and the optimal bandwidths are found through a cross-validation scheme. No easily distinguishable clusters are visible by using the KDE plots. An a priori estimate with regards to the number of clusters in the data is thus hard to assess.

From the visual inspection of the dataset it is evident that even though no obvious clustering can be observed, the dataset does not seem to be sampled from a single multivariate normal distribution. Clustering may thus still be a viable aid to help find partition the dataset into tissues with similar characteristics.

4.3 k-means clustering

k-means clustering was performed on the two datasets with $k \in \{1, \ldots 21\}$ and by using all avaliable samples. Convergence was obtained for all k. To find the optimal k, CH, DB, DB*, DB** and SIl were computed. All but one of the computed CVIs indicated and optimal k of two. The exception was DB that indicated the optimal k to be four. Plots of the CVI scores against k can be found in section A.6 of the appendix. It should be noted that none of the tested indices are comparing the clustering solution to the null hypothesis that there are no clustering present at all. As discussed previously and tested in section A.5 of the appendix, DB*, DB** and Sil on average performs better than DB, and thus $k = 2$ is considered the most valid result and will be considered in the following analysis.

Partitioning the dataset into 2 clusters by use of k-means resulted in one cluster containing 63.4% of the voxels in the dataset and the other cluster containing 36.6% of the voxels. The two groups are depicted as cluster by cluster histogram in figure 27. From the histogram it is evident that the separation

	Volume Cluster 1 [mm^3]	Volume Cluster 2 [mm^3]
mean	9958.94	16487.34
std	9391.46	13373.21
median	7305.05	13373.11
range	(46.20-36357.98)	(2425.58-47977.88)

Table 2: Descriptive statistics of cluster volume per patient using the k-means algorithm with $k = 2$.

between the clusters is best explained by the DWI of b-values 0, 25, 50 and $100\,\mathrm{s\,mm^{-2}}$ out of the original features. Each cluster seems to contain primarily high or low values due to either of the mentioned features. The tendency is also visible in DWI for b-value $500\,\mathrm{s\,mm^{-2}}$ and D where mean value of each cluster for each feature are clearly different. However, the overlap between the feature histograms for the clusters are considerably larger in this case. Descriptive statistics on the distribution of the cluster volume per patient in given in table 2.

4.4 Survival Analysis using k-means partitioned volumes

The patient cohort was partitioned according to the scheme described in the materials and methods section. That is for each treatment, CRT or No CRT, the patients were partitioned according to the median volume of each component into groups CRT High, CRT Low, No CRT High and No CRT Low. With $k = 2$ the Bonferroni corrected significance level was $\alpha_{\mathrm{BF}} < 0.0025$.

No significant difference between low and high cluster volumes were found in the CRT group. The KM estimates for the clusters 1 and 2 are shown in figures 20 and 21 respectively. The p-values reported in the figures are not corrected for multiple testing. The PH assumptions was checked to hold for the logrank tests.

In the No CRT group a significant difference in PFS was found between low and high cluster volumes for cluster 1 ($p = 0.018$). No significant difference was found for cluster 2 ($p = 0.093$). PH assumptions were found to hold for cluster 1, but not for cluster 2. KM estimates for clusters 1 and 2 are shown in figure 22 and 23 respectively.

The KM estimate and logrank test was also performed for the total tumour volume partitioned according to the median and divided into treatment groups in the same manner as cluster volumes. No significant association to PFS was found for the CRT ($p > 0.7$) or the No CRT ($p > 0.1$) treatment groups. The KM estimate for the total volume in CRT and No CRT groups respectively can be found in figures 36a and 36b in section A.7 of the appendix.

The results from univarite Cox regression are summarized in table 3. Volume of cluster component 1 was found to be significantly associated with PFS ($p < 0.006$). The results show that each increase of $1\,\mathrm{cm}^3$ in cluster volume 1 between patients in the No CRT group is compatible with an increase in the expected

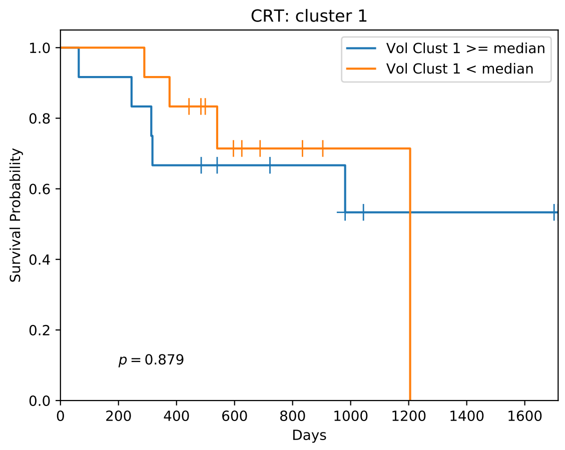


Figure 20: Kaplan-Meier (KM) estimate for the patient group treated with CRT partitioned according to median volume of cluster 1. Censored patients are indicated with crosses on the respective survival curves. The KM estimate fails to show any association between low or high volume of cluster 1 and progression free survival. The logrank test for difference in hazard between the two patient groups resulted in a p-value of 0.879.

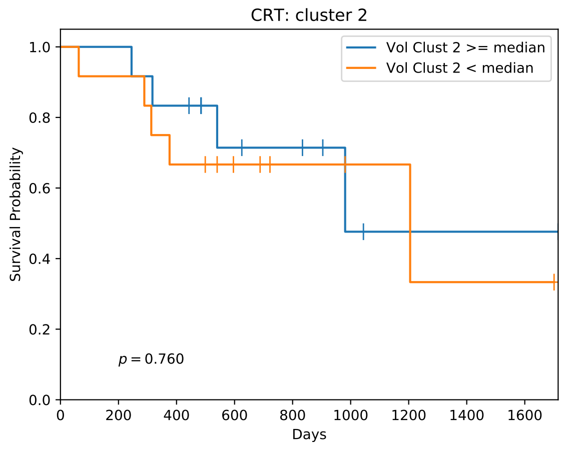


Figure 21: Kaplan-Meier (KM) estimate for the patient group treated with CRT partitioned according to median volume of cluster 2. Censored patients are indicated with crosses on the respective survival curves. The KM estimate fails to show any association between low or high volume of cluster 2 and progression free survival. The logrank test for difference in hazard between the two patient groups resulted in a p-value of 0.760.

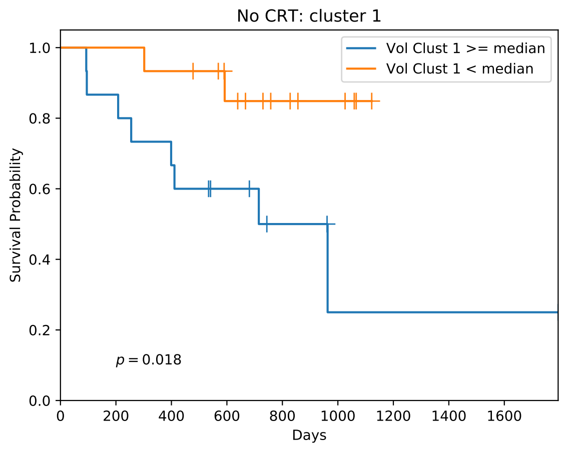


Figure 22: Kaplan-Meier (KM) estimate for the patient groups not treated with CRT partitioned according to median volume of cluster 1. Censored patients are indicated with crosses on the respective survival curves. The KM estimate shows that the group with lower volume of cluster 1 was associated with a greater chance of progression free survival. The logrank test for difference in hazard between the two patient groups resulted in a p-value of 0.018.

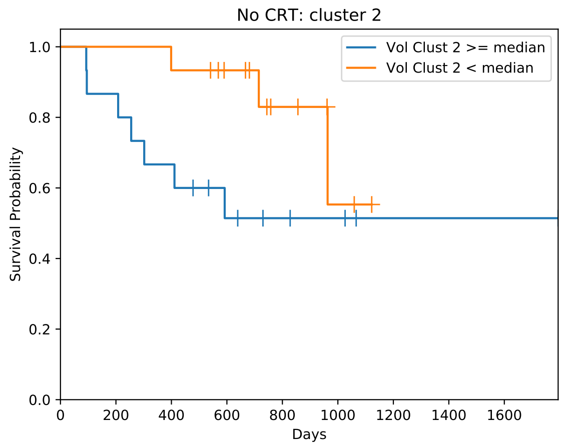


Figure 23: Kaplan-Meier (KM) estimate for the patient groups not treated with CRT partitioned according to median volume of cluster 1. Censored patients are indicated with crosses on the respective survival curves. The KM estimate with logrank test failed to show a significant difference in PFS between patients with low and high volume of cluster 2 yielding a p-value of 0.093.

Treatment	Risk Factor	β	e^{β}	p	95% CI (β)
CRT	VolClust1 [cm^3]	0.00375	1	0.922	(-0.0709 - 0.0784)
	VolClust2 [cm^3]	-0.047	0.954	0.213	(-0.121 - 0.0269)
	TotVol [cm^3]	-0.0208	0.979	0.364	(-0.0656 - 0.0241)
No CRT	VolClust1 [cm^3]	0.105	1.11	0.00573	(0.0304 - 0.179)
	VolClust2 [cm^3]	0.028	1.03	0.127	(-0.00791 - 0.0638)
	TotVol [cm^3]	0.0317	1.03	0.0166	(0.00575 - 0.0576)

Table 3: Results from univariate Cox regression using the cluster component volumes resulting from a k-means clustering analysis with $k = 2$ as risk factors. The regression was performed for each treatment group independently. The Volume of cluster 1 and the total tumour volume in the No CRT group showed a significant association to PFS. *VolClust* and *totVol* are used as abbrevations for volume of a cluster component and the total tumour volume respectively.

hazard of between 3.1% and 20%. The total tumour volume in the No CRT group also showed a significant association with PFS ($p < 0.02$). A relative increase in tumor volume of $1\,\mathrm{cm}^3$ between patient was compatible with an increase in the expected hazard of between 0.58% and 5.9%. No other cluster volumes in either treatment group showed a significant association to PFS. PH assumptions was checked to hold for each risk factor.

4.5 Gaussian mixture model clustering

Preliminary GMM clustering results on $2 \cdot 10^4$ samples from the dataset showed convergence of the BIC to a global minimum before exceeding the range of tested components. However, this convergence disappeared upon including additional samples. Performing the clustering as described in section 3.6, a local minimum in the BIC was observed for nine components. To avoid overfitting the data, and to keep the results fairly interpretable, nine components was thus selected. The BIC score and gradient BIC score are shown in figure 24. By examining the the gradient of the BIC, that is the change in the BIC from one n_c to the next, rather small changes are seen when adding additional components exceeding the ninth indicating that not much additional information is gained.

The voxels partitioned with the GMM into nine components was mapped back onto the axes of the original, z-scored features. The result is presented as a histogram in 28. Histograms of each component against all other components grouped together can be found in section A.8 of the appendix. Here it easier to distinguish the traits of each component. Descriptive statistics depicting the variation of the cluster component volumes per patient can be found in table 4. It is evident that the range of component volumes and total volume spans multiple orders of magnitude. From the table one can also see that all component constitute a considerable fraction of the tumour volume in at least

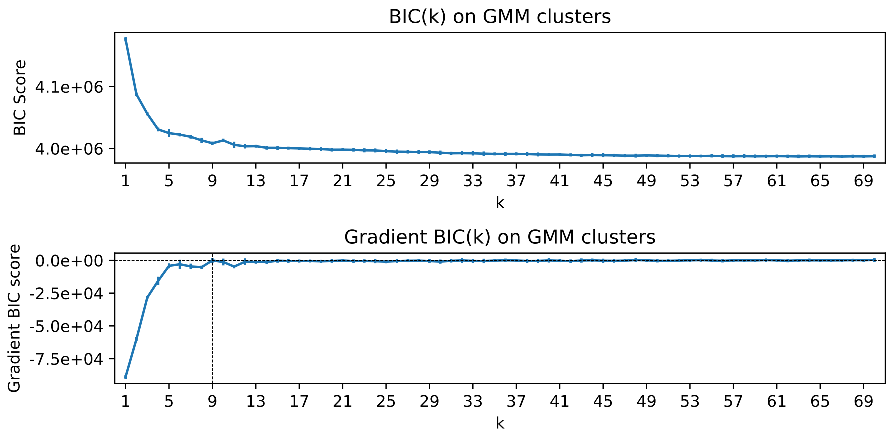


Figure 24: BIC score and gradient BIC score calculated for number of components $n_c \in \{1, 2 \ldots 70\}$. The GMM is fitted using 300000 random samples. On these samples the GMM is initialized 3 times and the best model is selected. This process is repeated 5 times for each n_c with different randomly sampled datasets each time. The line is the mean of the 5 repeats fits for each n_c for different random samples and the errorbars are the standard deviation.

one patient. This reaffirms the suspicion that none of the components consist of purely outliers in the data. Patients without tumour volume due the components 3, 6 and 7 are found in the CRT group, and patients without tumour volume due to components 6 and 7 in the NO CRT group.

The GMM solutions was found to be consistent across repeats. In the following the best performing model out of the five repeats was chosen as a reference model. The four remaining repeat models are named Model 1 through 4. A summary of calculated performance indices comparing the clustering solutions is shown in table 5. All repeat models seem to give roughly the same clustering solutions. This is indicated by the high mean and low standard deviation in the index values across the models. The obtained correspondence between the clustering solutions point to the validity of the given clustering solution and settings. Especially considering the fact that each model is fit using a different set of input data, and due to the inherit randomness in the initialization of the GMM components between repeats.

4.6 Survival analysis on the Gaussian Mixture Model Components

Kaplan-Meier estimates and logranks tests were performed on the component volumes partitioned according to the median component volume for each treatment group. The results are found in table 6. No significant difference in PFS was seen due to any of the components in the CRT group. In the No CRT

treatment	Volume per patient	mean [mm^3]	std [mm^3]	median [mm^3]	range [mm^3]
CRT	Component 1	$3.06 \cdot 10^3$	$7.66 \cdot 10^3$	491	$(46.3 - 2.8 \cdot 10^4)$
	Component 2	$1.91 \cdot 10^3$	$2.87 \cdot 10^3$	$1.08 \cdot 10^3$	$(28.7 - 1.44 \cdot 10^4)$
	Component 3	861	$1.37 \cdot 10^3$	227	$(0 - 5.57 \cdot 10^3)$
	Component 4	$6.28 \cdot 10^3$	$8.75 \cdot 10^3$	$3.96 \cdot 10^3$	$(459 - 4.41 \cdot 10^4)$
	Component 5	$6.57 \cdot 10^3$	$8.12 \cdot 10^3$	$4.78 \cdot 10^3$	$(44.4 - 4.04 \cdot 10^4)$
	Component 6	$2.85 \cdot 10^3$	$2.74 \cdot 10^3$	$2.03 \cdot 10^3$	$(0 - 1.16 \cdot 10^4)$
	Component 7	809	894	677	$(0 - 3.87 \cdot 10^3)$
	Component 8	$3.3 \cdot 10^3$	$5.76 \cdot 10^3$	$1.71 \cdot 10^3$	$(54.7 - 2.82 \cdot 10^4)$
	Component 9	815	$1.1 \cdot 10^3$	369	$(62 - 3.81 \cdot 10^3)$
	All Components	$2.64 \cdot 10^4$	$1.89 \cdot 10^4$	$2.01 \cdot 10^4$	$(5.38 \cdot 10^3 - 8.43 \cdot 10^4)$
No CRT	Component 1	$1.03 \cdot 10^3$	$1.62 \cdot 10^3$	396	$(1.24 - 7.76 \cdot 10^3)$
	Component 2	$1.36 \cdot 10^3$	$1.17 \cdot 10^3$	955	$(35.2 - 4.95 \cdot 10^3)$
	Component 3	979	$2.71 \cdot 10^3$	212	$(0 - 1.47 \cdot 10^4)$
	Component 4	$4.73 \cdot 10^3$	$6.33 \cdot 10^3$	$2.81 \cdot 10^3$	$(379 - 3.11 \cdot 10^4)$
	Component 5	$5.15 \cdot 10^3$	$7.5 \cdot 10^3$	$3.09 \cdot 10^3$	$(151 - 3.95 \cdot 10^4)$
	Component 6	$2.18 \cdot 10^3$	$1.64 \cdot 10^3$	$1.6 \cdot 10^3$	$(19.8 - 5.27 \cdot 10^3)$
	Component 7	847	$2.34 \cdot 10^3$	253	$(0 - 1.25 \cdot 10^4)$
	Component 8	$3.07 \cdot 10^3$	$5.14 \cdot 10^3$	$1.28 \cdot 10^3$	$(1.85 - 2.4 \cdot 10^4)$
	Component 9	585	759	235	$(2.47 - 2.73 \cdot 10^3)$
	All Components	$1.99 \cdot 10^4$	$1.81 \cdot 10^4$	$1.49 \cdot 10^4$	$(1.73 \cdot 10^3 - 9.25 \cdot 10^4)$

Table 4: Descriptive statistics on how the volume of each gaussian mixture model component and the total tumour volume fluctuates from patient to patient within each treatment group. It is evident that the volume for a given components can differ by several orders of magnitude between patients. Std is used as an abbreviation for standard deviation.

	Model 1	Model 2	Model 3	Model 4	μ	σ
κ	0.964	0.949	0.95	0.934	0.949	0.00857
$\mathcal{J}$	0.97	0.957	0.958	0.945	0.958	0.00713
BAS	0.959	0.937	0.946	0.935	0.944	0.0113

Table 5: Measures of repeatability and validity of the GMM clustering solutions. The GMM clustering is repeated 5 times with different randomly samples from the dataset. The best model in terms of BIC was selected as a reference model and components labels of repeat models were mapped to the closest component in terms of JS distance to the reference model components. Cohen's κ, Jaccard similairty index, $\mathcal{J}$, and the balanced accuracy score (BAS) are used to measure the agreement in label assignment between the reference model and repeat models (Model 1, Model 2, Model 3 and Model 4). The average, μ, and standard deviation, σ, across the models are computed. The performance measures show that for the given settings roughly the same clustering components are chosen in each repeat.

Treatment	Volume per patient	p
CRT	Component 1	0.215
	Component 2	0.211
	Component 3	0.314
	Component 4	0.760
	Component 5	0.431
	Component 6	0.988
	Component 7	0.731
	Component 8	0.203
	Component 9	0.714
No CRT	Component 1	0.034
	Component 2	0.510
	Component 3	0.116
	Component 4	0.322
	Component 5	0.159
	Component 6	0.001*
	Component 7	0.002*
	Component 8	0.828
	Component 9	0.136

Table 6: Result from Kaplan-Meier analysis with logrank test on the patients within each treatment group partitioned into low and high volume of each Gaussian mixture model component according to the median value. * denotes statistically significant associations at the Bonferroni corrected significance level of $\alpha_{\mathrm{BF}} = 0.0056$. A significant difference in PFS is seen for components 6 and 7 for the No CRT group.

group a significant difference in PFS was found for components 6 ($p = 0.001$) and 7 ($p = 0.002$) at the Bonferroni corrected significance level of $\alpha_{\mathrm{BF}} = 0.0056$. The Kaplan-Meier estimates for each component are shown in section A.9 of the appendix.

The results of univariate Cox regression performed on the gaussian mixture components are presented in table 7. The volume of component 1 in the treatment group not receiving CRT is significantly related to PFS ($p = 0.00133$) at significance level α_{BF}. The results for this component are compatible with an expected relative increase of hazard of between 17% and 103% with each increase of $1\,\mathrm{cm}^3$ in the patient's tumour volume due to this component. PH assumption was tested and found to hold for all tested risk factors.

Treatment	Risk Factor	β	e^{β}	p	95% CI (β)
CRT	VolumeComp1 [cm^3]	0.0264	1.03	0.525	(-0.055 - 0.108)
	VolumeComp2 [cm^3]	-0.148	0.862	0.509	(-0.589 - 0.292)
	VolumeComp3 [cm^3]	0.426	1.53	0.0437	(0.012 - 0.84)
	VolumeComp4 [cm^3]	-0.201	0.818	0.19	(-0.503 - 0.0997)
	VolumeComp5 [cm^3]	-0.0358	0.965	0.464	(-0.131 - 0.0599)
	VolumeComp6 [cm^3]	-0.0783	0.925	0.583	(-0.358 - 0.201)
	VolumeComp7 [cm^3]	-0.389	0.678	0.472	(-1.45 - 0.67)
	VolumeComp8 [cm^3]	-0.0749	0.928	0.554	(-0.323 - 0.173)
	VolumeComp9 [cm^3]	-0.137	0.872	0.728	(-0.91 - 0.636)
No CRT	VolumeComp1 [cm^3]	0.441	1.55	0.00133*	(0.172 - 0.71)
	VolumeComp2 [cm^3]	0.405	1.5	0.103	(-0.0823 - 0.893)
	VolumeComp3 [cm^3]	-0.124	0.883	0.637	(-0.64 - 0.391)
	VolumeComp4 [cm^3]	0.0712	1.07	0.0259	(0.00857 - 0.134)
	VolumeComp5 [cm^3]	0.0695	1.07	0.016	(0.0129 - 0.126)
	VolumeComp6 [cm^3]	0.417	1.52	0.0244	(0.0539 - 0.781)
	VolumeComp7 [cm^3]	0.174	1.19	0.0273	(0.0195 - 0.329)
	VolumeComp8 [cm^3]	-0.0748	0.928	0.416	(-0.255 - 0.105)
	VolumeComp9 [cm^3]	0.0785	1.08	0.829	(-0.634 - 0.791)

Table 7: Univariate Cox regression analysis for the components volumes of the Gaussian mixture model clustering. The Bonferroni corrected significance level due to multiple testing within each patient group is 0.0056. At this significance level we can see that only the volume of cluster component 1 for the patient group that did not recieve CRT is significantly related to PFS. Significant p-values are indicated with a *.

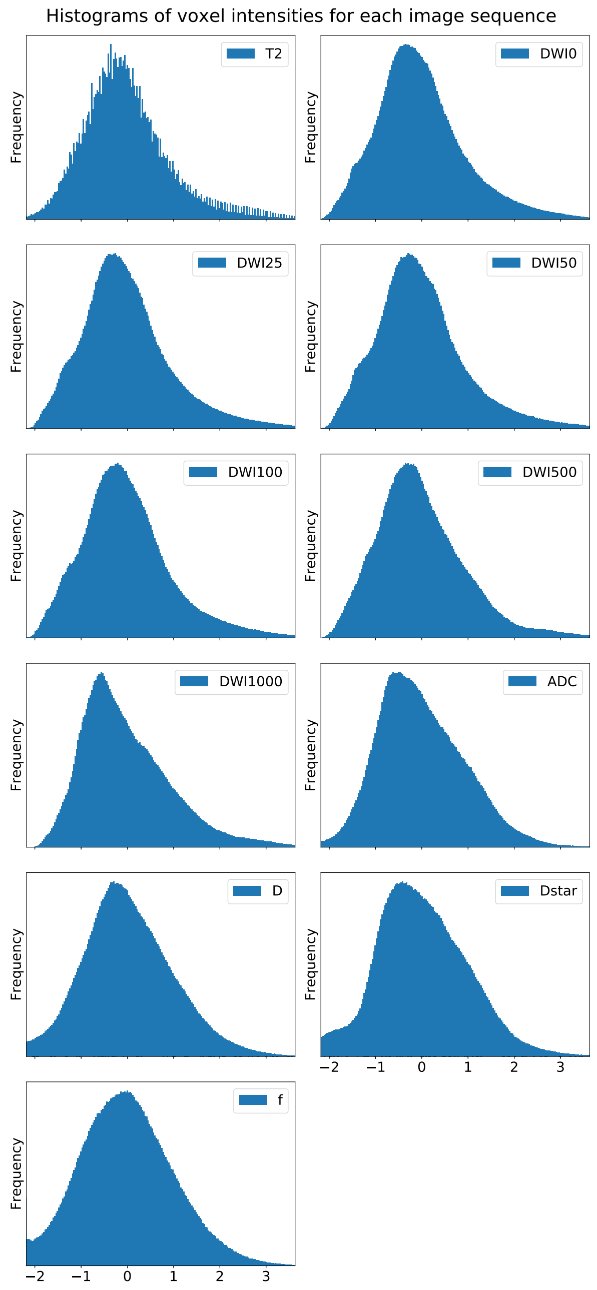


Figure 25: Histogram of the z-scored features. The upper limits on the x axis is to the 99.95th percentile to reduce the effect of outliers in visualizing the data.

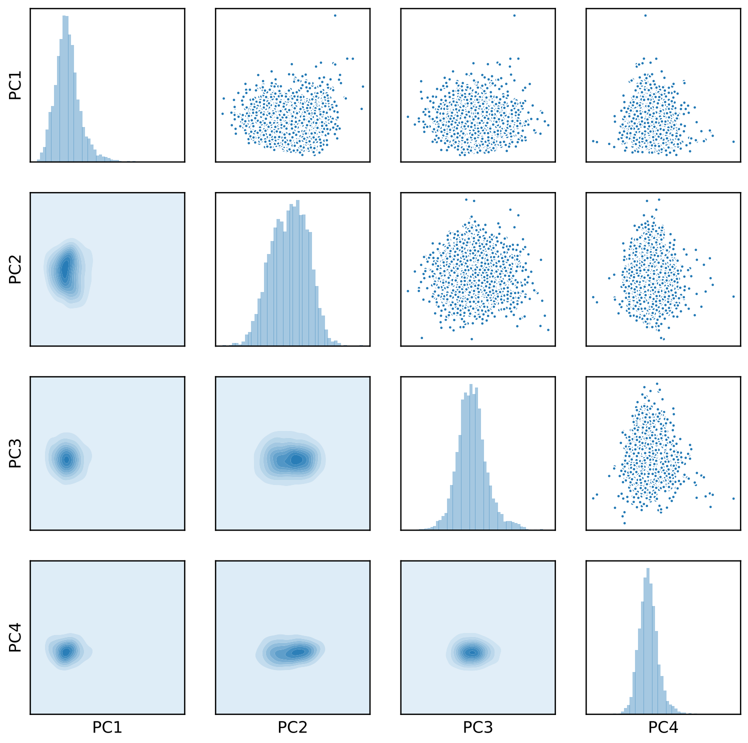


Figure 26: Custom plot matrix with scatter plots of principal components pairs above the diagonal, histograms of principal components on the diagonal and KDE plots of principal component pairs under the diagonal. The plotted data consists of the 4 first principal components of the dataset.

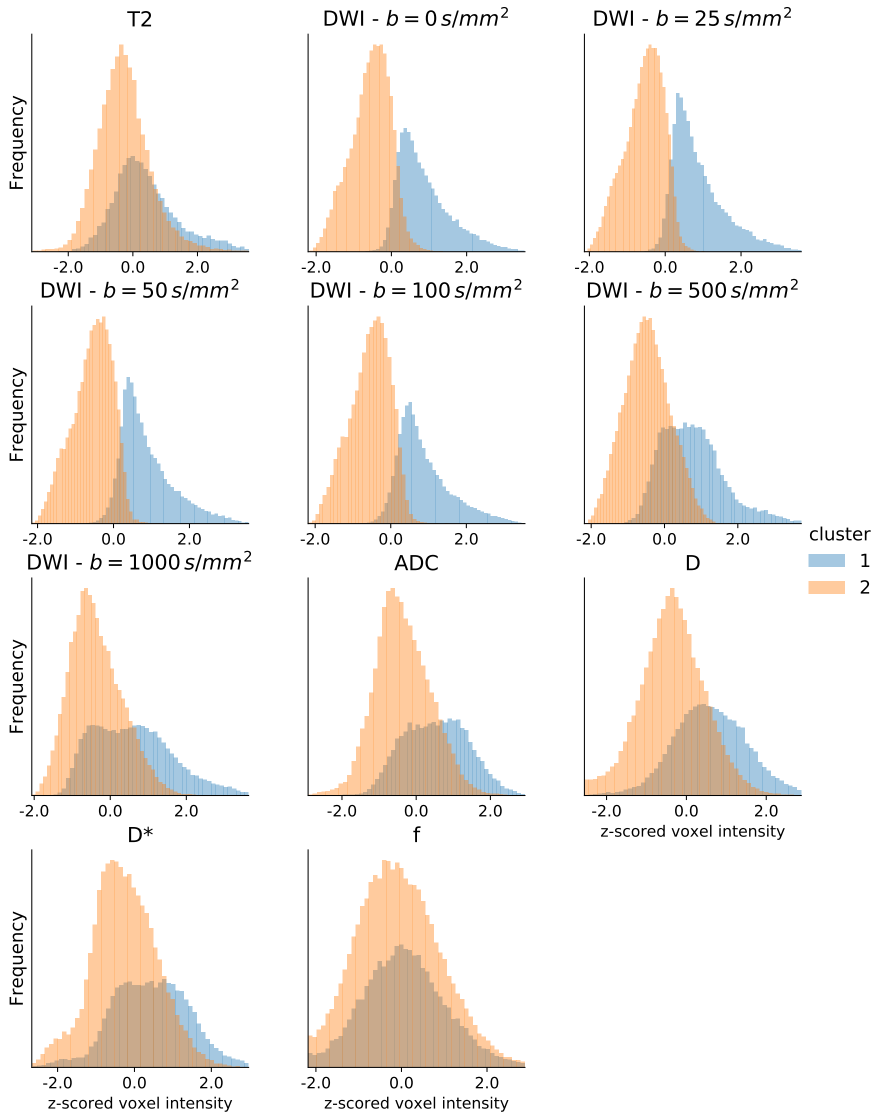


Figure 27: Histogram of the z-scored features in the dataset divided by cluster. The histogram is made by with 100000 randomly drawn samples from the dataset. Samples exceeding the 99.5th percentile in any feature were removed from the dataset before plotting to reduce the effect of outliers in selecting the number and positioning of bin edges, and the limits along the voxel intensity axis.

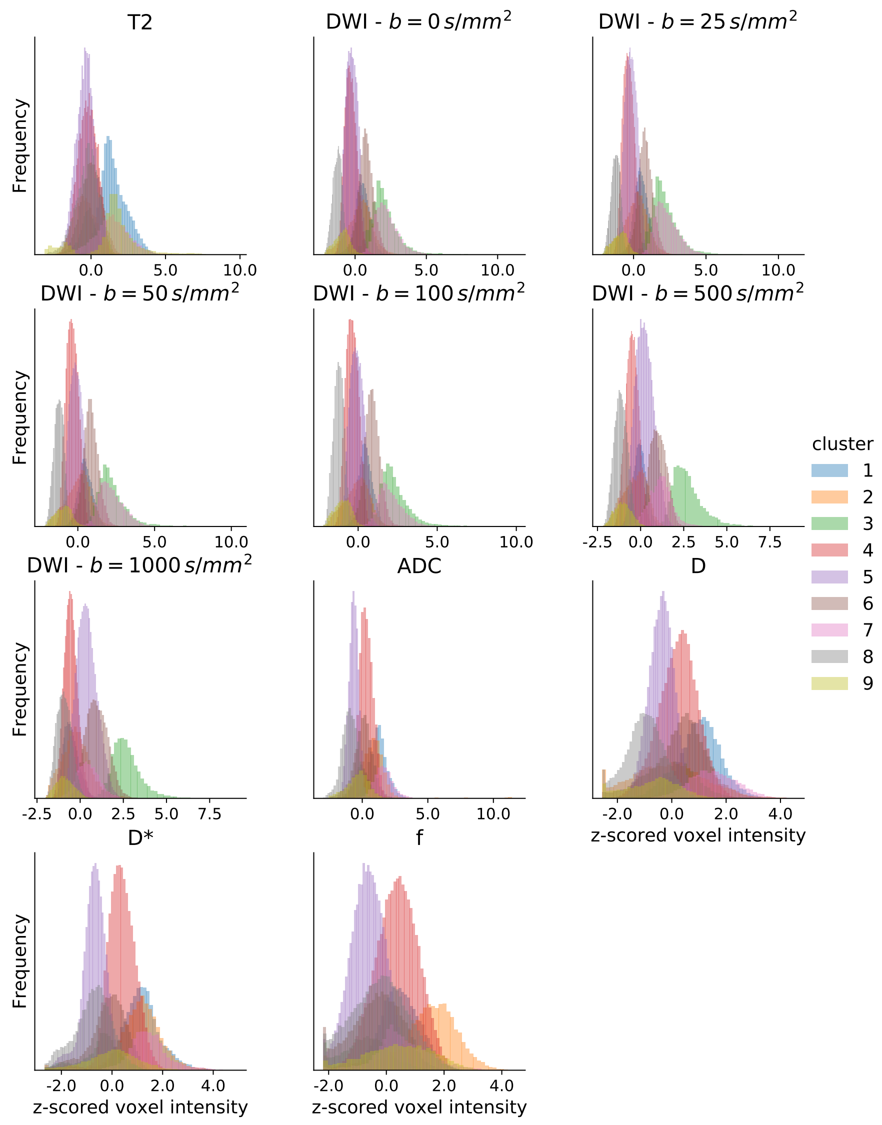


Figure 28: Histogram showing how the each component of the Gaussian mixture model with 9 components maps back onto the original features shows as a histogram in 25. Values on the frequency axis are omitted since the relative frequency between components are of the greatest importance.

5 Discussion

5.1 Validity of the Clustering Solutions

Noise is an inherent property of medical imaging modalities. As was found in the KDE and scatter plots for the principal components of the dataset (figure 26), it was hard to make a visual assessment with regards to a reasonable number of cluster. Using CVIs CH, DB, DB* and DB** for the k-means clustering and BIC for the GMM clustering the same problem was encountered. The CVIs used for the k-means clustering showed a minimum for $k = 2$. However, as mentioned earlier, $k = 2$ was also the lowest tested k. It is important to keep in mind that the CVIs used in the k-means analysis do not compare a clustering solutions to the null hypothesis that there are no distinguishable clusters in the data.

A study conducted by Andersen et al. [17] used k-means cluster analysis to partition dynamic contrast enhanced (DCE) MRI images with the goal of grouping tumour regions with similar vascularization characteristics. The identified region was related to primary tumour control. Another study by Grøndahl [18] used k-means cluster analysis to partition tumour voxels based on Toft parameters, Brix parameters and the relative signal increase. Both studies used a validity index proposed by Kim et al. [67]. In a comparison study conducted by Kim et al. [71] this index was shown to be outperformed by the proposed DB* and DB** indices. The index was left out in extensive testing of CVIs performed Arbelaitz et al. [65] due to requiring a normalization process prior to the application.

In the GMM clustering solution a similar issue was encountered. When calculating the BIC for cluster solutions with different number of components, n_c, no convergence to a global minimum was observed for $n_c \in \{1, 2, \ldots, 70\}$. This number of components way exceeds the number of tissue types it is reasonable to think one can discern using the given dataset. Here it must be mentioned that a tissue type or even a cluster in the data is not necessarily the same as a component in a GMM clustering solution. As pointed out by Baudry et al. [61] a single cluster can be poorly fitted by Gaussian distributions. Especially in cases where clusters are non-Gaussian of nature two or more Gaussian components might be needed to satisfyingly represent a cluster. An illustration of this point is shown by a small simulation study in section A.10 of the appendix. It is argued that since the the BIC is asymptotically consistent, that is it will find the true model if it is among the candidate model as the number of samples tends to infinity [114, 115]. There is no reason to believe that the dataset consists of samples drawn from pure multivariate Gaussian distributions, and thus the BIC is not expected to find the true number of components in this analysis. However, as mentioned by Xu et al. [51] and Everitt [46], the goal of clustering is not to provide an accurate characterization of the underlying probability distribution, and it should be judged based mostly on its usefulness, rather than whether its "True" or "False".

Multiple solutions has been proposed to relate GMM components belonging to the same cluster. Li [116] proposed using the mean vectors of the GMM clustering solution weighted according to the mixing proportion in a k-means algorithm to collect component into clusters. However, this solution requires that the number of clusters is known a priori. Baudry et al. [61] proposed an alternative solution in combining the components hierarchically according to an entropy criterion. None of these solutions were tested in this study, but they might be interesting proposals for further analysis.

Based on the observation that the BIC did not converge to a minimum might also warrant using mixture models with different components than the multivariate Gaussian components considered in this thesis. Another way of finding a suitable number of components may also be using different CVIs than the BIC. Some CVIs that may be considered, and that are implemented Mixmod, a popular software package for unsupervised classification, are the integrated completed likelihood, normalized entropy criterion, cross validation and double cross validation [117].

Two recent studies examining tumour heterogeneity using GMM models have proposed using visual inspection of the data [118] or using the knee of the log-likelihood function [16] as criteria for selecting the number of components. Visual inspection as a criteria for selecting the number of components is feasible when the number of features is low. A dataset, like the one used in this study, containing a larger number of features may not have its structure accurately represented by two- or three-dimensional marginal views [46]. The latter selection criteria used by Jalnefjord et al. [16] relates to the selection criteria used in this study as the BIC is simply the log-likelihood with a penalty term dependent on the number of components used to fit the model.

The repeated GMM clustering solutions using different samples from the dataset as input was, as shown in table 5, found to have excellent correspondence in the predicted labels. This serves to validate the selected clustering a solution in terms of detecting patterns in the dataset. Moreover, if the clustering would not perform similarly on subset of samples drawn from the same dataset, it would be no reason to think the clustering solution would be applicable to similar datasets acquired by other institutions. This may be considered the next logical step with regards to assessing the validity of the given clustering solution.

The stability of a particular clustering solution is also dependent on the delineated tumour volume as it determines which voxels are included in the dataset. Tumour delineation is known to be associated with significant inter- and intraobserver variation [119, 120]. P. Franco et al. [121] had 13 physicians delineate the gross tumour volume of two patients with locally advanced rectal cancer on CT images and compared the delineated volumes to those delineated by an expert observer. The sample mean of the Sørensen-Dice coefficient between the physicians and the expert were found to be 0.80 and 0.65 for the two patients respectively. The Sørensen-Dice coefficient is defined in section A.12 appendix. The stability of the clustering solutions towards inter- and intraobserver variability in tumour delineation requires further research.

No noise filtering of the MRI images or parameter maps was performed in this study. In the histograms, like the ones found in figures 25 and 27, an upper limit along the first axes of all features were set to the 99.95 percentile to remove the effects of outliers. However, this was not applied to the data prior to clustering. In a similar clustering study conducted by Torheim et al. [19] feature values that were considered unphysiologically high were removed prior to cluster analysis. Other proposed solutions have been to apply mean filters [18, 17] and median filters [16] with varying kernel sizes and shapes to the images.

Which features, in this case image sequences and parameter maps, that are included in the clustering analysis influence the obtained clustering solutions. Using different features as input parameters than the ones used in this study might yield different, but equally valid clusters. Based on the sequences included in this study it is expected that the cluster separation will be due to different diffusion, perfusion and T2 values of the tissue. Parameter maps derived from DCE MRI has been used in similar clsuter analysis studies where clusters where found whose cluster volume fractions related to local tumour control [17]. Which features are most meaningful to obtain the desired contrast between tissue types within the tumour volume of the given tumour, and thereby the given clustering solution, requires further analysis.

In the *k*-means clustering analysis the separation between the two clusters is best explained by the DWIs for low *b*-values (from b=0 s mm^{-2} to b=100 s mm^{-2}) as shown in the histogram 27. A difference in the feature mean between the two clusters is also visible in DWI with higher *b*-values (500 s mm^{-2} and 1000 s mm^{-2}), ADC, D and to some degree D*. Almost no separation between the clusters is seen due to f. Jalnefjord et al. [16] used IVIM features in a GMM clustering analysis of fourteen mice with human neuroendocrine tumors. The optimal number of components in the study was found to be two, where one component had low diffusion and high perfusion and the other high diffusion and low perfusion. Due to utilizing different input features, a different clustering algorithm and a different tumour, the results do in no way contradict the results obtained in this study. Which sequences are most relevant to include in a the cluster analysis depends on the end goal of the user [15] and on the dataset from which the features are derived.

5.2 Survival analysis

No studies was found by the author using similar clustering algorithms, MRI sequences and parameter maps to partition tumour volumes for patients with colorectal cancer. Searches were performed in the PubMed and google scholar databases. The study can nevertheless be compared to results obtained by histogram analysis and volumetry. Most studies on histogram analysis and volumetry found by the author explored the CRT response, and only a few patient cohorts included patients that did not receive preoperative CRT.

In the *k*-means clustering analysis a significant association was found with logrank test and univariate cox regression between the the volume of component 1 and PFS in the No CRT group. This component was characterized with by both above average ADC, D and D* values. Necrotic tissues are known to have high ADC values, but normally also exhibit low perfusion values [122]. The given clustering solution does then not appear to partition necrotic tissue efficiently as the perfusion fraction histograms of the two clusters are almost entirely overlapping. In the univariate Cox analysis both the volume of component one and the total volume were significantly related to the PFS in the No CRT group. There was however not sufficient statistical evidence to state that either of these were a better prognostic factor than the other.

No significant associations were found from the CRT group in the k-means clustering analysis. In a review study by Xie et al. [122] on the use of the ADC in predicting the CRT response in locally advanced rectal cancer it was found that conclusions on ADC varied. It was argued that ADC measurements can be high in both good and bad responders, and it was argued that ADC post treatment or the difference in ADC between pre- and post treatment was more promising as measures of CRT response. Since the k-means yielded clusters that were fairly well separated along the ADC axis it is expected that the results will be compatible with the conclusions of the study by Xie et al. [122].

In the GMM partitioning more components are considered than the number of clusters considered for the k-means case. This causes interpretability of each individual component to become more difficult. A significant difference in PFS was found by the logrank test between patients with low and high volume of component 6 and 7 for the No CRT group. Univariate Cox regression on the same components no longer yield significant results at the Bonferroni corrected significance level. The Cox regression is expected to better explain the general trend as it is not dependent on setting a particular volume threshold to partition the patients. A better threshold than the median cluster volume can perhaps be found through reciever operating characteristic (ROC) curve analysis. The exact opposite occurs for the volume of component 1 in the No CRT group, where no significant difference in PFS was found in the logrank test, but a significant association was found using univariate Cox. Its confidence interval does not overlap with the corresponding confidence interval of the total volume presented in table 3. Thus the volume of component 1 might be a better prognostic factor for PFS than the total volume for patients not recieving CRT.

It is here noted that the study contains fairly few patients within each treatment group. This is especially prominent within the CRT group containing only 24 patients. From the range of the follow-up times described in table 1 we can see that some patients have been followed only for a short amount of time. These two factors limit the statistical power of the study.

An additional factor that may contribute to increased probability of making type II errors is the choice of multiple comparison correction. The Bonferroni correction controls the Familywise error rate, that is the probability of having at least one false positive [123, 124]. However, in many cases it has been argued that the correction is to conservative [125, 124]. This is especially true when test are positively correlated [126]. The correlation between the component volumes considered was in some cases quite strong (Pearson correlation coefficient $>$ 0.7, visualized in figures 64a and 64b in the appendix). Due to the combined effect of fairly low statistical power due to the number of patients, the short follow-up times and the Bonferroni correction performed on correlated volume components a considerable risk of type II errors may be associated with this study.

The low statistical power of the CRT group and the conservative nature of the Bonferroni correction especially prominent for positively correlated causes the CRT group to be vulnerable to type II errors. Thus, even though not statistically significant at the Bonferroni corrected significance level, it is noted that the volume of component 3 in the CRT group is associated with a fairly large relative risk. It would be of interest to see if this trend continues if the volume of component 3 is tested for association with PFS in a patient cohort of greater statistical power.

In a study conducted by Bakke et al. [10] a significant association was found between the tumour volume and the CRT response as measured by the five year PFS. The difference in PFS was tested using the logrank test where the best volume cut-off between patient groups was found using ROC analysis. The fairly low statistical power of patient cohort might be one of the reasons that a significant association was not found between the total tumour volume in the CRT patient group and PFS in this study. Even though the study by Bakke et al. [10] contained a similar number of patients, 27 as compared to the 24 in the CRT patient group of this study, the median patient follow-up time was more than three times as long. The cut-off value found by Bakke et al. [10] is also expected to perform better than the fairly arbitrary cut-off of the median total volume used in this study.

The assumptions of the logrank test and the KM estimate are stated in section 2.5 of the theory. Of particular interest in survival analysis studies on cancer is the assumption that the event occurs at the specified time [89]. The time to events in this study is defined as the time until a local relapse or metastasis has been found or the patient dies. The exact time at which a local relapse or metastasis occurs for a patient is not known only that it has occurred sometime between two examinations. Deviations from the assumptions are most important if they are satisfied differently by the different groups being compared [88]. In this study a difference in the way that the assumptions are satisfied between the two patient groups would occur if one group has a larger fraction events occurring being death than the other patient group. The patient group having a larger fraction of patients where local relapse or metastasis occurring as the first event will have a survival probability that is biased upwards as compared to a patient group having a larger fraction of patient where death is the first occurring event. The extent to which this is the case in the current study has not be been analysed.

5.3 Suggestions for further research

For clustering analysis to be useful in the clinic the stability of the clustering solution must be analysed. Both in the sense that the same clusters are found in a dataset acquired by other institutions using slightly different hardware and sequence parameters and in the sense that the clusters must be stable to inter- and intraobserver variability in the tumour delineation. These areas require further research.

The CRT group contained a rather small number of patients with a median follow-up time of less than two years. Thus it was argued that there might be a considerable risk of type II errors. A study with a patient cohort receiving CRT with greater statistical power is recommended for further analysis. Of the components uncovered in the CRT analysis, component 3 seems especially promising. In a follow-up study only promising components need to be tested and thus less corrections due to multiple comparrisons will be needed.

Mapping the cluster labels back onto the images and examining it in the context of the anatomy revealed in the T2w images might give better insight into the nature of the components. In addition this will show whether voxels belonging to one cluster or component are gathered in one subvolume or a small number of subvolumes within the cluster, or whether voxels are spread out as a large number of small islets in the tumour volume. Matlab code was written to map each component or cluster back onto the tumour mask using different colors. The slices were unpacked as an image montage. However, the code was not used in this study. Code and examples of produced images are given in section A.3.5 of the appendix.

6 Conclusion

The results of the analysis shown that cluster analysis based on mpMRI and derived parameter maps using k-means and GMM clustering algorithms were able to find tumour subvolumes significantly related to the PFS in patients not receiving CRT. In particular the volume increase due to one component in the GMM clustering solution was found to be associated wit a greater relative risk than a similar increase in volume of the total tumour volume.

No significant associations were found between any cluster (k-means) or component (GMM) and the PFS in the CRT group. It was however noted that this group may have limited statistical power due to a relatively low number of patients and short follow-up times.

References

[1] *Cancer Registry of Norway. Cancer in Norway 2017 - Cancer incidence, mortality, survival and prevalence in Norway.* Oslo, 2018.

[2] Helsedirektoratet. *Pakkeforløp for tykk- og endetarmskreft IS-2519.* eng. 2016.

[3] E. Furey and K. S. Jhaveri. “Magnetic resonance imaging in rectal cancer”. In: *Magn Reson Imaging Clin N Am* 22.2 (May 2014), pp. 165–190.

[4] Sonia P Li and Anwar R Padhani. “Tumor response assessments with diffusion and perfusion MRI”. In: *Journal of Magnetic Resonance Imaging* 35.4 (2012), pp. 745–763.

[5] Roland Bammer. “Basic principles of diffusion-weighted imaging”. In: *European Journal of Radiology* 45.3 (2003), pp. 169–184. ISSN: 0720-048X. DOI: https://doi.org/10.1016/S0720-048X(02)00303-0. URL: http://www.sciencedirect.com/science/article/pii/S0720048X02003030.

[6] Anwar R Padhani et al. “Diffusion-weighted magnetic resonance imaging as a cancer biomarker: consensus and recommendations”. In: *Neoplasia* 11.2 (2009), pp. 102–125.

[7] NM Desouza et al. “Diffusion-weighted magnetic resonance imaging: a potential non-invasive marker of tumour aggressiveness in localized prostate cancer”. In: *Clinical radiology* 63.7 (2008), pp. 774–782.

[8] R. G. H. Beets-Tan et al. “Magnetic resonance imaging for clinical management of rectal cancer: Updated recommendations from the 2016 European Society of Gastrointestinal and Abdominal Radiology (ESGAR) consensus meeting”. In: *Eur Radiol* 28.4 (Apr. 2018), pp. 1465–1475.

[9] D. Le Bihan et al. “MR imaging of intravoxel incoherent motions: application to diffusion and perfusion in neurologic disorders”. In: *Radiology* 161.2 (Nov. 1986), pp. 401–407.

[10] Kine Mari Bakke et al. “Diffusion-weighted magnetic resonance imaging of rectal cancer: tumour volume and perfusion fraction predict chemoradiotherapy response and survival”. In: *Acta Oncologica* 56.6 (2017), pp. 813–818.

[11] D. Le Bihan et al. “Separation of diffusion and perfusion in intravoxel incoherent motion MR imaging”. In: *Radiology* 168.2 (Aug. 1988), pp. 497–505.

[12] Melissa R Junttila and Frederic J de Sauvage. “Influence of tumour micro-environment heterogeneity on therapeutic response”. In: *Nature* 501.7467 (2013), p. 346.

[13] Fabiana Bettoni et al. “Intratumoral Genetic Heterogeneity in Rectal Cancer: Are Single Biopsies representative of the entirety of the tumor?” In: *Annals of surgery* 265.1 (2017), e4–e6.

[14] Lejla Alic, Wiro J Niessen, and Jifke F Veenland. “Quantification of heterogeneity as a biomarker in tumor imaging: a systematic review”. In: *PloS one* 9.10 (2014), e110300.

[15] Anil K. Jain. “Data clustering: 50 years beyond K-means”. In: *Pattern Recognition Letters* 31.8 (2010). Award winning papers from the 19th International Conference on Pattern Recognition (ICPR), pp. 651–666. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2009.09.011. URL: http://www.sciencedirect.com/science/article/pii/S0167865509002323.

[16] Oscar Jalnefjord et al. “Data-driven identification of tumor subregions based on intravoxel incoherent motion reveals association with proliferative activity”. In: *Magnetic resonance in medicine* (2019).

[17] Erlend KF Andersen et al. “Pharmacokinetic analysis and k-means clustering of DCEMR images for radiotherapy outcome prediction of advanced cervical cancers”. In: *Acta Oncologica* 50.6 (2011), pp. 859–865.

[18] Aurora Rosvoll Grøndahl. *Analysis of Dynamic Contrast Enhanced MRI of Cervical Cancers*. 2015.

[19] Turid Torheim et al. “Cluster analysis of dynamic contrast enhanced MRI reveals tumor subregions related to locoregional relapse for cervical cancer patients”. In: *Acta oncologica* 55.11 (2016), pp. 1294–1298.

[20] Raymond W Ruddon. *Cancer Biology*. eng. 4th ed. Oxford University Press, 2007. ISBN: 0195175441.

[21] D. Hanahan and R. A. Weinberg. “Hallmarks of cancer: the next generation”. In: *Cell* 144.5 (Mar. 2011), pp. 646–674.

[22] P. C. Nowell. “The clonal evolution of tumor cell populations”. In: *Science* 194.4260 (Oct. 1976), pp. 23–28.

[23] Mel Greaves and Carlo C. Maley. “Clonal evolution in cancer”. In: *Nature* 481.7381 (2012). ISSN: 0028-0836.

[24] Charles Swanton. “Intratumor heterogeneity: evolution through space and time”. eng. In: *Cancer research* 72.19 (2012). ISSN: 1538-7445.

[25] Andriy Marusyk and Kornelia Polyak. “Tumor heterogeneity: Causes and consequences”. eng. In: *BBA - Reviews on Cancer* 1805.1 (2010), pp. 105–117. ISSN: 0304-419X.

[26] M. W. Schmitt, M. J. Prindle, and L. A. Loeb. “Implications of genetic heterogeneity in cancer”. In: *Ann. N. Y. Acad. Sci.* 1267 (Sept. 2012), pp. 110–116.

[27] L. A. Loeb, C. F. Springgate, and N. Battula. “Errors in DNA replication as a basis of malignant changes”. In: *Cancer Res.* 34.9 (Sept. 1974), pp. 2311–2321.

[28] Bendik Skarre Abrahamsen. *Segmentation of tumour volume in rectal cancer trough linear discriminant analysis*. Project thesis conducted at Norwegian University of Science and Technology submitted fall 2018. 2018.

[29] Lars G. Hanson. "Is quantum mechanics necessary for understanding magnetic resonance?" eng. In: *Concepts in Magnetic Resonance Part A* 32.5 (2008), pp. 329–340. ISSN: 1546-6086.

[30] *Spin Echo*. http://mriquestions.com/spin-echo1.html. Online; accessed 26.11.2018. 2018.

[31] F. Bloch. "Nuclear Induction". eng. In: *Physical Review* 70.7-8 (1946), pp. 460–474. ISSN: 0031-899X.

[32] E. O. Stejskal and J. E. Tanner. "Spin Diffusion Measurements: Spin Echoes in the Presence of a Time-Dependent Field Gradient". In: *The Journal of Chemical Physics* 42.1 (1965), pp. 288–292. DOI: 10.1063/1.1695690. eprint: https://doi.org/10.1063/1.1695690. URL: https://doi.org/10.1063/1.1695690.

[33] D. M. Koh, D. J. Collins, and M. R. Orton. "Intravoxel incoherent motion in body diffusion-weighted MRI: reality and challenges". In: *AJR Am J Roentgenol* 196.6 (June 2011), pp. 1351–1361.

[34] Nicholas C. Gourtsoyiannis, ed. *Clinical MRI of the Abdomen*. 1st ed. Springer-Verlag Berlin Heidelberg, 2011. ISBN: e-book: 978-3-540-85689-4, Hardcover: 978-3-540-85688-7, Softcover: 978-3-662-51882-3.

[35] D. Le Bihan. "What can we see with IVIM MRI?" In: *Neuroimage* (Dec. 2017).

[36] Tom M Mitchell. *Machine learning*. eng. New York, 1997.

[37] M. Kohli et al. "Implementing Machine Learning in Radiology Practice and Research". In: *AJR Am J Roentgenol* 208.4 (Apr. 2017), pp. 754–760.

[38] S. Wang and R. M. Summers. "Machine learning and radiology". In: *Med Image Anal* 16.5 (July 2012), pp. 933–951.

[39] Masashi Sugiyama. "Chapter 1 - Statistical Machine Learning". In: *Introduction to Statistical Machine Learning*. Ed. by Masashi Sugiyama. Boston: Morgan Kaufmann, 2016, pp. 3–8. ISBN: 978-0-12-802121-7. DOI: https://doi.org/10.1016/B978-0-12-802121-7.00012-1. URL: http://www.sciencedirect.com/science/article/pii/B9780128021217000121.

[40] Gareth James. *An Introduction to Statistical Learning : with Applications in R*. eng. New York, NY, 2013.

[41] Trevor Hastie, Jerome Friedman, and Robert Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. eng. Springer Series in Statistics, New York, NY, 2001. ISBN: 9780387216065.

[42] *Data Mining and Knowledge Discovery Handbook*. eng. 2nd ed. Springer series in solid-state sciences Magnetic bubble technology. Boston, MA, 2010. ISBN: 1-282-98082-3.

[43] "Bias-Variance Trade-offs". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 110–110. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_76. URL: https://doi.org/10.1007/978-0-387-30164-8_76.

[44] Richard O Duda. *Pattern classification*. eng. 2nd ed. New York: Wiley, 2001. ISBN: 0471056693.

[45] I. T. Jolliffe and J. Cadima. "Principal component analysis: a review and recent developments". In: *Philos Trans A Math Phys Eng Sci* 374.2065 (Apr. 2016), p. 20150202.

[46] Brian S Everitt. *Cluster Analysis*. eng. 5th edition. Vol. v.886. Wiley Series in Probability and Statistics. Chicester, 2010. ISBN: 1-280-76795-2.

[47] MATLAB®. *Matlab's implementation of Principal Component Analysis*. URL: https://se.mathworks.com/help/stats/pca.html (visited on 04/17/2019).

[48] Scikit-Learn. *Scikit-Learn's implementation of Principal Component Analysis*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (visited on 04/17/2019).

[49] James E. Gentle. *Matrix Algebra: Theory, Computations and Applications in Statistics*. eng. Springer Texts in Statistics. Cham: Springer International Publishing, 2017. ISBN: 978-3-319-64866-8.

[50] Jacob T. Vanderplas. *Python data science handbook : essential tools for working with data*. eng. First edition. Beijing, China, 2017. ISBN: 1-4919-1205-7.

[51] Rui Xu and Donald C. Wunsch. *Clustering*. IEEE Series on Computational Intelligence. Wiley-IEEE Press, 2009. ISBN: 9780470276808. URL: http://search.ebscohost.com/login.aspx?direct=true&db=e230xww&AN=254099&site=ehost-live.

[52] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. "Cluster Validity Methods: Part I". In: *SIGMOD Rec.* 31.2 (June 2002), pp. 40–45. ISSN: 0163-5808. DOI: 10.1145/565117.565124. URL: http://doi.acm.org/10.1145/565117.565124.

[53] H. Kashima et al. "K-means clustering of proportional data using L1 distance". In: *2008 19th International Conference on Pattern Recognition*. Dec. 2008, pp. 1–4. DOI: 10.1109/ICPR.2008.4760982.

[54] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. eng. 3rd ed. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011. ISBN: 0123814790.

[55] Hossein Pishro-Nik. *Introduction to probability, statistics, and random processes*. Kappa Research LLC, 2014. ISBN: 978-0-9906372-0-2.

[56] Arthur P Dempster, Nan M Laird, and Donald B Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22.

[57] Sabhia Firdaus and Md Ashraf Uddin. "A survey on clustering algorithms and complexity analysis". In: *International Journal of Computer Science Issues (IJCSI)* 12.2 (2015), p. 62.

[58] MATLAB®. *Matlab's implementation of fitting a Gaussian mixture model to data*. URL: https://se.mathworks.com/help/stats/fitgmdist.html (visited on 04/21/2019).

[59] Scikit-Learn. *Scikit-Learn's implementation of the Gaussian mixture model class*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html (visited on 04/21/2019).

[60] Preeti Arora, Deepali, and Shipra Varshney. "Analysis of K-Means and K-Medoids Algorithm For Big Data". In: *Procedia Computer Science* 78 (2016). 1st International Conference on Information Security & Privacy 2015, pp. 507–512. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2016.02.095. URL: http://www.sciencedirect.com/science/article/pii/S1877050916000971.

[61] Jean-Patrick Baudry et al. "Combining Mixture Components for Clustering". In: *Journal of Computational and Graphical Statistics* 19.2 (2010), pp. 332–353. DOI: 10.1198/jcgs.2010.08111. eprint: https://doi.org/10.1198/jcgs.2010.08111. URL: https://doi.org/10.1198/jcgs.2010.08111.

[62] Emanuel Parzen. "On Estimation of a Probability Density Function and Mode". eng. In: *Ann. Math. Statist.* 33.3 (1962), pp. 1065–1076. ISSN: 0003-4851.

[63] Murray Rosenblatt. "Remarks on Some Nonparametric Estimates of a Density Function". eng. In: *The Annals of Mathematical Statistics* 27.3 (1956), pp. 832–837. ISSN: 00034851.

[64] Artur Gramacki. *Nonparametric Kernel Density Estimation and Its Computational Aspects*. Springer International Publishing, 2018. ISBN: 3-319-71688-3.

[65] Olatz Arbelaitz et al. "An extensive comparative study of cluster validity indices". In: *Pattern Recognition* 46.1 (2013), pp. 243–256. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2012.07.021. URL: http://www.sciencedirect.com/science/article/pii/S003132031200338X.

[66] Tadeusz Caliński and Jerzy Harabasz. "A dendrite method for cluster analysis". In: *Communications in Statistics-theory and Methods* 3.1 (1974), pp. 1–27.

[67] Do-Jong Kim, Yong-Woon Park, and Dong-Jo Park. “A novel validity index for determination of the optimal number of clusters”. In: *IEICE Transactions on Information and Systems* 84.2 (2001), pp. 281–285.

[68] David L Davies and Donald W Bouldin. “A cluster separation measure”. In: *IEEE transactions on pattern analysis and machine intelligence* 2 (1979), pp. 224–227.

[69] Robert Tibshirani, Guenther Walther, and Trevor Hastie. “Estimating the number of clusters in a data set via the gap statistic”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.2 (2001), pp. 411–423.

[70] Peter J Rousseeuw. “Silhouettes: a graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of computational and applied mathematics* 20 (1987), pp. 53–65.

[71] Minho Kim and R.S. Ramakrishna. “New indices for cluster validity assessment”. In: *Pattern Recognition Letters* 26.15 (2005), pp. 2353–2363. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2005.04.007. URL: http://www.sciencedirect.com/science/article/pii/S016786550500125X.

[72] Caio Flexa et al. “Mutual equidistant-scattering criterion: A new index for crisp clustering”. In: *Expert Systems with Applications* 128 (2019), pp. 225–245. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2019.03.027. URL: http://www.sciencedirect.com/science/article/pii/S0957417419301897.

[73] Yanchi Liu et al. “Understanding of internal clustering validation measures”. In: *2010 IEEE International Conference on Data Mining*. IEEE. 2010, pp. 911–916.

[74] C. Flexa et al. “A Novel Equidistant-Scattering-Based Cluster Index”. In: *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*. Oct. 2018, pp. 540–545. DOI: 10.1109/BRACIS.2018.00099.

[75] Ujjwal Maulik and Sanghamitra Bandyopadhyay. “Performance evaluation of some clustering algorithms and validity indices”. In: *IEEE Transactions on pattern analysis and machine intelligence* 24.12 (2002), pp. 1650–1654.

[76] M. Halkidi and M. Vazirgiannis. “Clustering validity assessment: finding the optimal partitioning of a data set”. In: *Proceedings 2001 IEEE International Conference on Data Mining*. Nov. 2001, pp. 187–194. DOI: 10.1109/ICDM.2001.989517.

[77] Slobodan Petrovic. “A comparison between the silhouette index and the davies-bouldin index in labelling ids clusters”. In: *Proceedings of the 11th Nordic Workshop of Secure IT Systems*. sn. 2006, pp. 53–64.

[78] Hirotogu Akaike. “Information theory and an extension of the maximum likelihood principle”. In: *Selected papers of hirotugu akaike*. Springer, 1998, pp. 199–213.

[79] Gideon Schwarz et al. “Estimating the dimension of a model”. In: *The annals of statistics* 6.2 (1978), pp. 461–464.

[80] C. Biernacki, G. Celeux, and G. Govaert. “Assessing a mixture model for clustering with the integrated completed likelihood”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.7 (July 2000), pp. 719–725. ISSN: 0162-8828. DOI: 10.1109/34.865189.

[81] and P. S. Gopalakrishnan. “Clustering via the Bayesian information criterion with applications in speech recognition”. In: *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*. Vol. 2. May 1998, 645–648 vol.2. DOI: 10.1109/ICASSP.1998.675347.

[82] Jonathan G Campbell et al. “Linear flaw detection in woven textiles using model-based clustering”. In: *Pattern Recognition Letters* 18.14 (1997), pp. 1539–1548.

[83] Abhijit Dasgupta and Adrian E Raftery. “Detecting features in spatial point processes with clutter via model-based clustering”. In: *Journal of the American statistical Association* 93.441 (1998), pp. 294–302.

[84] Odd Aalen, Ornulf Borgan, and Hakon Gjessing. *Survival and event history analysis: a process point of view*. Springer Science & Business Media, 2008.

[85] P Armitage. *Statistical methods in medical research*. eng. 4th ed. Oxford: Blackwell, 2002. ISBN: 0632052570.

[86] Viv Bewick, Liz Cheek, and Jonathan Ball. “Statistics review 12: Survival analysis”. In: *Critical Care* 8.5 (Sept. 2004), p. 389. ISSN: 1364-8535. DOI: 10.1186/cc2955. URL: https://doi.org/10.1186/cc2955.

[87] Jerald F Lawless. *Statistical models and methods for lifetime data*. eng. 2nd ed. Wiley series in probability and statistics. Hoboken, N.J: Wiley-Interscience, 2003. ISBN: 0471372153.

[88] J Martin Bland and Douglas G Altman. “The logrank test”. In: *BMJ* 328.7447 (2004), p. 1073. ISSN: 0959-8138. DOI: 10.1136/bmj.328.7447.1073. eprint: https://www.bmj.com/content/328/7447/1073.full.pdf. URL: https://www.bmj.com/content/328/7447/1073.

[89] J Martin Bland and Douglas G Altman. “Survival probabilities (the Kaplan-Meier method)”. In: *BMJ* 317.7172 (1998), pp. 1572–1580. ISSN: 0959-8138. DOI: 10.1136/bmj.317.7172.1572. eprint: https://www.bmj.com/content/317/7172/1572.full.pdf. URL: https://www.bmj.com/content/317/7172/1572.

[90] Sang-Gue Park et al. “Logrank test for bivariate survival data”. In: *Communications in Statistics - Simulation and Computation* 29.2 (2000), pp. 533–540. DOI: 10.1080/03610910008813626. eprint: https://doi.org/10.1080/03610910008813626. URL: https://doi.org/10.1080/03610910008813626.

[91] Stephen W Lagakos. “Time-to-event analyses for long-term treatments—the APPROVe trial”. In: *New England Journal of Medicine* 355.2 (2006), pp. 113–117.

[92] P. Royston et al. “Combined test versus logrank/Cox test in 50 randomised trials”. In: *Trials* 20.1 (Mar. 2019), p. 172.

[93] David R Cox. “Regression models and life-tables”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 34.2 (1972), pp. 187–202.

[94] *The OxyTarget Study - Functional MRI of Hypoxia-Mediated Rectal Cancer Aggressiveness*. 2018. URL: http://www.acredit.no/the-oxytarget-study/ (visited on 11/16/2018).

[95] *Read nifti files with matlab #210*. https://github.com/rordenlab/dcm2niix/issues/210. Online; accessed 16.11.2018. 2018.

[96] E. Demidenko. “The next-generation K-means algorithm”. In: *Stat Anal Data Min* 11.4 (Aug. 2018), pp. 153–166.

[97] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[98] The MathWorks, Inc. *Matlab File Exchange*. 2019. URL: https://se.mathworks.com/matlabcentral/fileexchange/ (visited on 05/02/2019).

[99] Python Software Foundation. *Python Package Index (PyPI)*. 2019. URL: https://pypi.org/ (visited on 05/02/2019).

[100] David Arthur and Sergei Vassilvitskii. “k-means++: The advantages of careful seeding”. In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2007, pp. 1027–1035.

[101] Jianhua Lin. “Divergence measures based on the Shannon entropy”. In: *IEEE Transactions on Information theory* 37.1 (1991), pp. 145–151.

[102] N. Ramakrishnan and R. Bose. “Analysis of healthy and tumour DNA methylation distributions in kidney-renal-clear-cell-carcinoma using Kullback–Leibler and Jensen–Shannon distance measures”. In: *IET Systems Biology* 11.3 (2017), pp. 99–104. ISSN: 1751-8849. DOI: 10.1049/iet-syb.2016.0052.

[103] Solomon Kullback and Richard A Leibler. “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[104] Michele Tumminello, Fabrizio Lillo, and Rosario N Mantegna. “Kullback-Leibler distance as a measure of the information filtered from multivariate data”. In: *Physical Review E* 76.3 (2007), p. 031123.

[105] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[106] AP Majtey, PW Lamberti, and DP Prato. “Jensen-Shannon divergence as a measure of distinguishability between mixed quantum states”. In: *Physical Review A* 72.5 (2005), p. 052310.

[107] Michel Marie Deza and Elena Deza. “Encyclopedia of distances”. In: *Encyclopedia of distances*. Springer, 2009, pp. 1–583.

[108] Jacob Cohen. “A coefficient of agreement for nominal scales”. In: *Educational and psychological measurement* 20.1 (1960), pp. 37–46.

[109] Paul Jaccard. “Étude comparative de la distribution florale dans une portion des Alpes et des Jura”. In: *Bull Soc Vaudoise Sci Nat* 37 (1901), pp. 547–579.

[110] Kay Henning Brodersen et al. “The balanced accuracy and its posterior distribution”. In: *2010 20th International Conference on Pattern Recognition*. IEEE. 2010, pp. 3121–3124.

[111] Cameron Davidson-Pilon et al. *CamDavidsonPilon/lifelines: v0.21.1*. Apr. 2019. DOI: 10.5281/zenodo.2652543. URL: https://doi.org/10.5281/zenodo.2652543.

[112] William S Noble. “How does multiple testing correction work?” In: *Nature biotechnology* 27.12 (2009), p. 1135.

[113] Valentin Amrhein, Sander Greenland, and Blake McShane. *Scientists rise up against statistical significance*. 2019.

[114] S. I. Vrieze. “Model selection and psychological theory: a discussion of the differences between the Akaike information criterion (AIC) and the Bayesian information criterion (BIC)”. In: *Psychol Methods* 17.2 (June 2012), pp. 228–243.

[115] Yuhong Yang. “Can the strengths of AIC and BIC be shared? A conflict between model indentification and regression estimation”. In: *Biometrika* 92.4 (2005), pp. 937–950.

[116] Jia Li. “Clustering based on a multilayer mixture model”. In: *Journal of Computational and Graphical Statistics* 14.3 (2005), pp. 547–568.

[117] MIXMOD *Statistical Documentation*. 2016. URL: http://www.mixmod.org/IMG/pdf/statdoc_2016.pdf (visited on 05/27/2019).

[118] Mathew R Divine et al. “A population-based Gaussian mixture model incorporating 18F-FDG PET and diffusion-weighted MRI quantifies tumor tissue classes”. In: *Journal of Nuclear Medicine* 57.3 (2016), pp. 473–479.

[119] B. Segedin and P. Petric. “Uncertainties in target volume delineation in radiotherapy - are they relevant and what can we do about them?” In: *Radiol Oncol* 50.3 (Sept. 2016), pp. 254–262.

[120] C. F. Njeh. “Tumor delineation: The weakest link in the search for accuracy in radiotherapy”. In: *J Med Phys* 33.4 (Oct. 2008), pp. 136–140.

[121] P. Franco et al. “Variability of clinical target volume delineation for rectal cancer patients planned for neoadjuvant radiotherapy with the aid of the platform Anatom-e”. In: *Clin Transl Radiat Oncol* 11 (June 2018), pp. 33–39.

[122] Haiting Xie et al. “Effectiveness of the apparent diffusion coefficient for predicting the response to chemoradiation therapy in locally advanced rectal cancer: a systematic review and meta-analysis”. In: *Medicine* 94.6 (2015).

[123] Andrew Gelman, Jennifer Hill, and Masanao Yajima. “Why we (usually) don’t have to worry about multiple comparisons”. In: *Journal of Research on Educational Effectiveness* 5.2 (2012), pp. 189–211.

[124] Hervé Abdi. “Bonferroni and Šidák corrections for multiple comparisons”. In: *Encyclopedia of measurement and statistics* 3 (2007), pp. 103–107.

[125] Eurof Walters. “The P-value and the problem of multiple testing”. In: *Reproductive BioMedicine Online* 32.4 (2016), pp. 348–349. ISSN: 1472-6483. DOI: https://doi.org/10.1016/j.rbmo.2016.02.008. URL: http://www.sciencedirect.com/science/article/pii/S1472648316000675.

[126] Daria Salyakina et al. “Evaluation of Nyholt’s procedure for multiple testing correction”. In: *Human heredity* 60.1 (2005), pp. 19–25.

A Appendix

A.1 A guide to the appendix

Due to the rather extensive analysis and testing perform as part of this master's thesis the appendix has become quite large. The appendix consists of most of the code that was used in this study, references and information about packages and software that was used and figures that with value to the analysis, but that were to plentiful to fit in thesis itself. Figures of particular importance are referenced in the text, in addition, most sections in the appendix containing additional figures are also referenced in the text where they are used.

The code presented in the appendix represent to full process from reading in, interpolating and storing the dicom images to the survival analysis performed on the subvolumes obtained in the clustering solutions.

Due to the extensive nature of the appendix, a short description about each of the sections will be given here even though some might be a bit self explanatory.

A.2 Packages and environments: This is a guide to the packages and software used in this study including what the were primarily used for and which versions of the pacakges were used.

A.3 Matlab Code: This section contains a number of MATLAB® functions and scripts used in this thesis. Subsection A.3.1 contains code used for the conversion of dicom images to 3D image stacks in addition to the interpolation of the DWI and derived parameter maps to the resolution of the T2w images and cropping of the images. The author is grateful for lending the functions `InterpolateImage` and `getVoxelCoordinates` written by Fraziska Knuth. Subsection A.3.2 contains matlab code used to superimpose interpolated DWI as heatmaps on T2w images. Examples of these images can be found in figures 17, 19 and 18. In subsection A.3.3 code for another tool for checking alignment of images is shown. This tool makes an image montage of selected sequences where edges of corresponding images are exactly aligned. The code for building the dataset used in the clustering analysis is shown in section A.3.4. By building the dataset it is meant extracting the image intensities for each image sequence and parameter map and unpacking them as a matrix whose dimensionality is equal to the number of features time the number of voxels. In section A.3.5 code is shown for mapping the labels obtained through either clustering solution back onto the tumour volumes and unpacking the slices as an image montage. An example image is also given.

A.4 Python Code This section contains the custom python functions written for use in this thesis. This includes code for the custom plot containing scatter plots above the diagonal, histograms on the diagonal and KDE plots below the diagonal presented in subsection A.4.1. An example of such a plot is presented in 26. Subsection A.4.2 contains the implementation of the CVIs

DB* and DB**. Section A.4.3 contains the code used in z-scoring and performing PCA on the raw dataset. The code used to obtain the k-means and GMM clustering solutions is found in sections A.4.4 and A.4.5 respectively. Code for visually representing the clusters and the survival analysis based on the k-means partitioned volumes is found in section A.4.6. Finding the optimal number of components of the GMM clustering solutions is done with the code shown in section A.4.7. This section also contains the code for performing survival analysis on the dataset partitioned with the GMM clustering algorithm.

A.5 Test of DB, DB* and DB cluster validity indices** is a test comparing the implemented cluster validity indices DB* and DB** against DB.

A.6 Finding the optimal k for the k-means algorithm contains plots of CVIs used to find the optimal k for the k-means algorithm. Calculations were performed both for a dataset including interaction features and without interaction features. This is described in the text. Plots for CH, Sil, DB, DB* and DB** are shown.

A.7 KM estimates with logrank test for the total tumour volume presents the results of survival analysis with Kaplan-Meier estimate with logrank test for the total volume partitioned according to the median volume. The analysis is performed for both treatment groups.

A.8 One component vs. rest histograms for the GMM clustering solution presents histograms where one component of the GMM clustering solution is plotted against the rest of the components along the original z-scored feature axes.

A.9 Survival Curves Of the Gaussian Mixture Model (GMM) components presents the KM estimates and logrank test results for all components obtained through the GMM clustering partitioned according to the median cluster volume.

A.11 Correlation heatmaps presents the correlation matrices for the volumes of the GMM components and the total volume as heatmaps.

A.12 Dice-Sørensen Coefficient gives the definition of the Dice-Sørensen coefficient.

A.2 Packages and environments

Multiples packages and environments have been used in the development of the code for this thesis. For the initial processing and loading of the dicom images MATLAB® was utilized. MATLAB® was also used to extract the ROIs delineated by the radiologists and to extract information from the dicom headers. MATLAB® version R2018b (9.5.0.1033004) was used throughout this thesis.

Initially the clustering was performed in MATLAB® using tools from the *Statistics and Machine Learning Toolbox*. However, it was later made a change to python and all code used to produce the clustering results presented in this thesis was developed using python as programming language.

Python is a free and open-source programming language, and many of the tools that make the programming language easy to use across a wide range of applications are developed by contributors worldwide. A list of the packages and package versions used in this thesis will be given here along with what each individual package was primarily used for. The python version used in this thesis was Python 3.6.5.

Numpy [1] (version 1.14.3) pythons workhorse for scientific computations. In this thesis Numpy to calculations and for storing numbers in an efficient manner. Pandas and Scikit-Learn use Numpy extensively under the hood.

Matplotlib [2] (version 3.0.3) 2D plotting library for python. Used for many of the plots presented in the thesis.

Scikit-Learn [3] (version 0.20.3) , often abbreviated sklearn, is a popular module for machine learning and data analysis in python. It is built on top of Numpy, Scipy and Matplotlib. In this thesis the Scikit-Learn implementation of k-means clustering and GMM was used. In addition the indices DB* and DB** was coded using the Scikit-Learn implementation the DB index as a template. Built in datasets of Scikit-Learn was used to test the implementation of these new indices.

Pandas [4] (version 0.24.2) contains the pandas datas tructure. This is a high performance data strucute that behaves like a spreadsheet. The data structure is built on top of Numpy and contains a vast library of efficient native methods.

Seaborn [5] (version 1.1.0) library for making statistical graphics in python. Seaborn is build on top of Matplotlib and is closely integrated with Pandas data structures. Seaborn is used for a large number of plots in this thesis due to the convenience of its interaction Pandas datas tructures. Seaborn has a extensive gallery of beutiful example plots and a simple API.

Ipython notebook environment [6] **(version 6.4.0)** Now know as the Jupyter Notebook is an interactive computational environment where code, rich text, plot and more can be combined. Most of the python code written for this thesis was developed inside the Jupyter Notebook environment. The environment interacts very well with the pandas data structures making printed tables "pretty".

Scipy [7] **(version 1.1.0)** is a package containing a vast number of functions and methods useful for mathematics, statistics, science and engineering. In this thesis specifically the `scipy.stats` module was used to calculate the Jensen-Shannon distance between Gaussian mixture components.

Lifelines [8] **(version 0.21.1)** is a survival analysis package built on top of Pandas. This package was used for all the survival analysis in the thesis, and for making the Kaplan-Meier estimate plots.

StatsModels [9] **(version 0.9.0)** is a module of classes and functions for the estimation of many different statistical models. In this thesis it was used for multiple comparison corrections as the module contains a wide array of method. Though not strictly necessary due to the simplicity of the Bonferroni correction it was opted for the model still remains a useful tool.

Joblib [10] **(version 0.13.2)** is a set of tool for creating lightweight piplines in python. In this thesis it was used due to its simple API for running loops in parallel.

References

[1] S. Chris Colbert Stéfan van der Walt and Gaël Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation". In: *Computing in Science & Engineering* 13 (2011). DOI: 10.1109/MCSE.2011.37.

[2] John D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science & Engineering* 9 (2007). DOI: 10.1109/MCSE.2007.55.

[3] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[4] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference* (2010).

[5] Michael Waskom et al. *seaborn: statistical data visualization.* URL: https://seaborn.pydata.org/ (visited on 05/24/2019).

[6] F. Perez and B. E. Granger. "IPython: A System for Interactive Scientific Computing". In: *Computing in Science Engineering* 9.3 (May 2007), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53.

[7] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001–. URL: http://www.scipy.org/.

[8] Cameron Davidson-Pilon et al. *CamDavidsonPilon/lifelines: v0.21.1*. Apr. 2019. DOI: 10.5281/zenodo.2652543. URL: https://doi.org/10.5281/zenodo.2652543.

[9] Josef Perktold et al. *StatsModels: Statistics in Python*. 2018. URL: https://www.statsmodels.org/stable/index.html (visited on 05/07/2019).

[10] Joblib developers. *Joblib: running Python functions as pipeline jobs*. 2018. URL: https://joblib.readthedocs.io/en/latest/ (visited on 05/07/2019).

A.3 Matlab Code

A.3.1 Conversion of DICOM images to patient structs with interpolation and cropping

```
% Code to convert patient data to patient structs. In these
 ↪  structs the
% images can be accesed in field names according to methods(T2,
 ↪  DWI, IVIM,
% Binary). The IVIM and Binary have substructures according to
 ↪  submethods
% and the radiologist that have made the delineation.

%% Select which parts of the code you want to run. This also
 ↪  includes options for debugging

test =                      0;
convert =                   0;
interpolate =               0;
flipZdirOfMask =            0; % Now redundant as it is done
 ↪  automatically in the generateStructFields code.
crop =                      1;
writeBvalues =              1;
infoOnly =                  1;
rmvSlicesOutsideDWIFOV =    1; % should maybe be included in the
 ↪  crop function for ease of coding

forceAllPatients =          1;  % option to override the data
                                % from the patients in list
                                % of completed patients
forceMaxPatients =          1; maxPatNum = 3;
                                % Option to force a maximum
                                 ↪  number
                                % converted in a single run
convertSpecific =           1; specificPatient = 'Oxytarget_40
 ↪  PRE';

updatePatsDone =            1;
updatePatsExclude =         1;
sing =                      0;

% The matlab build in niftifunctions have a known bug. We
 ↪  circumvent this
% by adding corrected files higher up in the path
addpath C:\Users\bendi\OneDrive\Documents\Skole\Prosjektoppgave\⌟
 ↪  Kode\CorrectNiftilCode
```

```matlab

%% Define where the raw data is located
inputPath = 'C:\Users\bendi\OneDrive\Documents\Skole\Prosjektoppgave\Kode\Data_all_pats';

if isempty(inputPath)
    inputPath = ...
        uigetdir('','Choose the folder where the dicom files are located');
end
%% Define where to put the different kinds of output data
% in which directory to create all the folders with intermediary and final
% data

basePath = 'C:\Users\bendi\OneDrive\Documents\Skole\Prosjektoppgave\Kode';

if isempty(basePath)
    basePath = ...
        uigetdir('','Choose the folder where the data should be buildt');
end

if ~test
    outputPathPatients = fullfile(basePath,'PatientStructs');
else
    outputPathPatients = fullfile(basePath, 'PatientStructsTest');
end

outputPathRaw = fullfile(outputPathPatients, 'raw');
outputPathInterpolated = fullfile(outputPathPatients, 'interPolated');

outputPathInterpolatedAndCropped = fullfile(outputPathPatients,...
    'interPolatedAndCropped');
outputPathInterpolatedAndCroppedOnly = fullfile(outputPathPatients,...
    'interPolatedAndCroppedOnly');

outputPathInterpolatedAndCroppedOnlyInDWIFOV =...
    fullfile(outputPathPatients, 'interPolatedAndCroppedOnlyInDWIFOV');

```

```
outputPathInfoOnly = ...
    fullfile(outputPathPatients, 'InfoOnly');


filepath = fullfile(outputPathPatients,...
    'patDone.mat'); % Files storing the completed patients
patsExcludePath = fullfile(outputPathPatients,...
    'patExclude.mat'); % Files to exclude due to misalignment



%% Finding patient names and number of patients
patNames = ls(inputPath); patNames = patNames(3:end ,:);

% We make a list of patients that are already done so that we
↪  don't need to
% do these all over again
patNames = cellstr(patNames);

if ~forceAllPatients

[~,~,~] = mkdir(outputPathPatients);


% checking if the file already exists
if isfile(filepath)
    % if so we load it
     load(filepath);
     flag = ismember(lower(patNames),lower(patDone));
     patNames(flag) = [];
     if isempty(patNames)
         % Nothing to convert
         disp('All patients are converted');
         return
     end

else
     % File does not exist and we need to create it.
     patDone = {};
     save(filepath, 'patDone')
end

end

% Exclude patients that are misaligned

```

```
if isfile(patsExcludePath)
    % if so we load it
     load(patsExcludePath);
     flag = ismember(lower(patNames),lower(patExclude));
     patNames(flag) = [];
     if isempty(patNames)
         % Nothing to convert
         disp('No valid patients to convert');
         return
     end

else
     % File does not exist and we need to create it.
     patExclude = {};
     save(patsExcludePath, 'patExclude')
end




patNames = char(patNames);

% Find the number of patients to iterate over
numPats = size(patNames,1);
tic

%% Select which radiologist you want to use as. This changes how
 ↪  images are
% cropped. Cropping based on the extent of a box covering the
 ↪  entire tumour
% volume in the x, y and z direction.

% The avaliable radiologists. Needed for the crop.
radiologists = {'an', 'shh'};
%%
% some of masks seems to be arranged opposite according to
 ↪  zposition than
% the others. We need to flip them. They're flipped if
 ↪  flipZdirOfMask is 1

flipPats = {'oxytarget_32 pre', 'oxytarget_40 pre'};

%% test
% allows us to run the code for just one patient to test that
 ↪  everything
% works. When not testing set test to 0.
```

```

if test == 1
    numPats = 1;
    patNames = patNames(1,:);
end

%% Option to convert just a specific patient. Will overwrite the
↪  data if patient is already converted.
if convertSpecific
    patNames = cellstr(patNames);
    specPatFlag =
    ↪  ismember(lower(patNames),lower(specificPatient));

    if isempty(find(specPatFlag,1))
        disp('Patient not found in directory or is misaligned');
        return
    else
        patNames = patNames(specPatFlag);
    end
    numPats = numel(patNames);
    patNames = char(patNames);
end

%% If the number of patients exeeds the maximum number allowed
↪  reduce number of patients to the maximum.
 if forceMaxPatients
     if (numPats > maxPatNum) && ~convertSpecific
         numPats = maxPatNum;
         patNames = patNames(1:maxPatNum,:);
     end
 end


%% iteration through all patients for converting to structs
if convert ==1
    disp('Converting')
    flagDWIT2Aligned = ones(numPats,1);
for i = 1: numPats
    fprintf('Converting patient number %d out of %d.\n', i,
    ↪  numPats);
    fprintf('Patient name: %s\n', patNames(i,:));
    generateAndSaveStructFields(fullfile(inputPath,patNames(i,:)⌋
    ↪  ), outputPathRaw,
    ↪  patNames(i,:));
    timerval = toc;

    % we need to see wether the DWI images and T2 images are
     ↪  aligned
    load(fullfile(outputPathRaw, patNames(i,:)),'T2','DWI');
    if ~(T2.header{1}.ImageOrientationPatient ==
     ↪  DWI.header{1}.ImageOrientationPatient)
        % add to flag
        flagDWIT2Aligned(i) = 0;
        disp([patNames(i,:) ' is not aligned.'])
        patExclude{end+1} = patNames(i,:);
    end

    fprintf('Time spent converting: %.1f\n \n', timerval);
end
    patNames = cellstr(patNames);
    patNames = patNames(logical(flagDWIT2Aligned));
    numPats = numel(patNames);
    patNames = char(patNames);
end

%% Interpolation
% iteration through all patients for interpolation of DWI images
 ↪  to the T2 image size
if interpolate == 1
tic
% showing patient list for debugging
disp('Interpolating the patients:')
disp(patNames)

for i = 1:numPats
    clear T2 ivim DWI bin patientName
    fprintf('Interpolating patient number %d out of %d. \n', i,
     ↪  numPats);
    load(fullfile(outputPathRaw,char(patNames(i,:))));

    % Interpolating the DWI images
    intImg = InterpolateImage(T2, DWI);
    DWI.imgInt = intImg;

    % Interpolating the ivim images
    ivim = copyDWIInfoToIvim(DWI, ivim); % the ivim images are
     ↪  lacking crucial elements in their dicom header. These
     ↪  are added from the DWI images since
    % ther are derived from them.
    fields = fieldnames(ivim);
    for j = 1:length(fields)
```

```
        eval( ['intImg = InterpolateImage(T2, ivim.' fields{j} ');'])
        eval(['ivim.' fields{j},'.imgInt = intImg;'])
    end
    [~,~,~] = mkdir(outputPathInterpolated);
    save(fullfile(outputPathInterpolated,patNames(i,:)),'T2','ivim','DWI','bin','patientName');
    fprintf('Time spent interpolating: %.1f \n\n', toc);
end

end

%% FlipZDir
% Now redundant as it is integrated elsewhere

if flipZdirOfMask == 1
    disp('Flipping zdir of masks on required patients')
    for i=1:numPats
        if ismember(cellstr(patNames(i,:)), flipPats)
            disp(['Flipping patient ', patNames(i,:)])
            % converting the interpolated structs
            load(fullfile(outputPathInterpolated,char(patNames(i,:))),'bin');
            bin.an.mask = bin.an.mask(:,:,end:-1:1);
            save(fullfile(outputPathInterpolated,char(patNames(i,:))),'bin','-append');
        end
    end
end

%% cropping all the images so that they only contain the slices with cancer cells and also cropping the slices as described below
if crop == 1
    disp('Cropping')
% we want to crop the images so that we have a better balanced dataset.
% This should be done by taking the finding the maximum size of a rectangle
% covering the tumor in the x-y plane for all the images. Also the images
% without any tumour should be removed for the cropped images. We would
% like to save this information to separate structs to be able to use this
% data to efficiently build the dataset
```

```

[~,~,~] = mkdir(outputPathInterpolatedAndCropped);
[~,~,~] = mkdir(outputPathInterpolatedAndCroppedOnly);

for i = 1:numPats
    fprintf('Cropping images for patient %s\n', patNames(i,:))
    margin = 5; % percent of box dimension on each side
    load(fullfile(outputPathInterpolated,char(patNames(i,:))));
    [i1, i2, i3] = ind2sub(size(bin.an.mask), find(bin.an.mask)); % returns the indexes of the non-zero elements along each direction
    boxDim = [ max(i1) - min(i1), max(i2) - min(i2)];
    addedMargins = round((margin/100).*boxDim);
    dimStart = NaN(1,3); % storing the box starts in the first and second dimension
    dimEnd = NaN(1,3); % storing where the box ends in the first and second dimension
    if min(i1)-addedMargins(1) >= 1
        dimStart(1) = min(i1)-addedMargins(1);
    else
        dimStart(1) = 1;
    end

    if min(i2)-addedMargins(2) >=1
        dimStart(2) = min(i2)-addedMargins(2);
    else
        dimStart(2) = 1;
    end

    if max(i1) + addedMargins(1) <= size(bin.an.mask,1)
        dimEnd(1) = max(i1) + addedMargins(1);
    else
        dimEnd(1) = size(bin.an.mask);
    end

    if max(i2) + addedMargins(2) <= size(bin.an.mask,2)
        dimEnd(2) = max(i2) + addedMargins(2);
    else
        dimEnd(2) = size(bin.an.mask,2);
    end
    dimEnd(3) = max(i3);
    dimStart(3) = min(i3);

    T2.imgCrop = T2.img(dimStart(1):dimEnd(1),dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));
```

```
        DWI.imgCrop = DWI.imgInt(dimStart(1):dimEnd(1),dimStart(2):dimEnd(2),:,dimStart(3):dimEnd(3));
        bin.an.maskCrop = bin.an.mask(dimStart(1):dimEnd(1),dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));
        bin.shh.maskCrop = bin.shh.mask(dimStart(1):dimEnd(1),dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));

        ivim.ADC.imgCrop = ivim.ADC.imgInt(dimStart(1):dimEnd(1)...
            ,dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));
        ivim.D.imgCrop = ivim.D.imgInt(dimStart(1):dimEnd(1)...
            ,dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));
        ivim.Dstar.imgCrop = ivim.Dstar.imgInt(dimStart(1):dimEnd(1)...
            ,dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));
        ivim.f.imgCrop = ivim.f.imgInt(dimStart(1):dimEnd(1),...
            dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));

        bin.radiologistUsed = 'an';
        cropIndexStart = dimStart;
        cropIndexEnd = dimEnd;
        [~,~,~] = mkdir(fullfile(outputPathInterpolatedAndCropped,'an'));
        radiologist = 'an';
        save(fullfile(outputPathInterpolatedAndCropped,'an',patNames(i,:)),...
            'T2','ivim','DWI','bin','patientName', 'radiologist',...
            'cropIndexStart', 'cropIndexEnd');

        % We set the fields we need in the creation of the crop for the
        % radiologist shh to temporary variables so we can remove them from the
        % structs for easier saving.

        DWIimgIntTemp = DWI.imgInt;
        DWIimgTemp = DWI.img;
        T2imgTemp = T2.img;
        DWI = rmfield(DWI,'imgInt');
        DWI = rmfield(DWI, 'img');
        T2 = rmfield(T2, 'img');
        maskTempAn = bin.an.mask;
        bin.an = rmfield(bin.an, 'mask');
        maskTempShh = bin.shh.mask;
        bin.shh = rmfield(bin.shh, 'mask');

        ivimADCimgIntTemp = ivim.ADC.imgInt;
```

```matlab
        ivimDimgIntTemp = ivim.D.imgInt;
        ivimDstarimgIntTemp = ivim.Dstar.imgInt;
        ivimfimgIntTemp = ivim.f.imgInt;

        ivim.ADC = rmfield(ivim.ADC, 'imgInt');
        ivim.D = rmfield(ivim.D, 'imgInt');
        ivim.Dstar = rmfield(ivim.Dstar, 'imgInt');
        ivim.f = rmfield(ivim.f, 'imgInt');

        ivimADCimgTemp = ivim.ADC.img;
        ivimDimgTemp = ivim.D.img;
        ivimDstarimgTemp = ivim.Dstar.img;
        ivimfimgTemp = ivim.f.img;

        ivim.ADC = rmfield(ivim.ADC, 'img');
        ivim.D = rmfield(ivim.D, 'img');
        ivim.Dstar = rmfield(ivim.Dstar, 'img');
        ivim.f = rmfield(ivim.f, 'img');


        [~,~,~] = mkdir(outputPathInterpolatedAndCroppedOnly,'an');
        save(fullfile(outputPathInterpolatedAndCroppedOnly,'an',patN⌋
        ↪  ames(i,:)),...
            'T2','ivim','DWI','bin','patientName','radiologist',...
            'cropIndexStart','cropIndexEnd');
        clear radiologist;
        % We add the struct fields back so the code can access them.
        ↪  Here we could have rewritten the code to
        % use the temporary variables

        bin.shh.mask = maskTempShh;
        bin.an.mask = maskTempAn;
        DWI.imgInt = DWIimgIntTemp;
        DWI.img = DWIimgTemp;
        T2.img = T2imgTemp;

        ivim.ADC.imgInt = ivimADCimgIntTemp;
        ivim.D.imgInt = ivimDimgIntTemp;
        ivim.Dstar.imgInt = ivimDstarimgIntTemp;
        ivim.f.imgInt = ivimfimgIntTemp;

        ivim.ADC.img = ivimADCimgTemp;
        ivim.D.img = ivimDimgTemp;
        ivim.Dstar.img = ivimDstarimgTemp;
        ivim.f.img = ivimfimgTemp;

```

```matlab
    [i1, i2, i3] = ind2sub(size(bin.shh.mask), find(bin.shh.mask)); % returns the indexes of the non-zero elements along each direction
    boxDim = [ max(i1) - min(i1), max(i2) - min(i2)];
    addedMargins = round((margin/100).*boxDim);
    dimStart = NaN(1,3); % storing the box starts in the first and second dimension
    dimEnd = NaN(1,3); % storing where the box ends in the first and second dimension
    if min(i1)-addedMargins(1) >= 1
        dimStart(1) = min(i1)-addedMargins(1);
    else
        dimStart(1) = 1;
    end

    if min(i2)-addedMargins(2) >=1
        dimStart(2) = min(i2)-addedMargins(2);
    else
        dimStart(2) = 1;
    end

    if max(i1) + addedMargins(1) <= size(bin.shh.mask,1)
        dimEnd(1) = max(i1) + addedMargins(1);
    else
        dimEnd(1) = size(bin.shh.mask);
    end

    if max(i2) + addedMargins(2) <= size(bin.shh.mask,2)
        dimEnd(2) = max(i2) + addedMargins(2);
    else
        dimEnd(2) = size(bin.shh.mask,2);
    end
    dimEnd(3) = max(i3);
    dimStart(3) = min(i3);

    % saving the indexes at which we crop
    % want to crop T2, DWI, binary and IVIm images
    cropIndexStart = dimStart;
    cropIndexEnd = dimEnd;
    T2.imgCrop = T2.img(dimStart(1):dimEnd(1),dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));
    DWI.imgCrop = DWI.imgInt(dimStart(1):dimEnd(1),dimStart(2):dimEnd(2),:,dimStart(3):dimEnd(3));
    bin.an.maskCrop = bin.an.mask(dimStart(1):dimEnd(1),dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));
```

```
        bin.shh.maskCrop = bin.shh.mask(dimStart(1):dimEnd(1),dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));

        ivim.ADC.imgCrop = ivim.ADC.imgInt(dimStart(1):dimEnd(1)...
            ,dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));
        ivim.D.imgCrop = ivim.D.imgInt(dimStart(1):dimEnd(1)...
            ,dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));
        ivim.Dstar.imgCrop = ivim.Dstar.imgInt(dimStart(1):dimEnd(1)...
            ,dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));
        ivim.f.imgCrop = ivim.f.imgInt(dimStart(1):dimEnd(1),...
            dimStart(2):dimEnd(2),dimStart(3):dimEnd(3));

        bin.radiologistUsed = 'shh';
        [~,~,~] = mkdir(fullfile(outputPathInterpolatedAndCropped,'shh'));
        radiologist = 'shh';
        save(fullfile(outputPathInterpolatedAndCropped,'shh',patNames(i,:)),...
            'T2','ivim','DWI','bin','patientName','radiologist',...
            'cropIndexStart','cropIndexEnd');

        DWI = rmfield(DWI,'imgInt');
        DWI = rmfield(DWI, 'img');

        T2 = rmfield(T2, 'img');

        bin.an = rmfield(bin.an, 'mask');
        bin.shh = rmfield(bin.shh, 'mask');

        ivim.ADC = rmfield(ivim.ADC, 'imgInt');
        ivim.D = rmfield(ivim.D, 'imgInt');
        ivim.Dstar = rmfield(ivim.Dstar, 'imgInt');
        ivim.f = rmfield(ivim.f, 'imgInt');

        ivim.ADC = rmfield(ivim.ADC, 'img');
        ivim.D = rmfield(ivim.D, 'img');
        ivim.Dstar = rmfield(ivim.Dstar, 'img');
        ivim.f = rmfield(ivim.f, 'img');

        [~,~,~] = mkdir(outputPathInterpolatedAndCroppedOnly,'shh');
        save(fullfile(outputPathInterpolatedAndCroppedOnly,'shh',patNames(i,:)),...
            'T2','ivim','DWI','bin','patientName','radiologist',...
            'cropIndexStart','cropIndexEnd');
        clear radiologist
```

```matlab

end

end
%% WriteBValues
if writeBvalues == 1
    disp('Writing bvalues to the patients')
% write code here to extract the b values from the DWI images
 ↪  and reoprt
% them as a sorted list in the patient struct field DWI.bValues
for i=1:numPats
load(fullfile(outputPathRaw,char(patNames(i,:))));
DWI = writeBvaluesToStruct(DWI);
save(fullfile(outputPathRaw,patNames(i,:)),'T2','ivim','DWI','bi⌋
 ↪  n','patientName');

end
for i=1:numPats
load(fullfile(outputPathInterpolated,char(patNames(i,:))));
DWI = writeBvaluesToStruct(DWI);
save(fullfile(outputPathInterpolated,patNames(i,:)),'T2','ivim',⌋
 ↪  'DWI','bin','patientName');
end

% we do this for both of the radiologists
for j=1:length(radiologists)

for i=1:numPats
load(fullfile(outputPathInterpolatedAndCropped,radiologists{j},c⌋
 ↪  har(patNames(i,:))));
DWI = writeBvaluesToStruct(DWI);
save(fullfile(outputPathInterpolatedAndCropped,radiologists{j},p⌋
 ↪  atNames(i,:)),'T2','ivim','DWI','bin','patientName');
end

for i=1:numPats
load(fullfile(outputPathInterpolatedAndCroppedOnly,radiologists{⌋
 ↪  j},char(patNames(i,:))));
DWI = writeBvaluesToStruct(DWI);
save(fullfile(outputPathInterpolatedAndCroppedOnly,radiologists{⌋
 ↪  j},patNames(i,:)),'T2','ivim','DWI','bin','patientName');
end

end

end
```

```matlab
%% Write infoOnly files
if infoOnly == 1
    disp('Creating info only structs')
    % write a file only cotaining the info. This is for loading the file
    % quickly without having to load the image or the full header info. We
    % save one copy with the header and one without to make loading even
    % faster. For the moment only info for T2 and DWI is saved as the other
    % info is not used in the analysis

    [~,~,~] = mkdir(fullfile(outputPathInfoOnly,'header','an'));
    [~,~,~] = mkdir(fullfile(outputPathInfoOnly,'header','shh'));
    [~,~,~] = mkdir(fullfile(outputPathInfoOnly,'noHeader','an'));
    [~,~,~] = mkdir(fullfile(outputPathInfoOnly,'noHeader','shh'));

    for i=1:numPats
    % making infostruct for both radiologists for easy access to all info.
    % first for 'an'
    load(fullfile(outputPathInterpolatedAndCroppedOnly,'an',char(patNames(i,:))));
    fprintf('Making infostruct for patient %s\n', patNames(i,:))

    T2 = rmfield(T2,'imgCrop');
    DWI = rmfield(DWI,'imgCrop');
    bin.an = rmfield(bin.an, 'maskCrop');
    bin.shh = rmfield(bin.shh, 'maskCrop');
    save(fullfile(outputPathInfoOnly,'header','an',patNames(i,:)),'T2','DWI','patientName','radiologist','cropIndexStart','cropIndexEnd');

    T2 = rmfield(T2,'header');
    DWI = rmfield(DWI,'header');

    save(fullfile(outputPathInfoOnly,'noHeader','an',patNames(i,:)),'T2','DWI','patientName','radiologist','cropIndexStart','cropIndexEnd');

    % repeat for 'shh'
    load(fullfile(outputPathInterpolatedAndCroppedOnly,'shh',char(patNames(i,:))));
    T2 = rmfield(T2,'imgCrop');
    DWI = rmfield(DWI,'imgCrop');
    bin.an = rmfield(bin.an, 'maskCrop');
    bin.shh = rmfield(bin.shh, 'maskCrop');
    save(fullfile(outputPathInfoOnly,'header','shh',patNames(i,:)),'T2','DWI','patientName','radiologist','cropIndexStart','cropIndexEnd');

    T2 = rmfield(T2,'header');
    DWI = rmfield(DWI,'header');

    save(fullfile(outputPathInfoOnly,'noHeader','shh',patNames(i,:)),'T2','DWI','patientName','radiologist','cropIndexStart','cropIndexEnd');
    end
end

%% Remove slices outside the DWI fov
% for some slices the mask is outside the FOV of the DWI images and thus we
% suspect this can be confusing for the model. Thus we remove these slices.
if rmvSlicesOutsideDWIFOV == 1
    disp('Removing slices outside DWI FOV')
    for j=1:length(radiologists)
    [~,~,~] = mkdir(fullfile(outputPathInterpolatedAndCroppedOnlyInDWIFOV),radiologists{j});
    for i=1:numPats
        load(fullfile(outputPathInterpolatedAndCroppedOnly,radiologists{j},char(patNames(i,:))));
        flag = true(1,size(DWI.imgCrop,4));
        for zPos = 1:size(DWI.imgCrop,4)
            imSel=DWI.imgCrop(:,:,1,zPos);
            if sum(isnan(imSel(:))) == numel(imSel)
                flag(zPos)=0;
            end
        end

        % we need to update the crop index start and maybe also the crop
        % index end to account for the slices that are removed.

        sliceIndexes = sort(find(flag)); % The sort should be redundant
        cropIndexStart(3) = cropIndexStart(3) + sliceIndexes(1) - 1;
        cropIndexEnd(3) = cropIndexStart(3) +
        ↪  numel(sliceIndexes)-1;

        DWI.imgCrop = DWI.imgCrop(:,:,:,flag);
        T2.imgCrop = T2.imgCrop(:,:,flag);
        bin.an.maskCrop = bin.an.maskCrop(:,:,flag);
        bin.shh.maskCrop = bin.shh.maskCrop(:,:,flag);

        ivim.ADC.imgCrop = ivim.ADC.imgCrop(:,:,flag);
        ivim.D.imgCrop = ivim.D.imgCrop(:,:,flag);
        ivim.Dstar.imgCrop = ivim.Dstar.imgCrop(:,:,flag);
        ivim.f.imgCrop = ivim.f.imgCrop(:,:,flag);


        radiologist = radiologists{j};
        save(fullfile(outputPathInterpolatedAndCroppedOnlyInDWIF⌋
        ↪  OV,radiologists{j},char(patNames(i,:))),...
            'T2','bin','DWI','ivim','patientName','radiologist',⌋
            ↪  'cropIndexStart','cropIndexEnd');
    end
    end
end


%%
% When code is finished sing Hallelujah and add patients to the
↪  list of
% patients done
if updatePatsDone
load(filepath);
patDone = unique([cellstr(lower(patNames)); lower(patDone)]);
save(filepath, 'patDone');
end

if updatePatsExclude
temp = load(patsExcludePath);
patExclude = unique([lower(patExclude), lower(temp.patExclude)]);
save(patsExcludePath, 'patExclude');
end

if sing
load handel.mat
sound(y(1500:18000),Fs)
end
%%

```

```
function ivimStruct = copyDWIInfoToIvim(DWIstruct, ivimStruct)
[PatientsPosition, pixelSize, SliceThickness, ...
    SpacingBetweenSlices, dimSlice, PatientsOrientation] =
     ↪  extractHeaderInfo(DWIstruct.header);
fields = fieldnames(ivimStruct);
for i = 1:length(fields)
    eval(['ivimStruct.',fields{i},'.PatientsPosition =
     ↪  PatientsPosition;'])
    eval(['ivimStruct.',fields{i},'.pixelSize = pixelSize;'])
    eval(['ivimStruct.',fields{i},'.SliceThickness =
     ↪  SliceThickness;'])
    eval(['ivimStruct.',fields{i},'.SpacingBetweenSlices =
     ↪  SpacingBetweenSlices;'])
    eval(['ivimStruct.',fields{i},'.dimSlice = dimSlice;'])
    eval(['ivimStruct.',fields{i},'.PatientsOrientation =
     ↪  PatientsOrientation;'])
end
end

function DWIstruct = writeBvaluesToStruct(DWIstruct)
allBvals = NaN(1,length(DWIstruct.header));
    for j=1:length(DWIstruct.header)
    allBvals(j) = DWIstruct.header{j}.DiffusionBValue;
    bValues = unique(allBvals);
    DWIstruct.bValues = bValues;
    end
end
```

```
function generateAndSaveStructFields(inputPath, outputPath,
 ↪  patientName)


% Script that takes in 2D dicom data and converts it to 3D mat
 ↪  data. The
% function mimics the folder structure already present in the
 ↪  input data.
% The slices are sorted according to increasing z-pos in the 3D
 ↪  matrices.
% The data for the different patients are assumend to be in a
 ↪  folder on the
% form "oxytarget_# pre".  We here also want to be able to use
 ↪  the code
% written by Franziska Knuth for interpolating the images.

%% Find the folder structure of the import folder

% We create a folder to where we want to put the converted
 ↪  patient data
[~,~,~] = mkdir(outputPath);
dirs = getDirectoryNames(inputPath);
%% Make folders and convert DICOM to mat

for i=1:length(dirs)
    switch dirs(i)
        case 'T2'
            disp('Converting T2 images')
            T2InputPath = [inputPath, '\', 'T2'];
            infoTable = buildAndSortInfoTable('T2',T2InputPath);
            T2 = makeStruct(infoTable.infoStruct, T2InputPath,
             ↪  infoTable.fileNames);
            % I want to put as much of this into a function that
             ↪  can be
            % generalized to work on IWIM and DWI data also

        case {'ivim' , 'IVIM'}
            disp('Converting IVIM images');
            ivimInputPath = [inputPath, '\', char(dirs(i))];
            subdirs = getDirectoryNames(ivimInputPath);
            for j=1:length(subdirs)
                inPath = fullfile(ivimInputPath,
                 ↪  char(subdirs(j)));
                infoTable = buildAndSortInfoTable('IVIM',inPath);
                eval(['ivim.' char(subdirs(j)),
                ↪  '=makeStructIvim(infoTable.infoStruct,
                ↪  inPath, infoTable.fileNames);'])


                % create info table like for T2 consider moving
                ↪  into
                % separate function.

                % create function: explore subdirs
            end
            clear header imArray numFiles ivimInputPath
            ↪  ivimOutpurPath
            clear fileNames z_pos
        case 'DWI'
            disp('Converting DWI images')
            DWIInputPath = fullfile(inputPath, char(dirs(i)));
            infoTable = buildAndSortInfoTable('DWI',
            ↪  DWIInputPath);
            DWI = makeStruct(infoTable.infoStruct, DWIInputPath,
            ↪  infoTable.fileNames);


        case 'binary'
            disp('Converting nifti images');
            binaryInputPath = [inputPath, '\', char(dirs(i))];
            subdirs = getDirectoryNames(binaryInputPath);
            for j=1:length(subdirs)
                inPath =  fullfile(binaryInputPath,
                ↪  char(subdirs(j)));
                fileName = ls([inPath,'\','*.nii']);
                info = niftiinfo(fullfile(inPath,fileName));
                maskArray = rot90(fliplr(niftiread(fullfile(inPa⌋
                ↪  th,fileName)))); % The masks did not seem to
                ↪  be making sense at all. After looking at
                ↪  some images and masks it seems as
                % if they were mirrored and flipped 90 degrees.
                maskArray = maskArray./max(maskArray(:)); %
                ↪  Normalize mask array to contain only ones
                ↪  and zeros
                maskArray = double(maskArray); % convert mask
                ↪  array to same datatype as the imArray needed
                ↪  for mdl to run properly

                % check if the zaxis need to be inverted
                [~, ~, z1] =
                 ↪  transformPointsForward(info.Transform,1,1,1);
                [~, ~, z2] =
                 ↪  transformPointsForward(info.Transform,
                 ↪  1,1,100);
                needsInversion = (z2-z1) < 0;
                if needsInversion
                    maskArray = maskArray(:,:,end:-1:1);
                end



                eval(['bin.',char(subdirs(j,:)),' =
                 ↪  makeStructBin(maskArray,info);'])
            end
%         otherwise
%             disp('Aint nothing to do here')
    end

end




save(fullfile(outputPath,patientName),'T2','ivim','DWI','bin','p⌋
 ↪  atientName');
end

function directoryNames = getDirectoryNames(inputPath)
    dirs = struct2table(dir(inputPath));
    dirs.name = categorical(dirs.name);
    dirs(dirs.name == '..' | dirs.name == '.', :) = []; % remove
     ↪  dot and double dots from dir output as they are not
     ↪  folder names.
    directoryNames = dirs.name;
end

function infoTable = buildAndSortInfoTable(method,
 ↪  MethodInputPath)
 fileNames = ls([MethodInputPath,'\','*.dcm']); % Extract file
 ↪  names. Output a string array
 infoTable = table(fileNames,'VariableNames', {'fileNames'});
 numFiles = height(infoTable);
 z_pos = NaN(numFiles,1); % we record some common variables we
  ↪  want to sort by for all methods
 header = cell(numFiles,1); % for storing the dcm headers

 %% switch on method to also include and sort by b values in DWI

 switch method
     case 'DWI'
         bValues = NaN(numFiles,1);
         for j=1:numFiles
                header{j} = dicominfo([MethodInputPath,'\',infoTable.fileNames(j,:)]);
                z_pos(j) = header{j}.ImagePositionPatient(3);
                bValues(j) = header{j}.DiffusionBValue;
         end
         infoTable = addvars(infoTable, z_pos, bValues, header, 'NewVariableNames',{'z_pos','bValues','infoStruct'});
         infoTable = sortrows(infoTable, {'z_pos','bValues'},{'ascend','ascend'});
     case 'IVIM'
         % IVIM files do for some reason not include ImagePositionPatient
         % in the struct.
         disp(['NB: the IVIM images are not sorted according to z-position yet as this', ...
             'is missing in the dicomheader'])
         for j=1:numFiles
            header{j} = dicominfo([MethodInputPath,'\',infoTable.fileNames(j,:)]);

         end
         infoTable = addvars(infoTable, header, 'NewVariableNames' ,'infoStruct');

     otherwise
         for j=1:numFiles
                header{j} = dicominfo([MethodInputPath,'\',infoTable.fileNames(j,:)]);
                z_pos(j) = header{j}.ImagePositionPatient(3);

         end
         infoTable = addvars(infoTable, z_pos, header, 'NewVariableNames',{'z_pos','infoStruct'});
         infoTable = sortrows(infoTable, {'z_pos'},{'ascend'});
 end
end

```

```matlab
function methodStruct = makeStruct(dcmHeaders, methodInputPath, fileNames)
    imArray = dicom2mat(methodInputPath, fileNames);
    imArray = rescaleMR(dcmHeaders, imArray);
    image = imArray3Dto4D(dcmHeaders, imArray);
    [PatientsPosition, pixelSize, SliceThickness, ...
    SpacingBetweenSlices, dimSlice, PatientsOrientation] = extractHeaderInfo(dcmHeaders);
    methodStruct = struct('header', {dcmHeaders}, 'img', image, ...
        'PatientsPosition',PatientsPosition,'pixelSize',pixelSize, ...
        'SliceThickness',SliceThickness, 'SpacingBetweenSlices', SpacingBetweenSlices, ...
        'dimSlice', dimSlice, 'PatientsOrientation', PatientsOrientation);
end

function methodStruct = makeStructIvim(dcmHeaders, methodInputPath, fileNames)
    % A special case of the function is made due to the fact that the ivim
    % images lack the header information needed by the image3Dto4D function
    imArray = dicom2mat(methodInputPath, fileNames);
    imArray = rescaleMR(dcmHeaders, imArray);
    szIm = size(imArray);
    image = reshape(imArray, szIm(1), szIm(2), 1, szIm(3));
    methodStruct = struct('header', {dcmHeaders}, 'img', image);
end

function binStruct = makeStructBin(maskArray,info)
    binStruct = struct('info', info,'mask', maskArray);
end

function rescaledImArray = rescaleMR(dcmHeaders, imArray)
% Rescales the image using the rescale intercept and rescale slope found in
% the dicom header for each slice.
rescaledImArray=NaN(size(imArray));
for i=1:numel(dcmHeaders)
    intercept = dcmHeaders{i}.RescaleIntercept;
    slope = dcmHeaders{i}.RescaleSlope;
    for xind=1:size(imArray,1)
        for yind=1:size(imArray,2)
```

```
                rescaledImArray(xind,yind,i)=imArray(xind,yind,i)*sl⌟
                ↪  ope +
                ↪  intercept;
            end
        end
    end
    end
```

```matlab
function imArray = dicom2mat(inputPath, fileNames)
% dicom2mat converts the dicom files to mat files. Input and
↪  output paths
% are char vectors and the filenames are in a char array with a
↪  fileName in
% each row

numFiles = size(fileNames,1);
tempIm = dicomread([inputPath,'\' ,fileNames(1,:)]);
imArray = NaN(size(tempIm,1), size(tempIm,2), numFiles);
imArray(:,:,1) = tempIm;
if numFiles > 1
    for i=2:numFiles
        imArray(:,:,i) = dicomread([inputPath,'\'
         ↪  ,fileNames(i,:)]);
    end
end

end
```

```
function [PatientsPosition, pixelSize, SliceThickness, ...
    SpacingBetweenSlices, dimSlice, PatientsOrientation] =
    ↪  extractHeaderInfo(dcmHeaders)
% extractHeaderInfo extracts the information needed for
↪  interpolation of
% the images and saves it to variable corresponding to the
↪  struct fields
% needed. Saving the variables from the workspace and loading
↪  them with the
% s = load(filename) syntax yields a struct with the correct
↪  fields. The
% input is a cell array of headers that are sorted according to
↪  a choosen
% criterions

pixelSize = dcmHeaders{1}.PixelSpacing(1);
SliceThickness = dcmHeaders{1}.SliceThickness;
SpacingBetweenSlices = dcmHeaders{1}.SpacingBetweenSlices;
dimSlice = dcmHeaders{1}.Width;
PatientsOrientation = dcmHeaders{1}.ImageOrientationPatient;

% We make a list of all the unique values of ImagePositionPatient
PatientsPosition = dcmHeaders{1}.ImagePositionPatient';

for i=2:length(dcmHeaders)
    temp = sum(PatientsPosition(end, :) ==
    ↪  dcmHeaders{i}.ImagePositionPatient');
    if sum(temp(:)) < 3
        PatientsPosition = cat(1,PatientsPosition,
        ↪  dcmHeaders{i}.ImagePositionPatient');
    end
end
end
```

```matlab
function intImg = InterpolateImage(aimImg, startImg)
% Interpolates image in startImg to the coordinates of aimImg
% aimImg and startImg need to be structures as produced by
% LoadSortAndSave.m. (see documentation in getVoxelCoordinates)
%
%
% Version from 27.08.2018,
% by Franziska Knuth


%% Comment in, to run the script separated
% % Load data
% aimImg   = load('C:\ALL_DATA\MatlabFormat\24_PRE_T2.mat');
% startImg = load('C:\ALL_DATA\MatlabFormat\24_PRE_DWI.mat');

%% get Image coordinates, Scanner coordinates

[Xq,Yq,Zq] = getVoxelCoordinates(aimImg);
posQ = [Xq(:), Yq(:), Zq(:)];

[X, Y, Z]  = getVoxelCoordinates(startImg);
pos = [X(:), Y(:), Z(:)];


% Preapare empty image to save result
szImg = size(aimImg.img);
szDim3 = size(startImg.img,3);
intImg = zeros(szImg(1), szImg(2), szDim3 , szImg(4));

% loop over all images in 3d dimension (can be 1 or different
 ↪  b-values, timepoints,...)
for idx = 1:szDim3
    % get Volume as vector
    V = squeeze(startImg.img(:,:,idx,:));

    % interpolate the startImage
    % use linear interpolation between sample points
    % set outsideValues to NaN
    F = scatteredInterpolant(pos,V(:), 'linear','none');

    % get values of the image for the coordinates of aimImg
    Vq = F(posQ);

    % rearange from vector to matrix
    intImg(:,:,idx,:) = reshape(Vq, szImg);
end
```

```matlab
function [x,y,z] = getVoxelCoordinates(d)
%%
% d contains  (example)
%         PatientsPosition: [26×3 double]   specifies the x, y, and z
%                                           coordinates of the upper left
%                                           hand corner of each image slice
%                pixelSize: 0.3516          dimension in mm in xy plane
%           SliceThickness: 2.5000          dimension of voxel in z
%                                           direction
%     SpacingBetweenSlices: 2.7500
%                 dimSlice: 512             xy-plane = 512*512 pixel
%      PatientsOrientation: [1x6]           direction cosines
%                                           first 3: row value for x,y,z
%                                           last 3: collumn values for
%                                           x,y,z
%  See dicom documenation for details about the calcualation
%%

x = zeros(d.dimSlice, d.dimSlice, size(d.PatientsPosition,1));
y = x;
z = x;

X = d.PatientsOrientation(1:3)';    % row values
Y = d.PatientsOrientation(4:6)';    % collumn values
dimPix = d.pixelSize;               % in mm

% idx  = double(1:d.dimSlice);
idx = double(0:d.dimSlice-1);
[cMat,rMat] = meshgrid(idx,idx);

for k_idx = 1:size(d.PatientsPosition,1)
    S = d.PatientsPosition(k_idx,:);
    x(:,:,k_idx) = (X(1)*rMat+Y(1)*cMat)*dimPix + S(1);
    y(:,:,k_idx) = (X(2)*rMat+Y(2)*cMat)*dimPix + S(2);
    z(:,:,k_idx) = (X(3)*rMat+Y(3)*cMat)*dimPix + S(3);
end
```

A.3.2 DWI as heatmaps on T2w images

```
function overlayDWIOnT2AsHeatmap(varargin)
%overlayDWIOnT2AsHeatmap select a patient where you want the DWI
↪  image
% overlaid as a heatmap on the T2w image.

if isempty(varargin)
[file, path] =
↪  uigetfile(fullfile('PatientStructs\Interpolated','*.mat'));
else
    path = 'PatientStructs\Interpolated';
    file = varargin{1};
end

load(fullfile(path,file));
[~,~,~] = mkdir('DWIHeatmapOverT2');


% we put each succesive image into
numSlices = size(T2.img, 4);
sz = size(T2.img);
images = cell(1, numSlices);
imagesMat = NaN(sz(1), numSlices*sz(2),3);
im = [];

% splitting the channels
R = [];
G = [];
B = [];
fprintf('Slice: ')
for i = 1:numSlices
fprintf('%d ', i)
images{i} = overlayHeatmapOnImage(squeeze(T2.img(:,:,1,i)),...
    DWI.imgInt(:,:,1,i));

% imagesMat(1:sz(1),(i-1)*sz(2)+1:(i)*sz(2),:) = images{i};
% we do it the other way around

% fusing the channels

R = cat(2, R, images{i}(:,:,1));
G = cat(2, G, images{i}(:,:,2));
B = cat(2, B, images{i}(:,:,3));

end
```

```

fprintf('\n')

im = cat(3, R, G, B);

saveName = strsplit(file,'.mat'); %returns a cell array with
↪  wanted string as first field
saveName = saveName{1};

imwrite(im, fullfile('DWIHeatmapOverT2',[saveName '.tiff'] ))

end
```

```
function heatmapMat = overlayHeatmapOnImage(image, OverlayImage)


alpha = (~isnan(OverlayImage))*0.4;
% alpha(isnan(alpha)) = 0;
ax1 = axes('Position', [0 0 1 1]); %('Position',[1 1 512 512],'Units', 'pixels');
imshow(image, 'Colormap', gray, 'Parent', ax1);
caxis auto
hold on
ax2 = axes('Position', [0 0 1 1]); %('Position',[1 1 512 512],'Units', 'pixels');
OverlayImage = imshow( OverlayImage, 'Colormap', jet, 'Parent', ax2 );
% Set the color limits to be relative to the data values
caxis auto
% Set the AlphaData to be the transparency matrix created earlier
set( OverlayImage, 'AlphaData', alpha );

f=getframe;

heatmapMat = f.cdata;

end
```

A.3.3 Compare images to check alignment

```matlab
function compareImagesToCheckAlignment(varargin)
% CompateImagesToCheckAlignmet will print interpolated images of selected
% patients and methods for inspections. More specifically it will
% concatenate the interpolated image arrays so that the images are stacked
% directly on top of eachother. The varargin can either be empty in which
% case the code will ask you to select a patient an which methods to
% display. In case of one argument specified the code will assume that this
% is the patient name. In case of 2 inputs specified the code will assume
% the first inout is the patient name and the second is a cell array of the
% methods to be used.

savefolder = fullfile('ImagesForAlignmentComparison');
[~,~,~] = mkdir(savefolder);

if isempty(varargin)
[file, path] = uigetfile(fullfile('PatientStructs\Interpolated','*.mat'));
else
    path = 'PatientStructs\Interpolated';
    file = varargin{1};
end

load(fullfile(path,file));


methodsToCompare = {};

if length(varargin) < 2
avaliableMethods = {'T2', 'DWI 0','DWI 500','DWI 1000','ADC','D','Dstar','f'};
[indx, tf] = listdlg('PromptString','Select methods', ...
    'ListString', avaliableMethods);
end


if length(varargin) < 2
imCells = cell(1,length(indx));
```

```
for i=1:length(indx)
    method{i} = avaliableMethods{indx(i)};
    switch method{i}
        case 'T2'
            imCells{i} = squeeze(T2.img);
        case 'DWI 0'
            imCells{i} = squeeze(DWI.imgInt(:,:,find(DWI.bValues
             ↪  == 0), :));
        case 'DWI 500'
            imCells{i} = squeeze(DWI.imgInt(:,:,find(DWI.bValues
             ↪  == 500), :));
        case 'DWI 1000'
            imCells{i} = squeeze(DWI.imgInt(:,:,find(DWI.bValues
             ↪  == 1000), :));
        case 'ADC'
            imCells{i} = squeeze(ivim.ADC.imgInt);
        case 'D'
            imCells{i} = squeeze(ivim.D.imgInt);
        case 'Dstart'
            imCells{i} = squeeze(ivim.Dstar.imgInt);
        case 'f'
            imCells{i} = squeeze(ivim.f.imgInt);
    end
end
else
imCells = cell(1,length(varargin{2}));
for i=1:length(varargin{2})
    method{i} = varargin{2}{i};
    switch method{i}
        case 'T2'
            imCells{i} = squeeze(T2.img);
        case 'DWI 0'
            imCells{i} = squeeze(DWI.imgInt(:,:,find(DWI.bValues
             ↪  == 0), :));
        case 'DWI 500'
            imCells{i} = squeeze(DWI.imgInt(:,:,find(DWI.bValues
             ↪  == 500), :));
        case 'DWI 1000'
            imCells{i} = squeeze(DWI.imgInt(:,:,find(DWI.bValues
             ↪  == 1000), :));
        case 'ADC'
            imCells{i} = squeeze(ivim.ADC.imgInt);
        case 'D'
            imCells{i} = squeeze(ivim.D.imgInt);
        case 'Dstart'
            imCells{i} = squeeze(ivim.Dstar.imgInt);
        case 'f'
            imCells{i} = squeeze(ivim.f.imgInt);
    end
end

end

n = length(imCells);
figure('Visible','off');

for i=1:length(imCells)
    subplot(n,1,i)
    imds{i} = montage((imCells{i}-min(imCells{i}(:)))./(max((imCells{i}(:))-min(imCells{i}(:)))),'Size',[1 size(imCells{i},3)]);
end

% alternative solution
im = imds{1}.CData;
for i=1:(length(imds)-1)
    im = cat(1, im,imds{i+1}.CData);
end

% making the suggested name
saveName = strsplit(file,'.mat'); %returns a cell array with wanted string as first field
saveName = saveName{1};
methodString = [saveName,':'];
tempString = [saveName,':']; % copy for comparrison later
for i=1:length(method)
    saveName = [saveName, '_', method{i}];
    if isequal(methodString, tempString)
        methodString =[tempString ' ' method{i}];
    else
        methodString = [methodString,', ' ,method{i}];
    end
end
saveName = [saveName , '.tiff'];

figure;
imshow(im);
title(methodString,'Interpreter','none');
print(fullfile(savefolder,saveName),'-dtiff', '-r1600')
end
```

A.3.4 Building the dataset

```matlab
function [featureArray, FileNames, grpsPasInd]  = rebuildFeatureArrays(test, varargin)

    % rebuildFeatureArrays takes 3D image stacks saved as mat file
    % and extracts the image intesities of the delineated tumour volumes
    % for each image sequence as vectors. The vectors for each image
    % sequence is concatenated into an array.
    % Input:
    % test - is a logical flag indicating whether to build the
    % dataset will all patients or a subset of patient. If 1
    % the number of patients must be passed as the varargin.
    % Output:
    % featureArray -  an array containing the image intesities
    % of each image sequence extracted from the tumour volume.
    % FileNames - the file names of the files, in this case
    % patients included in the dataset.
    % grpsPasInd - an array containing the start and stop index
    % of each patient.

    dataPath = 'data\InterpolatedAndCroppedInDWIFOVOnly\an';
    FileNames = cellstr(ls(fullfile(dataPath,'*.mat')));

    if nargin > 2 || (nargin == 2 && ~isnumeric(varargin{1}))
        error('To many input arguments, or the wrong kind of input arguments')
    end



    if ~isempty(varargin)
        if varargin{1} < numel(FileNames)
            numPats = varargin{1};
        else
            numPats = numel(FileNames);
            fprintf("The number of wanted patients exceeds the number of\n")
            fprintf("avaliable patients. NumPats will be set to its maximum\n")
            fprintf("allowed value\n")
        end
    end

    if test
        FileNames = FileNames(1:numPats);
    end

    % exclude patients due to alignment issues of DWI and T2w
    ↪  images
    excludePatients = {'Oxytarget_133 PRE.mat', 'Oxytarget_148
    ↪  PRE.mat',...
        'Oxytarget_171 PRE.mat', 'Oxytarget_191
        ↪  PRE.mat','Oxytarget_146 PRE.mat'};

    [~, idx] = ismember(excludePatients, FileNames);
    if ~(sum(idx(:))==0) % if so there is no files to exclude
        FileNames(idx) = [];
    end

    numPats = numel(FileNames);
    imSize = nan(numPats,3);


    masks = cell(numPats,1);
    T2 = cell(numPats,1);
    DWI = cell(numPats,6);
    ADC = cell(numPats,1);
    D = cell(numPats,1);
    Dstar = cell(numPats,1);
    f = cell(numPats,1);
    grpsPasInd = nan(numPats,1);
    patData = cell(numPats,1);

    tic
    parfor i=1:numPats
        disp(i)
        patData{i} = load(fullfile(dataPath, FileNames{i}));
        masks{i} = logical(patData{i}.bin.an.maskCrop);
        T2{i} = patData{i}.T2.imgCrop(masks{i});
        grpsPasInd(i) = sum(masks{i}(:));
        for j=1:6
            DWI{i,j} = patData{i}.DWI.imgCrop(:,:,j,:);
            DWI{i,j} = DWI{i,j}(masks{i});
        end
        ADC{i} = patData{i}.ivim.ADC.imgCrop(masks{i});
        D{i} = patData{i}.ivim.D.imgCrop(masks{i});
        Dstar{i} = patData{i}.ivim.Dstar.imgCrop(masks{i});
        f{i} = patData{i}.ivim.f.imgCrop(masks{i});
        imSize(i,:) = size(patData{i}.T2.imgCrop);
        patData{i} = []; % Reducing memory used without
         ↪  interfering
        %with independent of loops.
    end
    toc

    % converting each imaging modality into a feature
    featureArray = cell2mat([T2, DWI, ADC, D, Dstar,f]);
    size(featureArray)
    grpsPasInd = cumsum(grpsPasInd); % To get the final index of
     ↪  all patient data
    grpsPasInd = [([0; grpsPasInd(1:end-1)]+1) grpsPasInd ]; %
     ↪  to get the start
    % index of given patient data now a 2*numpats vector
    clear T2 DWI ADC D Dstar f patData % For more memory saving
end
```

A.3.5 Code for mapping components back onto tumour masks

This code maps the cluster or component labels back onto the binarized tumour volumes and unpack the tumour slices as an image montage. An example image using 3 groups is shown in figure 29. The code is given below in the matlab function `grpsMontage`.

```matlab
function grpsMontage(binImage, grpsVec, imSize, varargin)
%grpsMontage prints a montage of tumour volumes with groups in
↪  different
%colors
% Can use bin image as input both as an image stack and as a
↪  vector.
% grpsVec is a vector containing the cancer groups. Saves image
↪  as an eps
% file and preview as a low res png to folder structure located
↪  in group
% images

%% pathNames for saving
epsPath = fullfile('cancerGroupImages', 'eps');
previewPath = fullfile('cancerGroupImages', 'png');

%% Colormaps from colorbrewer can be extended if more colours
↪  are needed

cmap = [255,255,255;
         102,194,165;
         252,141,98;
         141,160,203;
         231,138,195;
         166,216,84;
         255,217,47;
         229,196,148;
         179,179,179]./255;

%% Creating the images
grpsImage = zeros(imSize);
grpsImage(binImage) = grpsVec;

% montages causes some strange interpolation lines in the
↪  qualitative image
% data. using the numsubplots function to get indexes to create
↪  a more or
% less quadratical arrya using the slices

[n,~] = numSubplots(imSize(3));
```

```matlab
imCells = cell(n(1),n(2));
sliceIndx = 0:imSize(1)*imSize(2):prod(imSize);

% Fill the imCells with a nanMatrix of appropriate size
for i=1:numel(imCells)
    imCells{i} = nan(imSize(1), imSize(2));
end


for i=1:imSize(3)
   imCells{i} = reshape(grpsImage(sliceIndx(i)+1:sliceIndx(i+1)),imSize(1),imSize(2));
end

image = cell2mat(imCells);
figure('Visible','off');
% figure()
imshow(image);
colormap(cmap(1:numel(unique(grpsVec))+1,:));
caxis auto
uniqueLabelsVec = unique(grpsVec);
numGrps = max(uniqueLabelsVec);
% have to add one because we have unique(grpsVec) subregions in the
% tumour and thus +1 when we include area that is not tumour


colorbar('Ticks', numGrps/(2*(numGrps+1)):numGrps/(numGrps+1):numGrps,'TickLabels',[0; unique(grpsVec)])

if ~isempty(varargin) && mod(length(varargin),2) == 0
    for i = 1:2:length(varargin)
        switch varargin{i}
            case 'FileName'
                fileName = varargin{i+1};
            case 'Subfolder'
                if ~ischar(varargin{i+1})
                    error('Subolder name must be a char')
                else
                subfolder = varargin{i+1};
                epsPath = fullfile(epsPath, subfolder);
                previewPath = fullfile(previewPath, subfolder);
                end
        end
    end
```

```
else
    fileName = inputdlg('Enter file name');
    if isempty(fileName)
        error('Enter a valid file name')
    end
end

[~,~,~] = mkdir(epsPath);
[~,~,~] = mkdir(previewPath);



%% Saving the figure
% making low res png for preview and eps for report
set(gcf, 'color', 'W')
print(fullfile(epsPath, [fileName '.eps']),'-depsc')
print(fullfile(previewPath, [fileName '.png']),'-dpng','-r100')

end
```

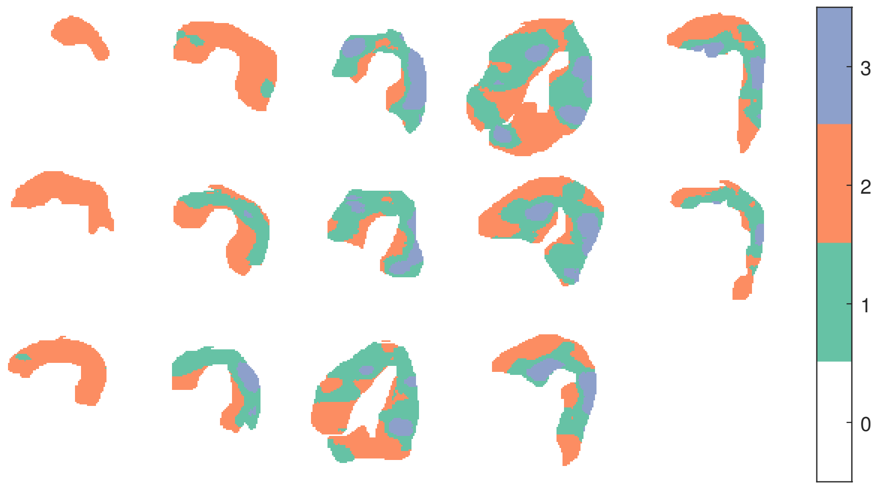


Figure 29: Example image showing the components mapped back onto the tumour slices as different colored pixels.

A.4 Python code

A general comment: The notion of without interactions refers to the fact that there were originally two datasets built. One where polynomial interaction features up to the third order were built using DWI with select b-values as described in *Materials & Methods* section 3.3. The intention was to explore whether this would give greater cluster separability and thus validity. However, the resulting dataset did not seem to have more easily distinguishable clusters and it was thus dropped from further analysis. However, this is the reason that some of the code in the appendix still will refers to ”*dataset without interactions*”.

A.4.1 Custom plots

```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import os
from itertools import combinations
from scipy.io import loadmat

from sklearn.preprocessing import PolynomialFeatures,
↪  StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KernelDensity

def kde_scatter_histplot(dataFrame,**kwargs):
    # plots the kde plots, scatter plots and histograms of the
    ↪  first 5 components of a pandas dataFrame.
    # Labeling assumes that the first five columns are principal
    ↪  components

    # default values
    bw = 0.2
    s = 5
    n_components = 5
    save = False

    for key, value in kwargs.items():
        if key == 'size':
            s = value
        elif key == 'bw':
            # Bandwidth is either a number or a dataFrame
            if type(value) == pd.core.frame.DataFrame:
                # normal functionality
                if not(value.shape[0] == n_components and value.shape[1] == n_components):
                    raise ValueError(f"DataFrame of incompatible length. Expected shape was ({n_components},{n_components})")
                else:
                    bw = value
        elif key == 'n_components':
            n_components = value
        elif key == 'savepath':
            savepath = value
            save = True
            # checks if folder where you want to save the figure exists
            if not(os.path.exists(os.path.dirname(savepath))):
                # The specified folder does not exist
                raise ValueError(f"The specified folder path {os.path.dirname(savepath)} does not exist.")
        else:
            return(print('unrecognized key'))


    # The following code expects that the bw is in the form of a pandas dataframe with indexes and colums being named
    # PC1,...,PC{n_components} according to the principal component number. Thus if bw is a number, we convert it to a
    # dataFrame that is bw in all positions.
    if not(type(bw)==pd.core.frame.DataFrame):
        PClist = [f"PC{i+1}" for i in range(n_components)]
        bw = np.ones((n_components,n_components))*bw
        bw = pd.DataFrame(bw, columns=PClist, index = PClist)

    fig, grid = plt.subplots(n_components, n_components, figsize=(8, 8))

    for i, row in enumerate(grid):
        for j, ax in enumerate(row):
            if i > j:
                sns.kdeplot(dataFrame.iloc[:,j], dataFrame.iloc[:,i], ax=ax, shade=True, bw=bw.loc[f'PC{j+1}',f'PC{i+1}'],vertical = True)
            elif i == j:
                sns.distplot(dataFrame.iloc[:,j],kde=False,ax=ax)
            else:
                sns.scatterplot(dataFrame.iloc[:,j], dataFrame.iloc[:,i], ax=ax, s=s)
            ax.set_xticks([])
            ax.set_yticks([])
            ax.set_ylabel('')
            ax.set_xlabel('')
            if j == 0:
                ax.set_ylabel(f'PC{i+1}') # python is zero indexed
            if i == n_components-1:
                ax.set_xlabel(f'PC{j+1}') # python is zero indexed
    if save:
        plt.savefig(savepath)
    plt.show()


def find_bandwidths_for_kde_matrix(dataFrame, bandwidths, numComponents):
    # we want to find the best bandwidths for the KDE matrix by cross validation.
    # This is quite computationally expensive so results should be saved when ran.
    # Anything more than ~20000 samples is infeasible to run.
    # The best bandwidth will be found for PC component £i£ plotted against
    # PC component £j£ for £i < j£ in this way only due to the fact that duplicates
    # (PC1, PC2 and PC2, PC2) are uninteresting as well as PCs with themselves.
    # Inputs:
    # dataFrame - pandas datafram with the Principal Components of the a dataSet
    # bandwiths - list of bandwidths. The code will find the best bandwidth in this list.
    # numComponents - number of components. Pairs of PC_1,...,PC_numComponents

    # First figure out if the best bandwidths is already found for the given combination
    # of components and data samples.

    # Create a file name that also serves as an ID for the experiment
    fileName = f"BestBandwidthsN{dataFrame.shape[0]}Grid{numComp
    ↪  onents}by{numComponents}"
    if os.path.exists(fileName):
        print("Experiment already run. Loading previous
        ↪  results.")
        return pd.read_pickle(fileName)
    else:
        # Run the experiment if not already tested
        PClist = [f"PC{i+1}" for i in range(numComponents)]
        bestBandwidths = pd.DataFrame(np.zeros((numComponents,nu
        ↪  mComponents)),columns=PClist,
        ↪  index=PClist)

        for i in range(numComponents):
            for j in range(numComponents):
                if i<j:
                    print(f"PC{i+1} PC{j+1}")
                    grid = GridSearchCV(KernelDensity(kernel='ga
                    ↪  ussian'),
                        {'bandwidth': bandwidths},
                        cv=10,n_jobs = -2, verbose = 1)
                    grid.fit(dataFrame.iloc[:,[i,j]])
                    bestBandwidths.loc[f"PC{i+1}",f"PC{j+1}"] =
                    ↪  grid.best_params_['bandwidth']
                    bestBandwidths.loc[f"PC{j+1}",f"PC{i+1}"] =
                    ↪  grid.best_params_['bandwidth']
        bestBandwidths.to_pickle(fileName) # storing the results
        return(bestBandwidths)

def kde_n_components(dataFrame, savepath,**kwargs):
    # Standard values
    bw = 0.25

    if not os.path.exists(savepath):
        print('Please sepcify a valid savepath')
        return
    for key, value in kwargs.items():
        if key == 'n_components':
            n_components = value
            dataFrame = dataFrame.iloc[:,0:n_components]
        elif key == 'bw':
            # Bandwidth is either a number or a dataFrame
            if type(value) == pd.core.frame.DataFrame:
                # normal functionality
                if not(value.shape[0] == n_components and
                ↪  value.shape[1] == n_components):
                    raise ValueError(f"DataFrame of incompatible
                     ↪  length. Expected shape was
                     ↪  ({n_components},{n_components})")
            else:
                bw = value
        else:
            return(print('unrecognized key'))

    comb = combinations(dataFrame.columns,2)

    # The following code expects that the bw is in the form of a
     ↪  pandas dataframe with indexes and colums being named
    # PC1,...,PC{n_components} according to the principal
     ↪  component number. Thus if bw is a number, we convert it
     ↪  to a
    # dataFrame that is bw in all positions.
    if not(type(bw)==pd.core.frame.DataFrame):
        PClist = [f"PC{i+1}" for i in range(n_components)]
        bw = np.ones((n_components,n_components))*bw
        bw = pd.DataFrame(bw, columns=PClist, index = PClist)

    for pair in comb:
        plt.figure()
        print(f'Bandwidth {pair[0]} {pair[1]}:
         ↪  {bw.loc[pair[0],pair[1]]}')
        sns.kdeplot(dataFrame.loc[:,pair[0]],dataFrame.loc[:,pai⌋
         ↪  r[1]], shade = True,
         ↪  bw=bw.loc[pair[0],pair[1]])
        plt.xticks([])
        plt.yticks([])
        plt.xlabel(pair[0])
        plt.ylabel(pair[1])
        plt.savefig(savepath + '/' + pair[0] + '_' + pair[1] +
         ↪  '_'+ 'kde_plot.pdf')
        plt.close()


if __name__ == "__main__":
    # For testing
    pass
```

A.4.2 Implementation of the DB* and DB** indices

```python
""""Cluster validity indices """

import numpy as np
from sklearn.metrics.pairwise import pairwise_distances
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import safe_indexing, check_X_y
from sklearn.metrics.cluster.unsupervised import check_number_of_labels

# Author: Bendik Skarre Abrahamsen <bendik.s.abrahamsen@gmail.com>

def davies_bouldin_star_score(X, labels):
    """Computes the Davies-Bouldin star score.
    The score was proposed as an improvement to the original Davies-Bouldin
    score [1] by Kim and Ramakrishna [2] by using the minimum of the
    between centroid distances instead of just the between centroid
    distances. Read more in the :ref:`User Guide <davies-bouldin_index>`.

    Parameters
    ----------
    X : array-like, shape (``n_samples``, ``n_features``)
        List of ``n_features``-dimensional data points. Each row corresponds
        to a single data point.
    labels : array-like, shape (``n_samples``,)
        Predicted labels for each sample.
    Returns
    -------
    score: float
        The resulting Davies-Bouldin score.
    References
    ----------
    .. [1] Davies, David L.; Bouldin, Donald W. (1979).
       `"A Cluster Separation Measure"
       <https://ieeexplore.ieee.org/document/4766909>`__.
       IEEE Transactions on Pattern Analysis and Machine Intelligence.
       PAMI-1 (2): 224-227

    ..[2] Minho Kim, R.S. Ramakrishna,
       New indices for cluster validity assessment,
       Pattern Recognition Letters,
       Volume 26, Issue 15,
       2005,
       Pages 2353-2363,
       ISSN 0167-8655,
       https://doi.org/10.1016/j.patrec.2005.04.007.
       (http://www.sciencedirect.com/science/article/pii/S016786
    ↪  550500125X)
    """

    X, labels = check_X_y(X, labels)
    le = LabelEncoder()
    labels = le.fit_transform(labels)
    n_samples, _ = X.shape
    n_labels = len(le.classes_)
    check_number_of_labels(n_labels, n_samples)

    intra_dists = np.zeros(n_labels)
    centroids = np.zeros((n_labels, len(X[0])), dtype=np.float)
    for k in range(n_labels):
        cluster_k = safe_indexing(X, labels == k)
        centroid = cluster_k.mean(axis=0)
        centroids[k] = centroid
        intra_dists[k] = np.average(pairwise_distances(
            cluster_k, [centroid]))

    centroid_distances = pairwise_distances(centroids)

    if np.allclose(intra_dists, 0) or
    ↪  np.allclose(centroid_distances, 0):
        return 0.0

    # intra_distsmatrix
    intra_dists_m = intra_dists[:, None] + intra_dists
    # We don't want elements along the diagonal
    di = np.diag_indices(intra_dists_m.shape[0])
    intra_dists_m[di] = np.nan
    np.fill_diagonal(centroid_distances, np.inf)
    score = np.nanmax(intra_dists_m, axis=1) /
    ↪  np.min(centroid_distances, axis=0)

    return np.mean(score)

def davies_bouldin_star_star_score(X, labels):
    """Computes the Davies-Bouldin star score.
    The score was proposed as an improvement to the original Davies-Bouldin
    score [1] by Kim and Ramakrishna [2] by using the minimum of the
    between centroid distances instead of just the between centroid
    distances.  In addition it uses the maxdiff between see Kim et al [2]
    for details.
    Read more in the :ref:`User Guide <davies-bouldin_index>`.

    Parameters
    ----------
    X : array-like, shape (``n_samples``, ``n_features``)
        List of ``n_features``-dimensional data points. Each row corresponds
        to a single data point.
    labels : array-like, shape (``n_cmax``,``n_samples``)
        Predicted labels for each sample for each proposed k.
    Returns
    -------
    score: list of floats
        The resulting Davies-Bouldin scores up to but not including
        n_cmax. Where n_cmax is the larger number of clusters that
        the clustering algorithm is run for.
    References
    ----------
    .. [1] Davies, David L.; Bouldin, Donald W. (1979).
       `"A Cluster Separation Measure"
       <https://ieeexplore.ieee.org/document/4766909>`__.
       IEEE Transactions on Pattern Analysis and Machine Intelligence.
       PAMI-1 (2): 224-227

    ..[2] Minho Kim, R.S. Ramakrishna,
       New indices for cluster validity assessment,
       Pattern Recognition Letters,
       Volume 26, Issue 15,
       2005,
       Pages 2353-2363,
       ISSN 0167-8655,
       https://doi.org/10.1016/j.patrec.2005.04.007.
```

```
        (http://www.sciencedirect.com/science/article/pii/S016786
↪   550500125X)
    """
    score = []
    e1 = []
    e2 = []
    for labels in labels:

        X, labels = check_X_y(X, labels)
        le = LabelEncoder()
        labels = le.fit_transform(labels)
        n_samples, _ = X.shape
        n_labels = len(le.classes_)
        check_number_of_labels(n_labels, n_samples)

        intra_dists = np.zeros(n_labels)
        centroids = np.zeros((n_labels, len(X[0])),
        ↪  dtype=np.float)
        for k in range(n_labels):
            cluster_k = safe_indexing(X, labels == k)
            centroid = cluster_k.mean(axis=0)
            centroids[k] = centroid
            intra_dists[k] = np.average(pairwise_distances(
                cluster_k, [centroid]))

        centroid_distances = pairwise_distances(centroids)

        if np.allclose(intra_dists, 0) or
        ↪  np.allclose(centroid_distances, 0):
            return 0.0

        # intra_distsmatrix
        intra_dists_m = intra_dists[:, None] + intra_dists
        # We don't want elements along the diagonal
        di = np.diag_indices(intra_dists_m.shape[0])
        intra_dists_m[di] = np.nan
        np.fill_diagonal(centroid_distances, np.inf)
        e1.append(np.nanmax(intra_dists_m, axis=1))
        e2.append(np.min(centroid_distances, axis=0))

    diff_nc = [e1[i]-e1[i+1][:-1] for i in range(len(e1)-1)]

    return [np.mean((e1[i]+np.max(diff_nc[i]))/e2[i]) for i in
    ↪  range(len(diff_nc))]


```

```python
if __name__ == "__main__":
    # For testing only
    pass
```

A.4.3 z-scoring and PCA

1 Rebuilding the dataset without interactions

In this section the dataset will be loaded from a matlab file and buildt into pandas dataframe. Each feature will first be z-scored and it will then be perfromed PCA on the resulting dataset. This will produce a dataset of principal components. This dataset will be saved and imported in the notebooks that perform the *k*-means and Gaussian mixture model clustering.

```
In [17]: from scipy.io import loadmat
         import numpy as np
         import seaborn as sns
         import matplotlib.pyplot as plt
         import pandas as pd
         import os
         from IPython.display import display, HTML
         from scipy.stats import shapiro, kstest, normaltest

         from sklearn.preprocessing import PolynomialFeatures, StandardScaler
         from sklearn.decomposition import PCA
         %matplotlib inline

         # for profiling
         import timeit

         # three significant digist is more than enough
         pd.options.display.float_format = '{:.2e}'.format
```

```
In [18]: # Loading the latest version of the FeatureArray
         featureArray = loadmat('./../Data/FeatureArray/featureArraysFinalv2.mat')
```

```
In [19]: for elem in featureArray.keys():
             print(elem)
```

```
__header__
__version__
__globals__
FileNames
featureArray
grpsPasInd
```

```
In [20]: columns = ['T2','DWI0','DWI25','DWI50','DWI100','DWI500','DWI1000','ADC',
             'D','Dstar','f']
         featureDF = pd.DataFrame(featureArray['featureArray'], columns=columns)
```

```
In [21]: featureDF.describe()
```

```
Out[21]:             T2     DWI0    DWI25    DWI50   DWI100   DWI500  DWI1000      ADC  \
         count 3.92e+06 3.92e+06 3.92e+06 3.92e+06 3.92e+06 3.92e+06 3.92e+06 3.92e+06
         mean  6.32e+02 7.42e+02 6.71e+02 6.42e+02 5.95e+02 3.69e+02 2.55e+02 1.34e+04
```

1

```
std   2.02e+02 3.01e+02 2.76e+02 2.60e+02 2.39e+02 1.39e+02 9.84e+01 4.43e+03
min   0.00e+00 6.11e+01 7.21e+01 7.42e+01 6.44e+01 5.75e+01 4.47e+01 1.25e+02
25%   5.03e+02 5.43e+02 4.90e+02 4.71e+02 4.38e+02 2.76e+02 1.85e+02 1.04e+04
50%   6.07e+02 7.03e+02 6.34e+02 6.09e+02 5.66e+02 3.50e+02 2.35e+02 1.29e+04
75%   7.28e+02 8.92e+02 8.04e+02 7.68e+02 7.14e+02 4.42e+02 3.07e+02 1.61e+04
max   2.61e+03 4.15e+03 3.77e+03 3.38e+03 3.05e+03 1.71e+03 1.20e+03 6.53e+04

              D    Dstar        f
count  3.92e+06 3.92e+06 3.92e+06
mean   1.17e+04 1.13e+04 1.06e+04
std    4.56e+03 4.22e+03 4.87e+03
min   -3.11e-12 0.00e+00 0.00e+00
25%    8.69e+03 8.42e+03 7.21e+03
50%    1.14e+04 1.10e+04 1.04e+04
75%    1.45e+04 1.40e+04 1.37e+04
max    3.26e+04 3.23e+04 3.25e+04
```

```
In [6]: normaltest(featureDF.DWI0)
```

```
Out[6]: NormaltestResult(statistic=778427.9879091659, pvalue=0.0)
```

1.1 Performing dimensionality reduction and principal component analysis

We'll perform dimensionality reduction by PCA. However, since the PCA finds the linear combination of features that explains the greates amount of variance in the dataset, it is not independent of the scale of the data. We thus will, as is customary, rescale the data using a z-transform. That is for each sample x_{ij} we subtract the feature mean μ_j and divide by the feature standard deviation. We thus obtain features with means of zero and standard deviations (and variances of 1).

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma}$$

```
In [7]: # initializing a scaler object
        scaler = StandardScaler()
        # Storing column names
        columnNames = featureDF.columns
        featureDF_z = pd.DataFrame(scaler.fit_transform(featureDF),
                                   columns=columnNames)
        pd.options.display.float_format = '{:.2f}'.format
        featureDF_z.describe()
```

```
Out[7]:              T2       DWI0      DWI25      DWI50     DWI100     DWI500  \
        count 3920118.00 3920118.00 3920118.00 3920118.00 3920118.00 3920118.00
        mean        0.00      -0.00       0.00      -0.00       0.00      -0.00
        std         1.00       1.00       1.00       1.00       1.00       1.00
        min        -3.13      -2.26      -2.17      -2.18      -2.22      -2.24
        25%        -0.64      -0.66      -0.65      -0.66      -0.66      -0.67
        50%        -0.13      -0.13      -0.13      -0.12      -0.12      -0.14
        75%         0.47       0.50       0.48       0.49       0.50       0.53
```

```
max         9.77      11.34      11.22      10.53      10.29       9.63

           DWI1000        ADC          D      Dstar          f
count 3920118.00 3920118.00 3920118.00 3920118.00 3920118.00
mean       -0.00       0.00      -0.00      -0.00      -0.00
std         1.00       1.00       1.00       1.00       1.00
min        -2.13      -3.00      -2.55      -2.67      -2.18
25%        -0.71      -0.68      -0.65      -0.67      -0.70
50%        -0.20      -0.12      -0.06      -0.07      -0.05
75%         0.54       0.60       0.62       0.65       0.64
max         9.65      11.71       4.60       4.98       4.49
```

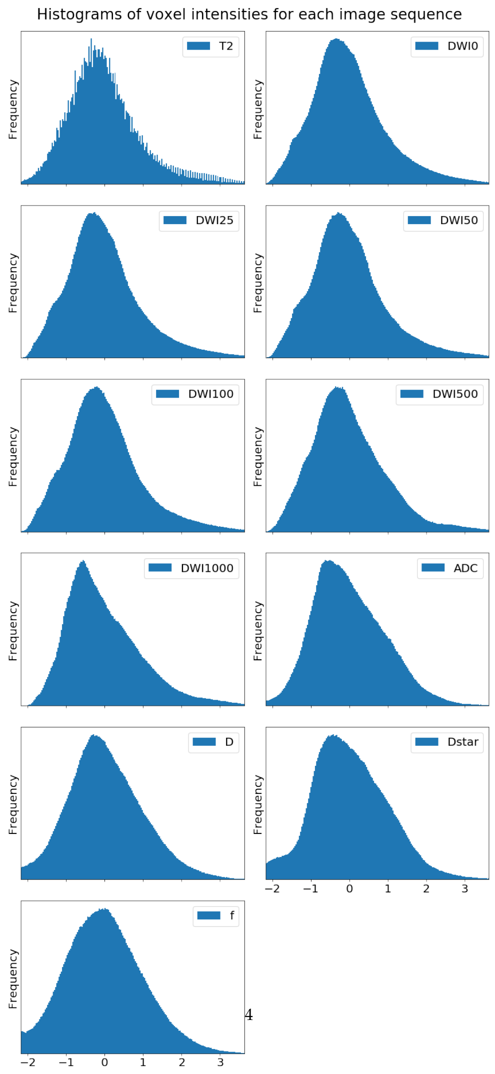
```
In [8]: featureDF_z.to_feather('./Data/featureDFRawZScaled')
```

```
In [9]: xlimH = np.percentile(featureDF_z.values,99.95,axis=0)
        xlimL = np.min(featureDF_z.values, axis=0) #-np.percentile(-featureDF_z.values,99,axis=0
        print(xlimH, xlimL)
```

```
[ 6.18029372  5.74551609  5.45867529  5.46083993  5.46679237  5.50761735
  5.447619   10.12628008  3.790955    3.93436736  3.62762841] [-3.1291956  -2.26257816 -2.168768
 -2.13257335 -2.99970649 -2.55475731 -2.67142681 -2.18248399]
```

```
In [25]: axList = featureDF_z.plot.hist(subplots=True,figsize=(12,26),
                                        layout=(-1,2),bins=500, color = 'C0', density=True,
                                       )
         axList = np.array(axList).reshape(-1)
         for c,ax in enumerate(axList):
             if c < featureDF.shape[1]:
                 ax.set_xlim((xlimL[c], xlimH[c]))
                 ax.set_yticks([])
                 ax.legend(fontsize=20)
                 ax.set_ylabel(ax.get_ylabel(), fontsize=20)
                 try:
                     ax.tick_params(axis='both', which='major', labelsize=20)
                 except:
                     pass
         plt.tight_layout(rect=[0, 0, 1, 0.98])
         plt.suptitle('Histograms of voxel intensities for each image sequence', va='bottom',siz
         plt.savefig('./figures/histograms/DFZHistogramNoInteractions.pdf')
```

3

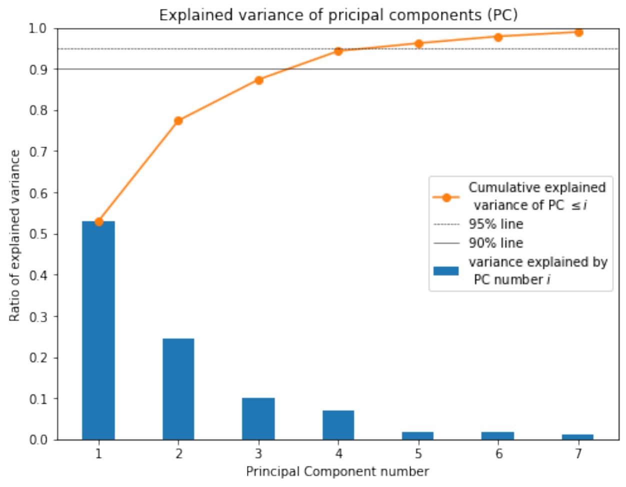

Histograms of voxel intensities for each image sequence
T2
DWI0
DWI25
DWI50
DWI100
DWI500
DWI1000
ADC
D
Dstar
f
Frequency
−2
−1
0
1
2
3

4

We'll now perform the PCA

```
In [12]: # initializing a PCA object
         pca_scaler = PCA()
         # creating the new column names
         PC = [f'PC{i+1}' for i in range(featureDF_z.shape[1])]
         featuresPCA = pd.DataFrame(pca_scaler.fit_transform(featureDF_z), columns= PC)
```

```
In [13]: plt.figure(figsize=(8,6))
         expl = pca_scaler.explained_variance_ratio_
         sumExpl = np.cumsum(expl)
         # creating a list of components that together explain up to but not including 99% of th
         x,y = zip(*[(c+1, expl) for (c, expl) in enumerate(sumExpl) if expl < 0.99])

         line = plt.plot(x,y, '-o', color = "tab:orange", label = "Cumulative explained\n varian
         hline = plt.axhline(0.95, color='k', linestyle = '--', linewidth=0.5, label = '95% line
         hline = plt.axhline(0.9, color='k', linestyle = '-', linewidth=0.5, label = '90% line')
         barC = plt.bar(x, expl[:len(x)], width = 0.4, label =  "variance explained by\n PC numb
         plt.legend(loc=7)
         plt.title("Explained variance of pricipal components (PC)")
         plt.xlabel("Principal Component number")
         plt.ylabel("Ratio of explained variance")
         plt.yticks(np.arange(0,1.1,0.1))
         plt.ylim(0, 1)
         plt.savefig('./figures/Pareto-NoInteractions.pdf')
```



Over 90% of the variance of the dataset is explained by the first four principal components and over 95% of the variance is explained by the first five principal components. The three first principal components explain almost 90% of the variance.

```
In [22]: print(f"The first three component explain {sumExpl[2]*100:.2f}% of the variance.")

The first three component explain 87.35% of the variance.
```

A table showing the comulative explained variance of the first 8 components is shown below.

```
In [30]: pd.options.display.float_format = '{:.2%}'.format
         pd.DataFrame([sumExpl[0:8]],
                            columns=featuresPCA.iloc[0:1,0:8].columns).rename(
             {0:'Cumulative variance explained'}, axis=0)

Out[30]:                                  PC1    PC2    PC3    PC4    PC5    PC6  \
         Cumulative variance explained 53.03% 77.43% 87.35% 94.30% 96.20% 97.87%

                                          PC7    PC8
         Cumulative variance explained 98.94% 99.45%
```

6

1.2 Saving the dataset

We now save the dataset in a python friendly format so that we can access it more easily when visualizing the data, running the *k*-means algorithm and Gaussian mixture.

```
In [14]: featuresPCA.to_feather('./Data/featuresPCAnoInteractions')
```

A.4.4 k-means Cluster Analysis

1 *k*-means clustering

```
In [ ]: import pandas as pd
        import numpy as np
        from sklearn.cluster import KMeans
        from sklearn.metrics import calinski_harabaz_score, davies_bouldin_score
        from cluster_validity_indices import davies_bouldin_star_score, davies_bouldin_star_star_score
        from sklearn.metrics import silhouette_score

        from sklearn.preprocessing import MinMaxScaler

        import matplotlib.pyplot as plt
        import seaborn as sns
        import os

        %matplotlib notebook
```

We want to use the *k*-means clsutering algorithm to cluster our data while also writing about the choice of selected parameters and options in the given implementation of the algorithm. We'll use the scikit-learn implementation of the *k*-means algorithm. We'll time the algorithm to see how fast it can cluster the full datasets, and we'll find the optimal *k* based on the DB, DB*, DB** and Silhouette coefficients.

First the dataset needs to be loaded

```
In [ ]: dataset = pd.read_feather('./Data/featuresPCAnoInteractions')
        # when testing the code a testset of 100000 samples will be used.
        figSavePath = './figures/k-meansClusteringFiguresNoInteractions'
        labelsSaveName = 'kmeansLabelsNoInteractions'
        resultsSavePath = './Results/kmeans/'
        test = 0
        if test:
            dataset = dataset.sample(10000)
            figSavePath = figSavePath + 'Test'
            labelsSaveName = labelsSaveName + 'Test'
        try:
            os.mkdir(figSavePath)
        except:
            print('Folder already exist')
```

1.1 Timing the *k*-means algorithm and checking for convergence

We want to see how fast the *k*-means algorithm is on the dataset for different *k*.

```
In [ ]: kMeansTime = []
        for k in np.arange(2,3):
            kmeans = KMeans(n_clusters=k,max_iter=600,
                            n_init=30,n_jobs=-1)
            time = %timeit -o -n1 -r1 kmeans.fit(dataset.iloc[:,0:4])
```

```
In [ ]: time.average
```

It takes roughly 150 seconds to perform *k*-means on the entire dataset. This is by using the standard parameters of the *k*-means implementation in sklearn. These are: using *k*-means++ to

1

assign the initial clsuter centers in a smart manner. This is done by selecting samples that are fairly far apart from eachother. For details about the *k*-means++ algorithm see Arthur and Vassilvitskii [1]. The *k*-means is run 10 times with different centroid seeds where the best ouput and selected clustering solution is the one that minimizes the within-cluster sum of squares distances. Max iterations is set to 300. The number of iterations should be checked after each completed run of the algorithm to see that convergence has occured before the maximum number of iteration has been reached. This can be accessed in the kmeansObject.n_iter_ field.

```
In [ ]: kmeansConvergenceTest = KMeans(n_clusters=5, max_iter=300,
                                       n_init=1, init='random').fit(dataset)
```

```
In [ ]: kmeansConvergenceTest.n_iter_
```

How long it takes to run the *k*-means algorithm is dependent on how fast it reaches convergence and stops. The *k*-means algorithm runs the maximal number of iteration in the worst case and is normally much quicker than this.

1.2 Running the *k*-means algorithm for different *k*

We want to test the *k*-means algorithm for *k* ranging from 2 to approxiamtely 20 and check ther performance by using different criterea. We'll save the resulting labels from the clustering with different k's. For k's where the clustering does not converge in the maximal number of iterations the reuslts will be flagged. Four pricipal components have been shown to contain more than 90% of the total variance of the dataset. Only these four principal components will be used in the analysis.

```
In [ ]: kList = np.arange(2,22)
        convergence = [] #will append logical values if KMeans.n_iter_ != KMeans.
        columns = [f'k={k}' for k in kList]
        labelsDF = pd.DataFrame()
        principalComponents = 4

        datasetRed = dataset.iloc[:,0:principalComponents]

        for c,k in enumerate(kList):
            print(c,k)
            kmeans = KMeans(n_clusters=k,max_iter=600,
                            n_init=30,n_jobs=-1).fit(datasetRed)
            labelsDF.loc[:,columns[c]] = kmeans.labels_
            if kmeans.n_iter_ != kmeans.max_iter:
                convergence.append(True)
        labelsDF.reset_index().to_feather(os.path.join(resultsSavePath,
                                                       labelsSaveName))
        np.save(os.path.join(resultsSavePath,
                labelsSaveName + 'convergence'),np.array(convergence))
```

We check that the algorithm reaches convergence for all the *k*. Technically we only check the chosen solution, which is the best solution in terms of intracluster distance squared. This should be the solution that is most likely to converge if any.

```
In [ ]: convergence
```

Now that we have DataFrame with the labels for the different *k* we can use these to calculate some cluster validity indices.

1.2.1 Calinski-Harabasz

```
In [ ]: CHScore = []
        for k in columns:
            print(k)
            CHScore.append(calinski_harabaz_score(datasetRed, labelsDF.loc[:,k]))
```

```
In [ ]: plt.figure(figsize=(8,4))
        sns.lineplot(x=kList, y=CHScore)
        plt.xticks(kList)
        plt.title(r'Calinski-Harabsz score $(\uparrow)$')
        plt.xlabel('k')
        plt.yticks([])
        plt.tight_layout()
        plt.savefig(os.path.join(figSavePath,'CHScore.pdf'))
```

```
In [ ]: print('The optimal k evaluated with the Calinski-Harabasz'+
              'criteria is {kList[np.argmax(CHScore)]}.')
```

1.2.2 Davies-Bouldin Famillily of Indices

```
In [ ]: DB = []
        DBs = []
        DBss = []
        np.seterr(divide='ignore')
        # in sklearns DB implementation the code deliberately divides by zero
        # and this causes numpy to throw an error. We want to ignore this.
        for k in columns:
            print(k)
            DB.append(davies_bouldin_score(datasetRed, labelsDF.loc[:,k]))
            DBs.append(davies_bouldin_star_score(datasetRed,labelsDF.loc[:,k]))
```

```
In [ ]: DBss = davies_bouldin_star_star_score(datasetRed, labelsDF.values.T)
```

```
In [ ]: def scale(elemList):
            return((elemList - min(elemList))/(max(elemList)-min(elemList)))
        plt.figure()
        sns.lineplot(x=kList[:-1], y=scale(DBss), label = 'DB**')
        sns.lineplot(x=kList, y=scale(DBs), label = 'DB*')
        sns.lineplot(x=kList, y=scale(DB), label='DB')
        plt.legend()
        plt.yticks([])
        plt.xticks(kList)
        plt.xlim([1, 22])
        plt.ylim([-0.01, 1.01])
        plt.title('Davies-Bouldin CVIs ($\downarrow$)')
        plt.ylabel('Score')
        plt.tight_layout()
        plt.savefig(os.path.join(figSavePath,'DBfamilyScores.pdf'))
```

```
In [ ]: print(f'The selected k are:\nDB:   {kList[np.argmin(DB)]}' +
              f'\nDB*:  {kList[np.argmin(DBs)]}\n'+
             f'DB**: {kList[np.argmin(DBss)]}')
```

1.2.3 Silhouette coefficient

The silhuette coefficient goes as n^2 in the number of samples. Thus we cannot run it on the entire dataset as it would be very slow. We will thus subsample the dataset a few times and look at the optimal k for each subsample. This will hopefully yield quite similar results in each fold.

```
In [ ]: labelsDF.index = datasetRed.index
```

```
In [ ]: silScore = []
        datasetWLabels = pd.concat([datasetRed,labelsDF], axis=1)
        sampleSize = 5000
        repetitions = 20
        for column in columns:
            print(column)
            temp = []
            for i in range(0,repetitions):
                dfSample = datasetWLabels.sample(sampleSize)
                temp.append(silhouette_score(dfSample.iloc[:,0:principalComponents],
                                             dfSample.loc[:,column]))
            silScore.append(temp)
```

Visualizing the silhouette scores for each k as a lineplot. The errorbars are the standard deviation for the bootstraps of a single k. As we can see, the standard deviation is quite small compared to the difference between different k values. As mentioned earlier, this implies that we don't need to include more samples in our bootstraps.

```
In [ ]: plt.figure(figsize=(8,6))
        repetitions = 20
        kLabels = np.array([[k]*repetitions for k in kList]).reshape(-1)
        sns.lineplot(x=kLabels, y=np.array(silScore).reshape(-1),
                     err_style="bars", ci="sd")
        plt.xticks(kList)
        plt.title(r'Silhouette score ($\uparrow$)')
        plt.xlabel('k')
        plt.ylabel('Score')
        plt.tight_layout()
        plt.savefig(os.path.join(figSavePath,'SilhouetteScores.pdf'))
```

1.3 Concluding remarks for *k*-means clustering validity

As can be seen in the plots above, clustering validity indices indicate that either 2 or 5 are the optimal indices.

[1] - "k-means++: The advantages of careful seeding" Arthur, David, and Sergei Vassilvitskii, Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics (2007)

A.4.5 Gaussian Mixture Models Cluster Analysis

1 Gaussian mixture models

We attempt to use gaussian mixture models to find a suitable number of components that is able to describe the dataet.

```
In [11]: %matplotlib notebook
         import numpy as np
         import seaborn as sns
         import pandas as pd
         import os
         from sklearn.mixture import GaussianMixture
         from joblib import Parallel, delayed
         import matplotlib.pyplot as plt
         from scipy.io import loadmat
         from lifelines.statistics import logrank_test
         from lifelines import KaplanMeierFitter
         from sklearn.cluster import KMeans

         from cluster_validity_indices import (davies_bouldin_star_score,
         davies_bouldin_star_star_score)
         from sklearn.metrics import silhouette_score, davies_bouldin_score
         from statsmodels.stats.multitest import multipletests
```

1.1 Loading the dataset

```
In [3]: dataset = pd.read_feather('./Data/featuresPCAnoInteractions')
        figSavePath = './GMMClusteringFigures'
        labelsSaveName = './Results/GMM/GMMLablesNoInteractions'

        try:
            os.mkdir(figSavePath)
        except:
            print('Folder already exists')
```

```
Folder already exists
```

1.2 Performing the GMM clustering

```
In [4]: PCs = 4
        kList = np.arange(1,51)
        reps = 5
        N = 300000 # number of samples
        n_init = 3
```

```
In [5]: datasetRed = dataset.iloc[:,0:4]
```

```
In [6]: def fitGaussianMixture(X, k):
            return(GaussianMixture(n_components=k,
                            max_iter=300, n_init=n_init).fit(X))
```

```
In [6]: mixMods = Parallel(n_jobs=-2,
                           verbose=10)(delayed(fitGaussianMixture
                                        )(datasetRed.sample(N,random_state=r),
                                        k) for k in kList for r in range(reps))
```

```
[Parallel(n_jobs=-2)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=-2)]: Done   2 tasks      | elapsed:    6.6s
[Parallel(n_jobs=-2)]: Done   7 tasks      | elapsed:   20.9s
[Parallel(n_jobs=-2)]: Done  12 tasks      | elapsed:   57.7s
[Parallel(n_jobs=-2)]: Done  19 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-2)]: Done  26 tasks      | elapsed:  5.5min
[Parallel(n_jobs=-2)]: Done  35 tasks      | elapsed:  8.5min
[Parallel(n_jobs=-2)]: Done  44 tasks      | elapsed: 12.8min
[Parallel(n_jobs=-2)]: Done  55 tasks      | elapsed: 19.6min
[Parallel(n_jobs=-2)]: Done  66 tasks      | elapsed: 30.4min
[Parallel(n_jobs=-2)]: Done  79 tasks      | elapsed: 42.3min
[Parallel(n_jobs=-2)]: Done  92 tasks      | elapsed: 57.2min
[Parallel(n_jobs=-2)]: Done 107 tasks      | elapsed: 79.2min
[Parallel(n_jobs=-2)]: Done 122 tasks      | elapsed: 103.7min
[Parallel(n_jobs=-2)]: Done 139 tasks      | elapsed: 137.2min
[Parallel(n_jobs=-2)]: Done 156 tasks      | elapsed: 172.8min
[Parallel(n_jobs=-2)]: Done 175 tasks      | elapsed: 224.4min
[Parallel(n_jobs=-2)]: Done 194 tasks      | elapsed: 280.8min
[Parallel(n_jobs=-2)]: Done 215 tasks      | elapsed: 347.6min
[Parallel(n_jobs=-2)]: Done 236 tasks      | elapsed: 420.8min
[Parallel(n_jobs=-2)]: Done 250 out of 250 | elapsed: 469.5min finished
```

Saving the gaussian mixture models.

```
In [13]: np.save(f'./Results/GMM/mixModels{N}samples{reps}reps'+
         f'{n_init}inits{np.min(kList)}kmin{np.max(kList)}kMaxNoInteractions',
                mixMods)
```

```
In [12]: mods = np.load('./Results/GMM/mixModels300000samples5reps'+
                        '3inits1kmin50kMaxNoInteractions.npy')
```

Since no convergence is clearly visible (see separate code for this figure), we'll have to try a few more *k*.

```
In [ ]: kList2 = np.arange(51, 71)
        mixMods2 = Parallel(n_jobs=-2,
                           verbose=10)(delayed(fitGaussianMixture
                            )(datasetRed.sample(N,random_state=r),
                            k) for k in kList2 for r in range(reps))
```

```
In [15]: np.save(f'./Results/GMM/mixModels{N}samples{reps}reps{n_init}'+
                 f'inits{np.min(kList2)}kmin{np.max(kList2)}kMaxNoInteractions',
                mixMods2)
```

A.4.6 k-means survival analysis

1 Interpreting the *k* - means results

It seems like the most likely number of clsuters using *k*-means on the full dataset without interaction features included is two clusters. In this notebook we'll try to explore the characteristics of the clusters. Specifically we'll map the clusters back on to histograms to look at the feature characteristic of a single group.

```
In [82]: %matplotlib inline
         import numpy as np
         import pandas as pd
         import seaborn as sns
         import matplotlib.pyplot as plt
         from scipy.io import loadmat
         from os import makedirs
         from os import path

         from jupyterthemes import jtplot
         jtplot.reset()

         from lifelines import KaplanMeierFitter
         from lifelines.statistics import logrank_test
         from scipy.io import loadmat
         from matplotlib.patches import Rectangle
         from statsmodels.stats.multitest import multipletests
```

```
In [83]: survivalCurvesFigurePath = './figures/survivalCurves/NoInteractions'
```

1.0.1 Loading the dataset and labels from *k*-means

```
In [84]: dataset = pd.read_feather('./Data/featuresPCAnoInteractions')
         labels = pd.read_feather('./Results/kmeans/kmeansLabelsNoInteractions')
         labels.set_index('index'); #semicolon supresses output
```

Checking that the loading was performed correclty and that the dataset and labels are the correct shapes.

```
In [85]: pd.options.display.float_format = '{:.2f}'.format
```

```
In [ ]: dataset.describe()
```

```
In [ ]: labels.describe()
```

```
In [88]: print(dataset.shape)
         print(labels.shape)

(3920118, 11)
(3920118, 21)
```

```
In [89]: datasetRaw = pd.read_feather('./Data/featureDFRawZScaled')
```

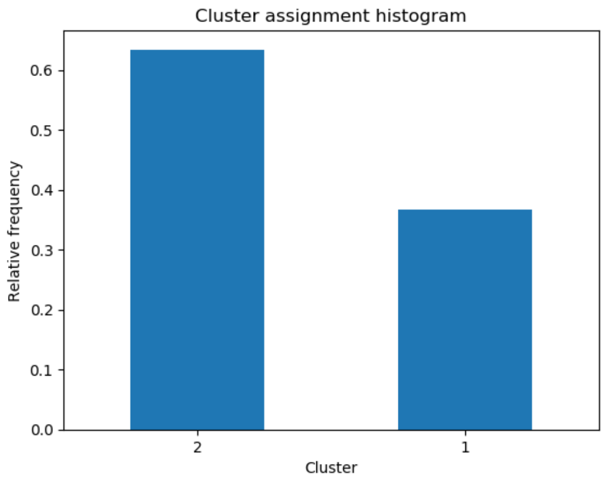
```
In [ ]: datasetRaw.describe()
```

1.0.2 Cluster assignment

A bar chart depicting the number of voxels assigned to each cluster.

```
In [91]: pd.options.display.float_format = '{:.4f}'.format
         valCount=(labels['k=2']+1).value_counts().divide(labels.shape[0])
         print(valCount)
         valCount.plot(kind = 'bar')
         plt.xlabel('Cluster')
         plt.ylabel('Relative frequency')
         plt.xticks(rotation = 0)
         plt.title('Cluster assignment histogram')
         plt.show()
```

```
2   0.6340
1   0.3660
Name: k=2, dtype: float64
```

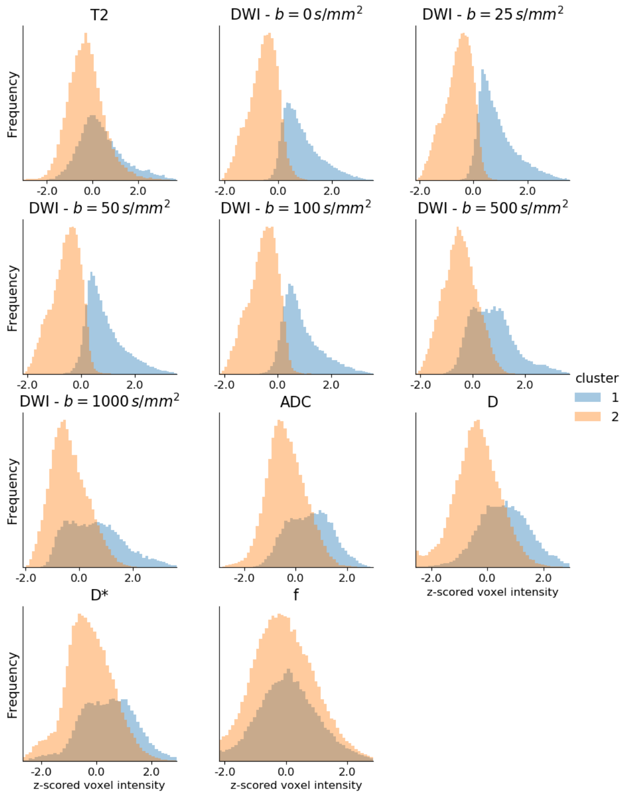


1.0.3 Plotting histograms for each image sequence and cluster

```
In [8]: datasetRawLong = datasetRaw.copy().sample(100000)
        datasetRawLong['cluster'] = labels.loc[:,'k=2']
```

The axis limits are set using the minimum or maximum of the minimums or maximums and the 99.5 percentile of the feature values for each feature to reduce noise. Voxels exceeding these limits on either feature is dropped from the dataset when plotting the histogram to make automatic calculation of the number of bins and bin edges perform as expected.

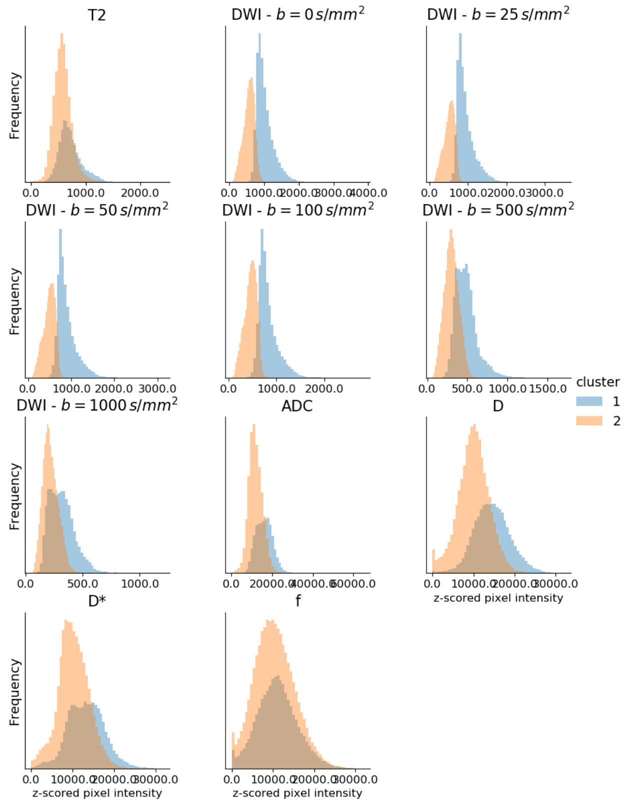
```
In [9]: percentiles = np.percentile(abs(datasetRawLong.iloc[:,:-1].values), 99.5, axis=0)
        mins = np.min(datasetRawLong.iloc[:,:-1].values, axis=0)
        maxes = np.max(datasetRawLong.iloc[:,:-1].values, axis=0)
        xLimH = [maxes[c] if maxes[c] < percentiles[c] else percentiles[c] for c,
        col in enumerate(datasetRawLong.iloc[:,:-1].columns)]
        xLimL = [mins[c] if mins[c] > -percentiles[c] else -percentiles[c] for c,
                col in enumerate(datasetRawLong.iloc[:,:-1].columns)];

In [10]: s1=datasetRawLong.shape
         for c,col in enumerate(datasetRawLong.columns[:-1]):
             datasetRawLong = datasetRawLong[datasetRawLong.loc[:,col] < xLimH[c]]
             datasetRawLong = datasetRawLong[datasetRawLong.loc[:,col] > xLimL[c]]
         s2=datasetRawLong.shape

In [11]: datasetRawLong = pd.melt(datasetRawLong, id_vars=['cluster'],
                                  var_name='ImageSeq', value_name='pixelIntensity')
         datasetRawLong['cluster'] = datasetRawLong['cluster']+1

In [12]: g = sns.FacetGrid(datasetRawLong, col='ImageSeq', hue='cluster', col_wrap=3,
                           sharex=False, sharey=False, legend_out = True)
         g = g.map(sns.distplot, 'pixelIntensity', kde = False)

         axes = g.axes.flatten()

         # Creating titles for the plot
         bVals = [0, 25, 50, 100, 500, 1000]
         DWItitles = [r'DWI - $b = %d \, s/mm^2$' % (bVal) for bVal in bVals]
         titles = ['T2'] + DWItitles + ['ADC', 'D', 'D*', 'f']
         g.add_legend(fontsize = 14)
         plt.setp(g._legend.get_title(), fontsize=14)
         for c,title in enumerate(titles):
             axes[c].set_title(title, fontsize = 16)
             axes[c].set_yticks([])
             axes[c].set_xticklabels(axes[c].get_xticks(),fontsize=12)
             axes[c].set_xlim([xLimL[c], xLimH[c]])
             if c in range(0,len(titles), 3):
                 axes[c].set_ylabel('Frequency', fontsize=14)
         # setting x-label for the bottom plots
         for ax in axes[-3:]:
             ax.set_xlabel('z-scored voxel intensity', fontsize = 12)
         plt.savefig('./figures/histograms/'+
                     'featureHistogramWithClustersNoInteractions.pdf')
         plt.show()
```



```
In [13]: rawDataDF=pd.read_feather('./Data/featureDFRaw')
         rawDataDF['cluster'] = labels.loc[:,'k=2']+1
         rawDataDF = rawDataDF.copy().sample(200000)
         rawDataDFLong = pd.melt(rawDataDF, id_vars=['cluster'],
                                 var_name='ImageSeq', value_name='pixelIntensity')
         rawDataDFLongSampled = rawDataDFLong
```

4

```
In [14]: g = sns.FacetGrid(rawDataDFLongSampled, col='ImageSeq',
                           hue='cluster', col_wrap=3,
                           sharex=False, sharey=False, legend_out = True)
         g = g.map(sns.distplot, 'pixelIntensity', kde = False)

         axes = g.axes.flatten()

         # Creating titles for the plot
         bVals = [0, 25, 50, 100, 500, 1000]
         DWItitles = [r'DWI - $b = %d \, s/mm^2$' % (bVal) for bVal in bVals]
         titles = ['T2'] + DWItitles + ['ADC', 'D', 'D*', 'f']
         g.add_legend(fontsize = 14)
         plt.setp(g._legend.get_title(), fontsize=14)
         for c,title in enumerate(titles):
             axes[c].set_title(title, fontsize = 16)
             axes[c].set_yticks([])
             axes[c].set_xticklabels(axes[c].get_xticks(),fontsize=12)
             if c in range(0,len(titles), 3):
                 axes[c].set_ylabel('Frequency', fontsize=14)
         # setting x-label for the bottom plots
         for ax in axes[-3:]:
             ax.set_xlabel('z-scored pixel intensity', fontsize = 12)
         plt.savefig('./figures/histograms/'+
                     'featureHistogramWithClustersNoInteractionsRawFeatures.pdf')
         plt.show()
```



1.0.4 Survival analysis

We are loading the index postitions of each patient. There will be ranked according to volume. Since the DWI are interpolated accroding to the DWI images, we'll need to extract the volume

```
In [92]: FileNames = loadmat('./../Data/FeatureArray/featureArraysFinalv2.mat',
                             variable_names = 'FileNames')
         try:
             FileNames=FileNames['FileNames']
         except:
             pass
         FileNames = FileNames.flatten()
         for c,FileName in enumerate(FileNames):
             FileNames[c]=FileName[0][:-4]
```

```
In [93]: pasInd=loadmat('./../Data/FeatureArray/featureArraysFinalv2.mat',
                        variable_names = 'grpsPasInd')
         pasInd=pasInd['grpsPasInd']-1 #ptyhon is zero indexed while matlab is not
```

```
In [94]: dicomInfoTable=pd.read_csv('./../Data/dicomInfo/dicomInfoTable.csv')
         dicomInfoTable.set_index('Row', inplace=True)
```

```
In [95]: dicomInfoTable.index = [patName.lower() for patName in dicomInfoTable.index]
```

```
In [96]: dicomInfoTable.index.names = ['PatientName']
```

```
In [97]: voxelDF = pd.DataFrame(np.concatenate((FileNames.reshape(-1,1),
                                                pasInd),axis=1))
         voxelDF.set_index(0,inplace=True)
         voxelDF.index = [patName.lower() for patName in voxelDF.index]
         voxelDF.index.names = ['PatientName']
         voxelDF.columns = ['voxIndStart', 'voxIndEnd']
         voxelDF['totNumVox'] = voxelDF.voxIndEnd - voxelDF.voxIndStart +1
```

```
In [98]: valCounts = [pd.value_counts(labels.loc[:,'k=2'].iloc[start:end]) for start,
                                   end in zip(voxelDF.loc[:,'voxIndStart'].values,
                                              voxelDF.loc[:,'voxIndEnd'].values)]
```

```
In [99]: numpats =len(valCounts)
         grp1 = np.zeros((numpats,1))
         grp2 = np.zeros((numpats,1))
         for c, elem in enumerate(valCounts):
             try:
                 grp1[c] = elem.loc[0]
             except:
                 pass
             try:
                 grp2[c] = elem.loc[1]
             except:
                 pass
```

```
In [100]: voxelDF['numVoxClust1'] = grp1.reshape(-1,1).astype(int)
          voxelDF['numVoxClust2'] = grp2.reshape(-1,1).astype(int)
```

Converting the number of voxels to volumes by multiplying with pixeldimensions and slice thickness

```
In [101]: dicomInfoTable['voxelVolume'] = (dicomInfoTable['PixelSpacing_1']*
                                           dicomInfoTable['PixelSpacing_2']*
                                           dicomInfoTable['SliceThickness'])
```

7

```
In [102]: voxelDF = voxelDF.merge(dicomInfoTable['voxelVolume'],
                       how='inner', left_index=True, right_index=True)
```

```
In [103]: voxelDF['VolClust1'] = voxelDF.voxelVolume * voxelDF.numVoxClust1
          voxelDF['VolClust2'] = voxelDF.voxelVolume * voxelDF.numVoxClust2
          voxelDF['totVolume'] = voxelDF.voxelVolume * voxelDF.totNumVox
```

1.0.5 Survival Analysis

Loading the patient information

```
In [104]: patInfoCRT = pd.read_csv('./../Data/PatientSurvivalInfo/CRT.csv',
                                   encoding='latin-1')
          patInfoNoCRT = pd.read_csv('./../Data/PatientSurvivalInfo/NoCRT.csv',
                                     encoding='latin-1')
```

Converting fileNames to patient names

```
In [105]: patInfoCRT['PatientName'] = np.array([fileName[:-4].lower()
                                              for fileName in patInfoCRT.fileNames])
          patInfoNoCRT['PatientName'] = np.array([fileName[:-4].lower()
                                                for fileName in patInfoNoCRT.fileNames])
          patInfoCRT.set_index('PatientName',inplace=True)
          patInfoNoCRT.set_index('PatientName',inplace=True)
          survInfoCRT = patInfoCRT.loc[:,['timeFirstEventOrRCens', 'status']]
          survInfoNoCRT = patInfoNoCRT.loc[:,['timeFirstEventOrRCens', 'status']]
          survInfoCRT = survInfoCRT.merge(voxelDF.loc[:,'VolClust1':'VolClust2'],
                                          left_index=True, right_index=True)
          survInfoNoCRT = survInfoNoCRT.merge(voxelDF.loc[:,'VolClust1':'VolClust2'],
                                              left_index=True, right_index=True)
          print(survInfoCRT.shape)
          print(survInfoNoCRT.shape)
          survInfoCRT.reset_index().to_feather('./Data/survInfoCRT')
          survInfoNoCRT.reset_index().to_feather('./Data/survInfoNoCRT')
```

```
(24, 4)
(30, 4)
```

Making a table of descriptive statistics of the volume of each cluster

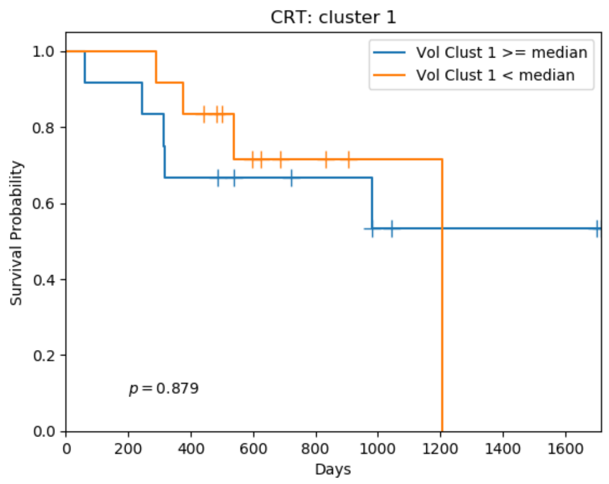
```
In [106]: survInfoCRTdescriptive = survInfoCRT.describe(percentiles=[]).loc[:,'VolClust1':'VolClust2']
          mn,mx = 'min', 'max'
          rng = [(f'({survInfoCRTdescriptive.loc[:,column].loc[mn]:.2f}-
                  {survInfoCRTdescriptive.loc[:,column].loc[mx]:.2f})')
                 for column in survInfoCRTdescriptive.columns]
          rng
          #survInfoCRTdescriptive.append(pd.DataFrame())
          survInfoCRTdescriptive=(
              survInfoCRTdescriptive.append(pd.DataFrame({'range':rng},
                              index=survInfoCRTdescriptive.columns).T))
          survInfoCRTdescriptive.rename({'50%':'median'}, axis=0, inplace=True)
          survInfoCRTdescriptive.drop('count', axis=0, inplace=True)
          pd.options.display.float_format = '{:.3g}'.format
          survInfoCRTdescriptive.rename({'VolClust1':'Volume Cluster 1',
                                         'VolClust2':'Volume Cluster 2'},
                                        axis=1, inplace=True)
          survInfoCRTdescriptive.to_csv('./Results/kmeans/kmeans'+
                                        'ClustersDescriptive.csv')
          survInfoCRTdescriptive
```

```
          File "<ipython-input-106-49d9cc600460>", line 3
        rng = [(f'({survInfoCRTdescriptive.loc[:,column].loc[mn]:.2f}-
                                                                        ^
    SyntaxError: EOL while scanning string literal
```

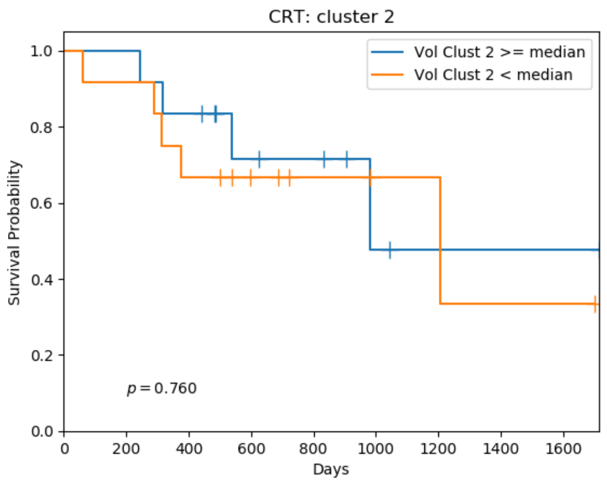
1.1 Creating Kaplan-Meier estimates and performing log-rank tests

```
In [107]: clust1MoreMedCRT = (survInfoCRT['VolClust1'] >=
                              np.median(survInfoCRT['VolClust1']))
          clust2MoreMedCRT = (survInfoCRT['VolClust2'] >=
                              np.median(survInfoCRT['VolClust2']))
          clust1MoreMedNoCRT = (survInfoNoCRT['VolClust1'] >=
                                np.median(survInfoNoCRT['VolClust1']))
          clust2MoreMedNoCRT = (survInfoNoCRT['VolClust2'] >=
                                np.median(survInfoNoCRT['VolClust2']))
          TCRT = survInfoCRT['timeFirstEventOrRCens']
          TNoCRT = survInfoNoCRT['timeFirstEventOrRCens']
          ECRT = survInfoCRT['status']
          ENoCRT = survInfoNoCRT['status']

          # fitting Kaplan-Meier curves
          res = logrank_test(TCRT[clust1MoreMedCRT], TCRT[~clust1MoreMedCRT],
                             ECRT[clust1MoreMedCRT], ECRT[~clust1MoreMedCRT])
          print(f'Difference in survival at a p-value of {res.p_value}')

          plt.figure()
          ax = plt.subplot(111)
          kmf = KaplanMeierFitter()
          kmf.fit(TCRT[clust1MoreMedCRT], ECRT[clust1MoreMedCRT],
                  label='Vol Clust 1 >= median')
          kmf.plot(ax=ax, ci_show=False, show_censors = True)
          kmf.fit(TCRT[~clust1MoreMedCRT], ECRT[~clust1MoreMedCRT],
                  label='Vol Clust 1 < median')
          kmf.plot(ax=ax, ci_show=False, show_censors = True)
          plt.title('CRT: cluster 1')
          plt.xlabel('Days')
          plt.ylabel('Survival Probability')
          plt.text(200,0.1,r'$p={:.3f}$'.format(res.p_value))
          plt.ylim([0, 1.05])
          plt.savefig(path.join(survivalCurvesFigurePath,
                                'CRT-Cluster1.pdf'))
          plt.show()
```

```
Difference in survival at a p-value of 0.8793394784672972
```

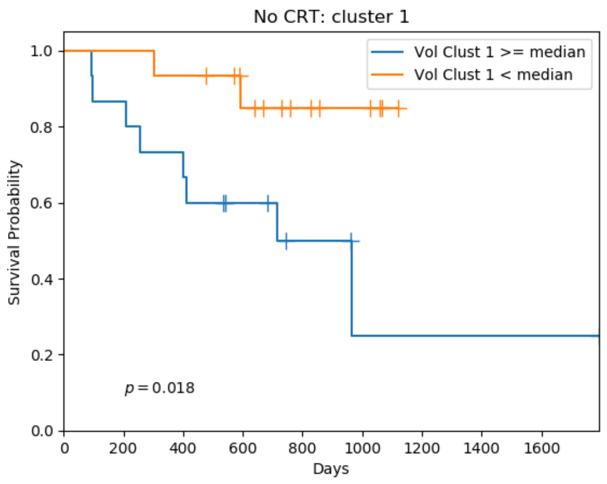


```
In [108]: # fitting Kaplan-Meier curves
          res = logrank_test(TCRT[clust2MoreMedCRT], TCRT[~clust2MoreMedCRT],
                             ECRT[clust2MoreMedCRT], ECRT[~clust2MoreMedCRT])
          print(f'Difference in survival at a p-value of {res.p_value}')
          plt.figure()
          ax = plt.subplot(111)
          kmf = KaplanMeierFitter()
          kmf.fit(TCRT[clust2MoreMedCRT], ECRT[clust2MoreMedCRT],
                  label='Vol Clust 2 >= median')
          kmf.plot(ax=ax, ci_show=False, show_censors = True)
          kmf.fit(TCRT[~clust2MoreMedCRT], ECRT[~clust2MoreMedCRT],
                  label='Vol Clust 2 < median')
          kmf.plot(ax=ax, ci_show=False, show_censors = True)
          plt.title('CRT: cluster 2')
          plt.xlabel('Days')
          plt.ylabel('Survival Probability')
          plt.text(200,0.1,r'$p={:.3f}$'.format(res.p_value))
          plt.ylim([0, 1.05])
          plt.savefig(path.join(survivalCurvesFigurePath,
                              'CRT-Cluster2.pdf'))
          plt.show()

Difference in survival at a p-value of 0.7598733335686022
```



```
In [109]: # fitting Kaplan-Meier curves
          res = logrank_test(TNoCRT[clust1MoreMedNoCRT], TNoCRT[~clust1MoreMedNoCRT],
                             ENoCRT[clust1MoreMedNoCRT], ENoCRT[~clust1MoreMedNoCRT])
          print(f'Difference in survival at a p-value of {res.p_value}')

          plt.figure()
          ax = plt.subplot(111)
          kmf = KaplanMeierFitter()
          kmf.fit(TNoCRT[clust1MoreMedNoCRT], ENoCRT[clust1MoreMedNoCRT],
                  label='Vol Clust 1 >= median')
          kmf.plot(ax=ax, ci_show=False, show_censors = True)
          kmf.fit(TNoCRT[~clust1MoreMedNoCRT], ENoCRT[~clust1MoreMedNoCRT],
                  label='Vol Clust 1 < median')
          kmf.plot(ax=ax, ci_show=False, show_censors = True)
          plt.title('No CRT: cluster 1')
          plt.xlabel('Days')
          plt.ylabel('Survival Probability')
          plt.text(200,0.1,r'$p={:.3f}$'.format(res.p_value))
          plt.ylim([0, 1.05])
          plt.savefig(path.join(survivalCurvesFigurePath, 'NoCRT-Cluster1.pdf'))
          plt.show()
```

```
Difference in survival at a p-value of 0.01839104261474097
```

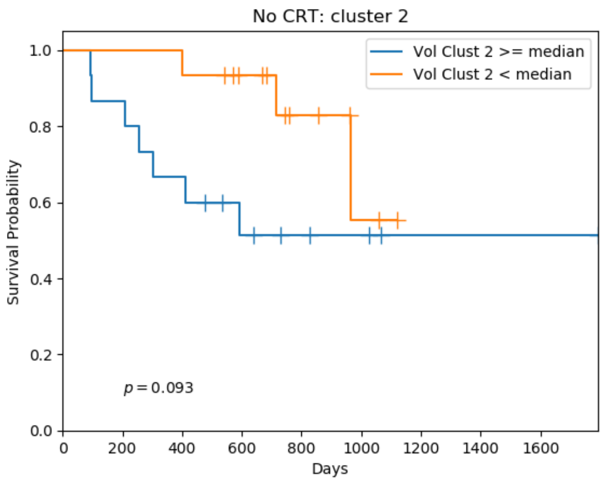


```
In [110]: # fitting Kaplan-Meier curves
          res = logrank_test(TNoCRT[clust2MoreMedNoCRT], TNoCRT[~clust2MoreMedNoCRT],
                             ENoCRT[clust2MoreMedNoCRT], ENoCRT[~clust2MoreMedNoCRT])
          print(f'Difference in survival at a p-value of {res.p_value}')

          plt.figure()
          ax = plt.subplot(111)
          kmf = KaplanMeierFitter()
          kmf.fit(TNoCRT[clust2MoreMedNoCRT], ENoCRT[clust2MoreMedNoCRT],
                  label='Vol Clust 2 >= median')
          kmf.plot(ax=ax, ci_show=False, show_censors = True)
          kmf.fit(TNoCRT[~clust2MoreMedNoCRT], ENoCRT[~clust2MoreMedNoCRT],
                  label='Vol Clust 2 < median')
          kmf.plot(ax=ax, ci_show=False, show_censors = True)
          plt.title('No CRT: cluster 2')
```

12

```
        plt.xlabel('Days')
        plt.ylabel('Survival Probability')
        plt.ylim([0, 1.05])
        plt.text(200,0.1,r'$p={:.3f}$'.format(res.p_value))
        plt.savefig(path.join(survivalCurvesFigurePath, 'NoCRT-Cluster2.pdf'))
        plt.show()
```

```
Difference in survival at a p-value of 0.09312688592673207
```



Chekking proportional hazard assumptions

```
In [111]: from lifelines import CoxPHFitter
          cph = CoxPHFitter()
          cphDF = pd.DataFrame({'Time': TCRT, 'Event': ECRT, 'Group1': clust1MoreMedCRT})
          cph.fit(cphDF, 'Time', 'Event')
          cph.check_assumptions(cphDF,p_value_threshold=0.05, advice=True)
          cphDF = pd.DataFrame({'Time': TCRT, 'Event': ECRT, 'Group2': clust2MoreMedCRT})
          cph.fit(cphDF, 'Time', 'Event')
          cph.check_assumptions(cphDF,p_value_threshold=0.05, advice=True)
```

```
Proportional hazard assumption looks okay.
Proportional hazard assumption looks okay.
```

13

```
In [112]: from lifelines import CoxPHFitter
          cph = CoxPHFitter()
          cphDF = pd.DataFrame({'Time': TNoCRT, 'Event': ENoCRT, 'Group1': clust1MoreMedNoCRT})
          cph.fit(cphDF, 'Time', 'Event')
          cph.check_assumptions(cphDF,p_value_threshold=0.05, advice=True)
          cphDF = pd.DataFrame({'Time': TNoCRT, 'Event': ENoCRT, 'Group2': clust2MoreMedNoCRT})
          cph.fit(cphDF, 'Time', 'Event')
          cph.check_assumptions(cphDF,p_value_threshold=0.05, advice=True)
```

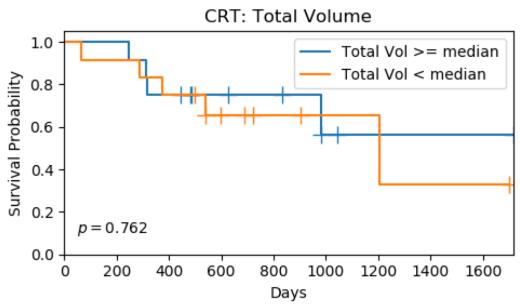
```
Proportional hazard assumption looks okay.
The ``p_value_threshold`` is set at 0.05. Even under the null hypothesis of no violations, some
covariates will be below the threshold by chance. This is compounded when there are many covariates.
Similarly, when there are lots of observations, even minor deviances from the proportional hazard
assumption will be flagged.

With that in mind, it's best to use a combination of statistical tests and visual tests to determine
the most serious violations. Produce visual plots using ``check_assumptions(..., show_plots=True)``
and looking for non-constant lines. See link [A] below for a full example.

<lifelines.StatisticalResult>
         test_name = proportional_hazard_test
 null_distribution = chi squared
degrees_of_freedom = 1

---
             test_statistic    p  -log2(p)
Group2 km              4.33 0.04      4.74
       rank            3.63 0.06      4.14


1. Variable 'Group2' failed the non-proportional test: p-value is 0.0375.

   Advice: with so few unique values (only 2), you can include `strata=['Group2', ...]` in the call
in `.fit`. See documentation in link [E] below.

---
[A]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assumption.htm
[B]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assumption.htm
[C]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assumption.htm
[D]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assumption.htm
[E]  https://lifelines.readthedocs.io/en/latest/jupyter_notebooks/Proportional%20hazard%20assumption.htm
```

Checking the difference in PFS due to the total volume

```
In [113]: survInfoCRT = survInfoCRT.merge(voxelDF.loc[:,'totVolume'],
                                          left_index=True, right_index=True)
          survInfoNoCRT = survInfoNoCRT.merge(voxelDF.loc[:,'totVolume'],
                                              left_index=True, right_index=True)

          totVolMoreMedCRT = (survInfoCRT['totVolume'] >=
                              np.median(survInfoCRT['totVolume']))
          totVolMoreMedNoCRT = (survInfoNoCRT['totVolume'] >=
                                np.median(survInfoNoCRT['totVolume']))
```

```
In [114]: # fitting Kaplan-Meier curves
          res = logrank_test(TCRT[totVolMoreMedCRT], TCRT[~totVolMoreMedCRT],
                             ECRT[totVolMoreMedCRT], ECRT[~totVolMoreMedCRT])
          print(f'Difference in survival at a p-value of {res.p_value}')

          plt.figure(figsize=(5,3))
          ax = plt.subplot(111)
          kmf = KaplanMeierFitter()
          kmf.fit(TCRT[totVolMoreMedCRT], ECRT[totVolMoreMedCRT], label='Total Vol >= median')
          kmf.plot(ax=ax, ci_show=False, show_censors = True)
          kmf.fit(TCRT[~totVolMoreMedCRT], ECRT[~totVolMoreMedCRT], label='Total Vol < median')
          kmf.plot(ax=ax, ci_show=False, show_censors = True)
          plt.title('CRT: Total Volume')
          plt.xlabel('Days')
          plt.ylabel('Survival Probability')
          plt.text(50,0.1,r'$p={:.3f}$'.format(res.p_value))
          plt.ylim([0, 1.05])
          plt.tight_layout()
          plt.savefig(path.join(survivalCurvesFigurePath, 'CRT-TotalVolume.pdf'))
          plt.show()
```

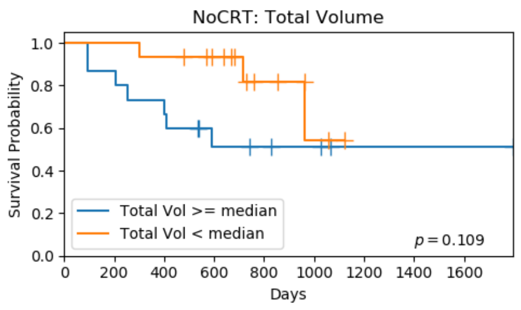
```
Difference in survival at a p-value of 0.7620702361816797
```



```
In [115]: # fitting Kaplan-Meier curves
          res = logrank_test(TNoCRT[totVolMoreMedNoCRT], TNoCRT[~totVolMoreMedNoCRT],
                             ENoCRT[totVolMoreMedNoCRT],
                            ENoCRT[~totVolMoreMedNoCRT])
          print(f'Difference in survival at a p-value of {res.p_value}')

          plt.figure(figsize=(5,3))
          ax = plt.subplot(111)
          kmf = KaplanMeierFitter()
```

```
kmf.fit(TNoCRT[totVolMoreMedNoCRT], ENoCRT[totVolMoreMedNoCRT],
        label='Total Vol >= median')
kmf.plot(ax=ax, ci_show=False, show_censors = True)
kmf.fit(TNoCRT[~totVolMoreMedNoCRT], ENoCRT[~totVolMoreMedNoCRT],
        label='Total Vol < median')
kmf.plot(ax=ax, ci_show=False, show_censors = True)
plt.title('NoCRT: Total Volume')
plt.xlabel('Days')
plt.ylabel('Survival Probability')
plt.text(1400,0.05,r'$p={:.3f}$'.format(res.p_value))
plt.ylim([0, 1.05])
plt.tight_layout()
plt.savefig(path.join(survivalCurvesFigurePath,
                      'NoCRT-TotalVolume.pdf'))
plt.show()
```

```
Difference in survival at a p-value of 0.10944661460250339
```



1.2 Univariate Cox model using the cluster volumes

```
In [116]: from lifelines import CoxPHFitter
          from IPython.core.display import display, HTML
```

1.2.1 No CRT

```
In [ ]: DFNOCRT1 = pd.DataFrame({'Duration': TNoCRT, 'Event':ENoCRT,
                                 'VolClust1':survInfoNoCRT.VolClust1/1000})
        cph1NoCRT = CoxPHFitter()
        cph1NoCRT.fit(DFNOCRT1, duration_col='Duration',
                      event_col='Event', show_progress=True,step_size=0.01)
        cph1NoCRT.print_summary(decimals=8)
        cph1NoCRT.check_assumptions(DFNOCRT1)
        cph1NoCRT.score_ # This is the c-index = the area under the ROC curve
```

```
In [118]: pd.options.display.float_format = '{:.8f}'.format
          cph1NoCRT.summary
```

```
Out[118]:                 coef  exp(coef)   se(coef)          z          p   -log2(p)  \
          VolClust1 0.10477057 1.11045580 0.03791947 2.76297559 0.00572771 7.44782709

                     lower 0.95  upper 0.95
          VolClust1  0.03044977  0.17909136
```

```
In [ ]: DFNOCRT2 = pd.DataFrame({'Duration': TNoCRT, 'Event':ENoCRT,
                                 'VolClust2':survInfoNoCRT.VolClust2/1000})
        cph2NoCRT = CoxPHFitter()
        cph2NoCRT.fit(DFNOCRT2, duration_col='Duration',
                      event_col='Event', show_progress=True)
        cph2NoCRT.print_summary(decimals=8)
        cph2NoCRT.score_
```

```
In [120]: pd.options.display.float_format = '{:.8f}'.format
          cph2NoCRT.summary.append(
              cph1NoCRT.summary).sort_index().to_csv(
              './Results/kmeans/UniCoxNoCRT.csv')
          cphUniNoCRT = cph2NoCRT.summary.append(cph1NoCRT.summary
                                              ).sort_index()
```

1.3 CRT

```
In [ ]: DFCRT1 = pd.DataFrame({'Duration': TCRT, 'Event':ECRT,
                               'VolClust1':survInfoCRT.VolClust1/1000})
        cph1CRT = CoxPHFitter()
        cph1CRT.fit(DFCRT1, duration_col='Duration', event_col='Event',
                    show_progress=True,step_size=0.01)
        cph1CRT.print_summary(decimals=8)
        cph1CRT.score_
```

```
In [122]: cph1CRT.summary
```

```
Out[122]:                 coef  exp(coef)   se(coef)          z          p   -log2(p)  \
          VolClust1 0.00374741 1.00375444 0.03808745 0.09838966 0.92162289 0.11775155

                     lower 0.95  upper 0.95
          VolClust1 -0.07090262  0.07839744
```

```
In [ ]: DFCRT2 = pd.DataFrame({'Duration': TCRT, 'Event':ECRT,
                               'VolClust2':survInfoCRT.VolClust2/1000})
        cph2CRT = CoxPHFitter()
        cph2CRT.fit(DFCRT2, duration_col='Duration', event_col='Event',
                    show_progress=True, step_size=0.01)
        cph2CRT.print_summary(decimals=8)
        cph2CRT.score_
```

```
In [124]: cph2CRT.summary.append(cph1CRT.summary).sort_index(
          ).to_csv('./Results/kmeans/UniCoxCRT.csv')
          cphUniCRT = cph2CRT.summary.append(cph1CRT.summary
                                          ).sort_index()
```

1.3.1 Univariate cox summary

```
In [125]: cphCRTsum = cphUniCRT.append(cphUniNoCRT).reset_index()
          cphCRTsum.rename({'index':'Risk Factor'}, axis=1, inplace=True)
          cphCRTsum['treatment'] = ['CRT', 'CRT', 'No CRT', 'No CRT']

          cphCRTsum.set_index(['treatment', 'Risk Factor'], inplace=True)
          cphCRTsum
```

```
Out[125]:                              coef  exp(coef)   se(coef)           z  \
          treatment Risk Factor
          CRT       VolClust1    0.00374741 1.00375444 0.03808745  0.09838966
                    VolClust2   -0.04701540 0.95407270 0.03772286 -1.24633718
          No CRT    VolClust1    0.10477057 1.11045580 0.03791947  2.76297559
                    VolClust2    0.02795032 1.02834460 0.01829582  1.52768896

                                          p   -log2(p)  lower 0.95  upper 0.95
          treatment Risk Factor
          CRT       VolClust1   0.92162289 0.11775155 -0.07090262  0.07839744
                    VolClust2   0.21264063 2.23351078 -0.12095085  0.02692004
          No CRT    VolClust1   0.00572771 7.44782709  0.03044977  0.17909136
                    VolClust2   0.12658978 2.98176712 -0.00790883  0.06380947
```

```
In [126]: # convert to latex friendly table
          cphCRTsumLatex = cphCRTsum.drop(['se(coef)','z', '-log2(p)'], axis=1)
          cphCRTsumLatex.rename({'coef':'$\\beta$',
                                'exp(coef)':'$e^\\beta$'}, inplace=True, axis = 1)
```

```
In [127]: from IPython.lib.deepreload import reload
          %load_ext autoreload
          %autoreload 2
          from latex_table_utils import merge_columns_by_name, build_latex_table
          help(merge_columns_by_name)
          help(build_latex_table)
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
Help on function merge_columns_by_name in module latex_table_utils:

merge_columns_by_name(df, name_merged, col_name_low, col_name_high, sigDig)

Help on function build_latex_table in module latex_table_utils:

build_latex_table(dataIn, col_names, row_names, filename, sigDig)
```

```
In [128]: cphCRTsumLatex = merge_columns_by_name(cphCRTsumLatex,
                          '$95 \\%$ CI' ,'lower 0.95', 'upper 0.95', 3)
```

```
In [129]: build_latex_table(cphCRTsumLatex, 1, 1,
          './Results/kmeans/tables/UnivariateCox.txt', 3)
```

```
Index(['treatment', 'Risk Factor', '$\beta$', '$e^\beta$', 'p', '$95 \%$ CI'], dtype='object')
```

```
Out[129]:                        $\beta$ $e^\beta$        p           $95 \%$ CI
          treatment Risk Factor
          CRT       VolClust1    0.00375         1    0.922   (-0.0709 - 0.0784)
                    VolClust2     -0.047     0.954    0.213    (-0.121 - 0.0269)
          No CRT    VolClust1      0.105      1.11  0.00573     (0.0304 - 0.179)
                    VolClust2      0.028      1.03    0.127  (-0.00791 - 0.0638)
```

Checking proportional hazard assumptions

```
In [130]: print(cph1CRT.check_assumptions(DFCRT1))
          print(cph2CRT.check_assumptions(DFCRT2))
          print(cph1NoCRT.check_assumptions(DFNOCRT1))
          print(cph2NoCRT.check_assumptions(DFNOCRT2))

Proportional hazard assumption looks okay.
None
Proportional hazard assumption looks okay.
None
Proportional hazard assumption looks okay.
None
Proportional hazard assumption looks okay.
None
```

Also including the total volume in the univariate cox analysis

```
In [ ]: DFTotVolCRT = pd.DataFrame({'Duration': TCRT, 'Event':ECRT,
                               'totVol':survInfoCRT.totVolume/1000})
        cphTotVolCRT = CoxPHFitter()
        cphTotVolCRT.fit(DFTotVolCRT, duration_col='Duration',
            event_col='Event', show_progress=True, step_size=0.01)
        cphTotVolCRT.print_summary(decimals=8)
        cphTotVolCRT.score_
        cphTotVolCRT.check_assumptions(DFTotVolCRT)

In [ ]: DFTotVolNoCRT = pd.DataFrame({'Duration': TNoCRT, 'Event':ENoCRT,
                               'totVol':survInfoNoCRT.totVolume/1000})
        cphTotVolNoCRT = CoxPHFitter()
        cphTotVolNoCRT.fit(DFTotVolNoCRT, duration_col='Duration',
                event_col='Event', show_progress=True, step_size=0.01)
        cphTotVolNoCRT.print_summary(decimals=8)
        cphTotVolNoCRT.score_
        cphTotVolNoCRT.check_assumptions(DFTotVolNoCRT)
```

Building a latex table for the thesis

```
In [133]: totVolSummary = cphTotVolCRT.summary.append(cphTotVolNoCRT.summary)
          totVolSummary.reset_index(inplace=True)
          totVolSummary['Treatment'] = ['CRT', 'No CRT']
          totVolSummary.rename({'index':'Risk Factor'},
                               inplace=True, axis=1)
          totVolSummary.set_index(['Treatment', 'Risk Factor'],
                                  inplace=True)
          # convert to latex friendly table
          totVolSummary = totVolSummary.drop(['se(coef)',
                                        'z', '-log2(p)'], axis=1)
          totVolSummary.rename({'coef':'$\\beta$',
                  'exp(coef)':'$e^\\beta$'}, inplace=True, axis = 1)
          totVolSummary = merge_columns_by_name(totVolSummary,
                          '$95 \\%$ CI' ,'lower 0.95', 'upper 0.95', 3)

In [134]: uniCoxSummary = cphCRTsumLatex.append(totVolSummary).sort_index()
          uniCoxSummary.rename({'$95 \\%$ CI':r'95\% CI ($\beta$)',
                                'p':'$p$'}, axis = 1, inplace = True)
          unit = r'[$\SI{}{\centi\meter\cubed}$]'
          uniCoxSummary.reset_index(inplace=True)
          uniCoxSummary['Risk Factor'] = [f'{rf} {unit}'
                                  for rf in uniCoxSummary['Risk Factor']]
          uniCoxSummary.rename({'treatment':'Treatment'},
                               axis=1, inplace=True)
          uniCoxSummary.set_index(['Treatment',
                                   'Risk Factor'], inplace=True)
          build_latex_table(uniCoxSummary, 1, 1,
                            './Results/kmeans/tables/UnivariateCoxWTot.txt', 3)

Index(['Treatment', 'Risk Factor', '$\beta$', '$e^\beta$', '$p$',
       '95\% CI ($\beta$)'],
      dtype='object')


Out[134]:                                                   $\beta$ $e^\beta$      $p$  \
          Treatment Risk Factor
          CRT       VolClust1 [$\SI{}{\centi\meter\cubed}$]  0.00375         1    0.922
                    VolClust2 [$\SI{}{\centi\meter\cubed}$]   -0.047     0.954    0.213
                    totVol [$\SI{}{\centi\meter\cubed}$]     -0.0208     0.979    0.364
          No CRT    VolClust1 [$\SI{}{\centi\meter\cubed}$]    0.105      1.11  0.00573
                    VolClust2 [$\SI{}{\centi\meter\cubed}$]    0.028      1.03    0.127
                    totVol [$\SI{}{\centi\meter\cubed}$]      0.0317      1.03   0.0166

                                                             95\% CI ($\beta$)
          Treatment Risk Factor
          CRT       VolClust1 [$\SI{}{\centi\meter\cubed}$]   (-0.0709 - 0.0784)
                    VolClust2 [$\SI{}{\centi\meter\cubed}$]    (-0.121 - 0.0269)
                    totVol [$\SI{}{\centi\meter\cubed}$]      (-0.0656 - 0.0241)
          No CRT    VolClust1 [$\SI{}{\centi\meter\cubed}$]     (0.0304 - 0.179)
                    VolClust2 [$\SI{}{\centi\meter\cubed}$]  (-0.00791 - 0.0638)
                    totVol [$\SI{}{\centi\meter\cubed}$]      (0.00575 - 0.0576)
```

A.4.7 Gaussian Mixture Model survival analysis

1 Gaussian Mixture Models - Results and Survival Analysis

```
In [ ]: %matplotlib inline
        import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        from lifelines import KaplanMeierFitter, CoxPHFitter
        from lifelines.statistics import logrank_test
        import os
        from joblib import Parallel, delayed
        from IPython.lib.deepreload import reload
        %load_ext autoreload
        %autoreload 2
        from latex_table_utils import merge_columns_by_name, build_latex_table
```

1.1 Calculating the BIC and the gradient BIC

```
In [ ]: modsK1to50 = np.load('./Results/GMM/mixModels300000samples5reps'+
                             '3inits1kmin50kMaxNoInteractions.npy')
        modsK51to70 = np.load('./Results/GMM/mixModels300000samples5reps'+
                              '3inits51kmin70kMaxNoInteractions.npy')
        allMods = np.concatenate((modsK1to50,modsK51to70), axis=0)

In [ ]: dataset = pd.read_feather('./Data/featuresPCAnoInteractions')
        datasetRed = dataset.iloc[:,0:4]

In [ ]: kList = np.arange(1,71)
        reps = 5
        seeds = np.tile(np.array([r for r in range(reps)]),len(kList))
        N = 300000 # number of samples

In [ ]: def calculateBIC(mdl, X):
            return(mdl.bic(X))

In [ ]: BICScore = Parallel(n_jobs=-1,verbose=10)(delayed(
            calculateBIC)(mdl, datasetRed.sample(N, random_state=r))
                                 for mdl, r in zip(allMods, seeds))

In [ ]: np.save('./Results/GMM/BICscorek1to70', BICScore)

In [ ]: BICScore = np.load('./Results/GMM/BICscorek1to70.npy')

In [ ]: BICforGrad = np.array(
        [np.array(BICScore).reshape(len(kList),
                           reps)[:,r] for r in range(reps)])
        xForGrad = np.array(
        [np.repeat(kList,reps).reshape(len(kList),
                            reps)[:,r] for r in range(reps)])

In [ ]: from matplotlib.ticker import FormatStrFormatter
        import matplotlib as mpl

        plt.figure(figsize=(8,4))
```

```
ax1 = plt.subplot(212)
ax1.yaxis.set_major_formatter(FormatStrFormatter('%.1e'))
sns.lineplot(x=xForGrad.reshape(-1), y=np.gradient(BICforGrad,
                axis=1).reshape(-1), ci='sd', err_style='bars')
plt.xlabel('k')
plt.xlim([0, 71])
plt.axhline(0, ls='--', lw = 0.5, c='k')
plt.axvline(9, ls='--', lw=0.5, c='k')
plt.xticks(kList[0:-1:4])
plt.ylabel('Gradient BIC score')
plt.title('Gradient BIC(k) on GMM clusters')
# plt.yticks([])
ax2 = plt.subplot(211, sharex=ax1)
ax2.yaxis.set_major_formatter(FormatStrFormatter('%.1e'))
sns.lineplot(x=np.repeat(kList,reps),
             y=BICScore, ci='sd', err_style='bars')
plt.xlabel('k')
plt.ylabel('BIC Score')
plt.xlim([0, 71])
# plt.yticks([])
plt.title('BIC(k) on GMM clusters')
plt.xticks(kList[0:-1:4])
plt.tight_layout()
plt.savefig('./figures/GMMBICandGradBICAllK.pdf')
```

Local minima for 9 components and the change in the BIC for adding additional components exceeding the

1.2 Analysing the model with the selected number of components ($n_c = 9$)

Firstly we extract all the models made with the wanted number of components. From the plots above it was found that we had a local minimum for 9 compnonents. After this we can see that the gradient of the BIC flattes indicating the ther performance increase by adding additional components is small. It is worth pointing out that the BIC will converge to the actual model when $N \to \infty$. However, since it is in no way ceratin that the components are actually gaussian, it may use many components to approximate a non-gaussian probability distribution function. Thus we will settle for a "resonable" number of components.

```
In [ ]: mixMods_k9 = [mdl for mdl in allMods if mdl.n_components == 9]

In [ ]: BICscore_k9 = [mixMod.bic(datasetRed.sample(N,random_state=seed)) for mixMod,
                      seed in zip(mixMods_k9, np.arange(0,reps))]

In [ ]: bestMod_k9 = mixMods_k9[np.argmin(BICscore_k9)]
        testMod_k9 = (mixMods_k9[:np.argmin(BICscore_k9)] +
                      mixMods_k9[np.argmin(BICscore_k9)+1:])
        print(np.argmin(BICscore_k9))

In [ ]: means = [mod.means_ for i,mod in enumerate(mixMods_k9)]
        mdls = [f'mdl{i+1}' for i in range(len(means))]
        # finding the closest components in the data
        def closest_node(node, nodes):
            nodes = np.asarray(nodes)
```

2

```
            deltas = nodes - node
            dist_2 = np.einsum('ij,ij->i', deltas, deltas)
            return(np.argmin(dist_2), dist_2[np.argmin(dist_2)])
        refMod = means[np.argmin(BICscore_k9)]
        refModName = mdls[np.argmin(BICscore_k9)]
        del mdls[np.argmin(BICscore_k9)]
        del means[np.argmin(BICscore_k9)]
        supComps = []
        supDists = []
        for i,mean in enumerate(means):
            tempMean = mean
            dists = np.zeros_like(means[0][:,0])
            comps = np.zeros_like(means[0][:,0], dtype=int)
            for k in range(refMod.shape[0]):
                comps[k], dists[k] = closest_node(refMod[k], tempMean)
                np.delete(tempMean, comps[k], 0)
            supComps.append(comps)
            supDists.append(dists)
```

We have now selected the best perfoming of the models with 9 components in terms of the BIC score. And we'll continue the analysis using this model.

Predicting the labels of the dataset usign the selected model and finding the corresponding components in the repeat GMM solutions using the JS distance.

```
In [ ]: labels_k9 = bestMod_k9.predict(datasetRed) + 1
```

```
In [ ]: labelsAllMods = pd.DataFrame(labels_k9, columns=['refModel'])
        labelsAllMods = labelsAllMods.merge(
            pd.DataFrame({mdls:mixMod.predict(datasetRed)+1 for mdls,
                    mixMod in zip(mdls, testMod_k9)}), left_index=True,
                        right_index=True)
```

```
In [ ]: probRef = bestMod_k9.predict_proba(datasetRed)
        probTest = mixMods_k9[0].predict_proba(datasetRed)
```

```
In [ ]: from scipy.spatial.distance import jensenshannon

        jsDist = {}
        for mdl,mdlName in zip(testMod_k9,mdls):
            print(mdlName)
            probTest = mdl.predict_proba(datasetRed)
            jsDistTemp = np.zeros((9,9))
            for refColNum in range(probRef.shape[1]):
                for compNum in range(probTest.shape[1]):
                    jsDistTemp[refColNum,compNum]=jensenshannon(
                        probRef[:,refColNum], probTest[:,compNum])
            jsDist[mdlName] = jsDistTemp
```

```
In [ ]: maps = {}
        for mdlName in mdls:
            mdlMap = {}
            for column in range(jsDist[mdlName].shape[1]):
                mdlMap[column+1] = np.argmin((jsDist[mdlName])[:,column])+1
            maps[mdlName] = mdlMap
```

```
In [ ]: labelsAllModsCorr = labelsAllMods.copy()
        for mdlMap in maps:
            print(mdlMap)
            labelsAllModsCorr[mdlMap].replace(maps[mdlMap], inplace=True)
```

1.2.1 Calculating the correspodence in labeling between the corresponding components of the GMM clusterin solutions

```
In [ ]: from sklearn.metrics import cohen_kappa_score
        kappaScores ={mdlName:cohen_kappa_score(
            labelsAllModsCorr.refModel,
            labelsAllModsCorr[mdlName]) for mdlName in mdls}
        kappaScores
```

```
In [ ]: from sklearn.metrics import jaccard_similarity_score
        jaccardScores ={mdlName:jaccard_similarity_score(
            labelsAllModsCorr.refModel,
            labelsAllModsCorr[mdlName]) for mdlName in mdls}
        jaccardScores
```

```
In [ ]: from sklearn.metrics import confusion_matrix
        confMatrices ={mdlName:confusion_matrix(
            labelsAllModsCorr.refModel,
            labelsAllModsCorr[mdlName]) for mdlName in mdls}
```

```
In [ ]: from sklearn.metrics import accuracy_score, balanced_accuracy_score
        accScore ={mdlName:accuracy_score(
            labelsAllModsCorr.refModel,
            labelsAllModsCorr[mdlName]) for mdlName in mdls}
        balAccScore ={mdlName:balanced_accuracy_score(
            labelsAllModsCorr.refModel,
            labelsAllModsCorr[mdlName]) for mdlName in mdls}
        print(accScore, balAccScore)
```

```
In [ ]: kappa = pd.DataFrame(kappaScores, index=['$\kappa$'])
        j = pd.DataFrame(jaccardScores, index=['$\mathcal J$'])
        ba = pd.DataFrame(balAccScore, index=['BAS'])
        agreement = kappa.append(j).append(ba)
        agreement.columns = [f'Model {i+1}' for i in range(4)]
        agreement['Average'] = agreement.sum(axis=1)/4
        agreement['$\sigma$'] = agreement.iloc[:,0:3].std(axis=1)
        agreement
```

```
In [ ]: %load_ext autoreload
        %autoreload 2
        from latex_table_utils import merge_columns_by_name, build_latex_table
        help(build_latex_table)
        build_latex_table(agreement, 1, 1,
                          './Results/GMM/tables/repeatability.txt', 3)
```

1.3 Component Distribution Histogram

Making a histogram of the component distribution to see that no component is unreasonably small or large.

```
In [ ]: # Histogram
        plt.figure()
        plt.title('Histogram')
        sns.distplot(labels_k9, kde=False, bins=np.arange(0.5,10,1))
        plt.xlabel('Component')
        plt.ylabel('Number of voxels')
        plt.savefig('./figures/histograms/ComponentsDistGMMk9.pdf')
        plt.show()

In [ ]: componentDist = pd.DataFrame(pd.Series(labels_k9).value_counts().sort_index()).T
```

1.4 Histogram of the groups on the original features

```
In [ ]: datasetRaw = pd.read_feather('./Data/featureDFRawZScaled')
        datasetCluster = datasetRaw.copy()
        datasetCluster['cluster'] = labels_k9
        datasetCluster = datasetCluster.sample(300000)
        datasetCluster = pd.melt(datasetCluster, id_vars=['cluster'],
                    var_name='ImageSeq', value_name='pixelIntensity')
        datasetCluster['cluster'] = datasetCluster['cluster']

In [ ]: g = sns.FacetGrid(datasetCluster, col='ImageSeq',
                          hue='cluster', col_wrap=3,
                          sharex=False, sharey=False,
                          legend_out = True)
        g = g.map(sns.distplot, 'pixelIntensity', kde = False)

        axes = g.axes.flatten()

        # Creating titles for the plot
        bVals = [0, 25, 50, 100, 500, 1000]
        DWItitles = [r'DWI - $b = %d \, s/mm^2$' % (bVal) for bVal in bVals]
        titles = ['T2'] + DWItitles + ['ADC', 'D', 'D*', 'f']
        g.add_legend(fontsize = 14)
        plt.setp(g._legend.get_title(), fontsize=14)
        for c,title in enumerate(titles):
            axes[c].set_title(title, fontsize = 16)
            axes[c].set_xticklabels(axes[c].get_xticks(),fontsize=12)
            axes[c].set_yticks([])
            if c in range(0,len(titles), 3):
                axes[c].set_ylabel('Frequency', fontsize=14)
        # setting x-label for the bottom plots
        for ax in axes[-3:]:
            ax.set_xlabel('z-scored voxel intensity', fontsize = 14)
        plt.savefig('./figures/histograms/'+
                    'featureHistogramWithClustersNoInteractionsGMMk9.pdf')
        plt.show()

In [ ]: def GMM1vAllHist(datasetCluster, k):
            datasetClusterk = datasetCluster.copy()
            datasetClusterk['cluster'] = [0  if sample != k else k
                                    for sample in datasetCluster['cluster']]
            g = sns.FacetGrid(datasetClusterk, col='ImageSeq',
                              hue='cluster', col_wrap=3,
                          sharex=False, sharey=False, legend_out = True)
            g = g.map(sns.distplot, 'pixelIntensity', kde = False)

            axes = g.axes.flatten()

            # Creating titles for the plot
            bVals = [0, 25, 50, 100, 500, 1000]
            DWItitles = [r'DWI - $b = %d \, s/mm^2$' % (bVal)
                         for bVal in bVals]
            titles = ['T2'] + DWItitles + ['ADC', 'D', 'D*', 'f']
            g.add_legend(fontsize = 14)
            plt.setp(g._legend.get_title(), fontsize=14)
            for c,title in enumerate(titles):
                axes[c].set_title(title, fontsize = 16)
                axes[c].set_xticklabels(axes[c].get_xticks(),fontsize=12)
                axes[c].set_yticks([])
                if c in range(0,len(titles), 3):
                    axes[c].set_ylabel('Frequency', fontsize=14)
            # setting x-label for the bottom plots
            for ax in axes[-3:]:
                ax.set_xlabel('z-scored voxel intensity', fontsize = 12)
            plt.savefig(f'./figures/histograms/GMM1vAll/k{k}.pdf')
            plt.close();
```

```
In [ ]: for k in np.arange(1,10):
            GMM1vAllHist(datasetCluster, k)
```

1.5 Survival Analysis

```
In [ ]: from scipy.io import loadmat
        from statsmodels.stats.multitest import multipletests
```

In the survival analysis we will make Kaplan-Meier curves, perform the log-rank tets on patient groups according to the if a given patient has below of above the median of a given cluster component and we'll perform univariate and multivariate Cox regression.

Importing the patient file names and a list of which voxels are belonging to which patient.

```
In [ ]: FileNames = loadmat('./../Data/FeatureArray/featureArraysFinalv2.mat',
                            variable_names = 'FileNames')
        try:
            FileNames=FileNames['FileNames']
        except:
            pass
        FileNames = FileNames.flatten()
        for c,FileName in enumerate(FileNames):
            FileNames[c]=FileName[0][:-4]

        pasInd=loadmat('./../Data/FeatureArray/featureArraysFinalv2.mat',
                       variable_names = 'grpsPasInd')
        pasInd=pasInd['grpsPasInd']-1 #ptyhon is zero indexed while matlab is not
        print(pasInd.shape, FileNames.shape)

        dicomInfoTable=pd.read_csv('./../Data/dicomInfo/dicomInfoTable.csv')
        dicomInfoTable.set_index('Row', inplace=True)
        dicomInfoTable.index = [patName.lower()
                                for patName in dicomInfoTable.index]
        dicomInfoTable.index.names = ['PatientName']
        voxelDF = pd.DataFrame(np.concatenate(
            (FileNames.reshape(-1,1),pasInd),axis=1))
        voxelDF.set_index(0,inplace=True)
        voxelDF.index = [patName.lower() for patName in voxelDF.index]
        voxelDF.index.names = ['PatientName']
        voxelDF.columns = ['voxIndStart', 'voxIndEnd']
        voxelDF['totNumVox'] = voxelDF.voxIndEnd - voxelDF.voxIndStart +1
        voxelDF.head()

        labels_k9_DF = pd.DataFrame(labels_k9)
        valCounts = [pd.value_counts(
            labels_k9_DF.iloc[start:end,0]) for start,
            end in zip(voxelDF.loc[:,'voxIndStart'].values,
            voxelDF.loc[:,'voxIndEnd'].values)]

In [ ]: numpats = len(valCounts)
        numk = len(valCounts[0])
        grps = np.zeros((numpats,numk))
        for c, elem in enumerate(valCounts):
            for k in range(numk):
                try:
                    grps[c,k] = elem.loc[k+1]
                except:
                    pass

In [ ]: dicomInfoTable['voxelVolume'] = (dicomInfoTable['PixelSpacing_1']*
            dicomInfoTable['PixelSpacing_2']*dicomInfoTable['SliceThickness']
        dicomInfoTable.head()
        voxelDF = voxelDF.merge(dicomInfoTable['voxelVolume'],
                                how='inner', left_index=True,
                            right_index=True)

In [ ]: pd.options.display.float_format = '{:.2f}'.format
        for k in range(numk):
            voxelDF[f'VolumeComp{k+1}'] = grps[:,k]*voxelDF.voxelVolume
        voxelDF['totalVolume'] = voxelDF.loc[:,
                            'VolumeComp1':'VolumeComp9'].sum(axis=1)
```

Computing the mean, range and standard deviation of the components and building it into a table of descriptive statistics

```
In [ ]: statsDF = voxelDF.loc[:,'VolumeComp1':'totalVolume'].describe()

In [ ]: mx, mn = 'max', 'min'
        volRange = [f'({statsDF[column].loc[mn]:.3g},{statsDF[column].loc[mx]:.3g})'
                    for column in statsDF.columns]

In [ ]: from latex_table_utils import merge_columns_by_name, build_latex_table
        help(merge_columns_by_name)
        help(build_latex_table)
```

```
In [ ]: descriptiveTable = merge_columns_by_name(statsDF.T, 'range', 'min', 'max', 3)
        descriptiveTable.reset_index(inplace=True)
        descriptiveTable.rename({'index':'Volume of Component'}, axis=1, inplace=True)
        descriptiveTable['Volume of Component'] = [i for i in range(1,10)] + ['All']
        descriptiveTable['Treatment'] = ['']
```

```
In [ ]: descriptiveTable = pd.DataFrame({'mean':statsDF.loc['mean',:],
                              'standard deviation':statsDF.loc['std',:]
                               ,'median':statsDF.loc['50%', :],
                             'range':volRange})
```

```
In [ ]: descriptiveTable
```

1.6 KM estimate and logrank survival analysis

```
In [ ]: survInfoCRT = pd.read_feather('./Data/survInfoCRT')
        survInfoNoCRT = pd.read_feather('./Data/survInfoNoCRT')
        survInfoCRT.set_index('PatientName', inplace=True)
        survInfoNoCRT.set_index('PatientName', inplace=True)
        survInfoCRT = survInfoCRT.iloc[:,0:2]
        survInfoNoCRT = survInfoNoCRT.iloc[:,0:2]

        survInfoCRT = survInfoCRT.merge(voxelDF.loc[:,'VolumeComp1':'VolumeComp9'],
                                        left_index=True, right_index=True)
        survInfoNoCRT = survInfoNoCRT.merge(voxelDF.loc[:,'VolumeComp1':'VolumeComp9'],
                                            left_index=True, right_index=True)
```

```
In [ ]: def survDiffClusterMedian(volSeries, T, E):
            compMoreMed = (volSeries >= np.median(volSeries))
            res = logrank_test(T[compMoreMed], T[~compMoreMed],
                               E[compMoreMed], E[~compMoreMed])
            return res.p_value
```

```
In [ ]: TCRT = survInfoCRT.timeFirstEventOrRCens
        ECRT = survInfoCRT.status
        survDiffClusterMedian(survInfoCRT['VolumeComp3'],TCRT, ECRT)
        pvalCRT = []
        for k in range(numk):
            pvalCRT.append(survDiffClusterMedian(survInfoCRT[f'VolumeComp{k+1}'],
                                                 TCRT, ECRT))
        pvalCRT
```

```
In [ ]: TNoCRT = survInfoNoCRT.timeFirstEventOrRCens
        ENoCRT = survInfoNoCRT.status
        pvalNoCRT = []
        for k in range(numk):
            pvalNoCRT.append(survDiffClusterMedian(
                survInfoNoCRT[f'VolumeComp{k+1}'],TNoCRT, ENoCRT))
        pvalNoCRT
```

```
In [ ]: def survInfoToLatexTableFormat(survInfo, treatment):
            # Extract volume components
            tableDF = survInfo.loc[:,[f'VolumeComp{i}' for i in range(1,10)]]
            tableDF['All Components'] = tableDF.sum(axis=1)
```

```
    tableDF = tableDF.describe().T
    tableDF = merge_columns_by_name(tableDF, 'range', 'min', 'max', 3)
    tableDF = tableDF.loc[:,['mean', 'std', '50%', 'range']]
    tableDF.reset_index(inplace=True)
    tableDF['treatment'] = [treatment]*tableDF.shape[0]
    tableDF.loc[:,'index'] = [f'Component {i}'
                    for i in range(1, 10)] + ['All Components']
    tableDF.rename({'index':'Volume per patient', '50%':'median'},
                   axis=1, inplace = True)
    tableDF.set_index(['treatment', 'Volume per patient'],
                      inplace = True)
    return tableDF
```

```
In [ ]: survCRTtoLatex = survInfoToLatexTableFormat(survInfoCRT, 'CRT')
        survNoCRTtoLatex = survInfoToLatexTableFormat(survInfoNoCRT, 'No CRT')
        survCRTtoLatex.append(survNoCRTtoLatex)

        build_latex_table(survCRTtoLatex.append(survNoCRTtoLatex),
                          1, 1, './Results/GMM/tables/GMMDescriptive.txt', 3)
```

1.7 Bonferronicorrected p-values CRT

```
In [ ]: pd.options.display.float_format = '{:.5g}'.format
        pvalCRTBF=multipletests(pvalCRT, alpha=0.05,
                method='BonFerroni', is_sorted=False, returnsorted=False)
        corrAlpha = pvalCRTBF[3]
        pvalCRTBFDF = pd.DataFrame({'pvals':pvalCRTBF[1], 'significant':pvalCRTBF[0]}).T
        pvalCRTBFDF.columns = [f'Component {k}' for k in range(1,10)]
        pvalCRTBFDF.loc['$p$',:] = [f'{pval:.4f}' if pval > corrAlpha else f'{pval:.4f}*'
                                    for pval in pvalCRT]
        pvalCRTBFDF
```

1.8 Bonferronicorrected p-values No CRT

```
In [ ]: pd.options.display.float_format = '{:.5g}'.format
        pvalNoCRTBF=multipletests(pvalNoCRT, alpha=0.05, method='ho',
                                  is_sorted=False, returnsorted=False)
        corrAlpha = pvalNoCRTBF[3]
        pvalNoCRTBFDF = pd.DataFrame({'pvals_uncorr':pvalNoCRT,'pvals':pvalNoCRTBF[1],
                                     'significant':pvalNoCRTBF[0]}).T
        pvalNoCRTBFDF.columns = [f'Component {k}' for k in range(1,10)]
        pvalNoCRTBFDF.loc['$p$',:] = [f'{pval:.4f}' if pval > corrAlpha else f'{pval:.4f}*'
                                    for pval in pvalNoCRTBFDF.loc['pvals_uncorr',:].values]
        pvalNoCRTBFDF
```

```
In [ ]: pvalNoCRTToLatex = pvalNoCRTBFDF.T.reset_index().loc[:,['$p$', 'index']]
        pvalNoCRTToLatex['treatment'] = ['NoCRT']*pvalNoCRTToLatex.shape[0]
        pvalNoCRTToLatex.rename({'index': 'component'}, axis=1, inplace = True)
        pvalNoCRTToLatex.set_index(['treatment', 'component'],inplace=True)
```

```
In [ ]: pvalCRTToLatex = pvalCRTBFDF.T.reset_index().loc[:,['$p$', 'index']]
        pvalCRTToLatex['treatment'] = ['CRT']*pvalCRTToLatex.shape[0]
        pvalCRTToLatex.rename({'index': 'component'}, axis=1, inplace = True)
        pvalCRTToLatex.set_index(['treatment', 'component'], inplace=True)
```

```
In [ ]: help(build_latex_table)
```

```
In [ ]: pvalBFToLatex = pvalCRTToLatex.append(pvalNoCRTToLatex)
        build_latex_table(pvalBFToLatex, 1,1, './Results/GMM/tables/pvalsLogRankBfCorrk9.txt', 3)
```

1.9 Plotting the Kaplan-Meier estimates for CRT and non-CRT

```
In [ ]: def survCurvePlot(k, T, E, survInfo, treatment, show):
            # input:
            # k - component number
            # T - durations/time
            # E - Event flag
            # Treatment - string to differentiate CRT from NoCRT when saving
            # show - boolean. If False saves the figure without showing it
            medFlag = survInfo[f'VolumeComp{k}'] >= np.median(survInfo[f'VolumeComp{k}'])

            res = logrank_test(T[medFlag], T[~medFlag], E[medFlag],
                              E[~medFlag])

            plt.figure()
            ax = plt.subplot(111)
            kmf_mm = KaplanMeierFitter()
            kmf_mm.fit(T[medFlag], E[medFlag], label='Vol >= med')
            kmf_mm.plot(ax=ax, ci_show=False, show_censors = True)
            kmf_lm = KaplanMeierFitter()
            kmf_lm.fit(T[~medFlag], E[~medFlag], label='Vol < med')
            kmf_lm.plot(ax=ax, ci_show=False, show_censors = True)
            plt.title(r'Component {:d}: $p={:.4f}$'.format(k,res.p_value))
            plt.xlabel('Days')
            plt.ylabel('Survival Probability')
            plt.ylim([0, 1.05])

            plt.savefig(f'./figures/GMMSurvivalFigures/GMMKMComp{k:d}{treatment}.pdf')
            # from lifelines.plotting import add_at_risk_counts
            # add_at_risk_counts(kmf_mm, kmf_lm, ax=ax)
            if show:
                plt.show()
            else:
                plt.close()
```

```
In [ ]: # testing:
        survCurvePlot(3, TNoCRT, ENoCRT, survInfoNoCRT, 'NoCRT', True)
```

Making the KM estimates for all CRT and non-CRT components.

```
In [ ]: for k in range(1,10):
            survCurvePlot(k, TNoCRT, ENoCRT, survInfoNoCRT, 'NoCRT', False)
            survCurvePlot(k, TCRT, ECRT, survInfoCRT, 'CRT', False)
```

1.10 Univariate Cox Regression

We want to build a pandas dataframe containing all results from an univariate Cox regression survival model. In particular we want the a p-value, 95% confidence intervals, the coefficient, and exp(coef) and the c-index.

```
In [ ]: volCompNameList = [f'VolumeComp{k:d}' for k in range(1,10)]
```

```
In [ ]: def uniCox(T, E, survInfo, varName):
            DF = pd.DataFrame({'Duration': T, 'Event':E,
                               varName:survInfo.loc[:,varName]/1000})
            cph = CoxPHFitter()
            cph.fit(DF, duration_col='Duration', event_col='Event',
                    step_size=0.1, show_progress=True)
            res = cph.summary
            res['c-index'] = cph.score_
            print(cph.check_assumptions(DF))
            return(res)
```

1.10.1 CRT

```
In [ ]: pd.options.display.float_format = '{:.3g}'.format
        CRTUniCox = pd.DataFrame()
        for comp in volCompNameList:
            CRTUniCox=CRTUniCox.append(uniCox(TCRT, ECRT, survInfoCRT, comp))
        CRTUniCox
```

1.10.2 No CRT

```
In [ ]: pd.options.display.float_format = '{:.3g}'.format
        NoCRTUniCox = pd.DataFrame()
        for comp in volCompNameList:
            try:
                NoCRTUniCox=NoCRTUniCox.append(uniCox(TNoCRT, ENoCRT, survInfoNoCRT, comp))
            except:
                pass
        NoCRTUniCox
```

1.10.3 Summary of univariate Cox Regression

```
In [ ]: CRTUniCox.reset_index(inplace=True)
        CRTUniCox.rename({'index':'parameter'}, axis=1, inplace=True)
        CRTUniCox['treatment'] = ['CRT']*CRTUniCox.shape[0]
        CRTUniCox.set_index(['treatment', 'parameter'], inplace=True)
```

```
In [ ]: NoCRTUniCox.reset_index(inplace=True)
        NoCRTUniCox.rename({'index':'parameter'}, axis=1, inplace=True)
        NoCRTUniCox['treatment'] = ['NoCRT']*NoCRTUniCox.shape[0]
        NoCRTUniCox.set_index(['treatment', 'parameter'], inplace=True)
```

```
In [ ]: UniCoxDF = CRTUniCox.append(NoCRTUniCox)
```

```
In [ ]: uniCoxLatex = UniCoxDF.drop(['se(coef)','z', '-log2(p)', 'c-index'], axis=1)
        uniCoxLatex.rename({'coef':'$\\beta$',
                       'exp(coef)':'$e^\\beta$', 'p':'$p$'}, inplace=True, axis = 1)
```

```
In [ ]: from latex_table_utils import merge_columns_by_name
        uniCoxLatex = merge_columns_by_name(uniCoxLatex,
                        '$95 \\%$ CI' ,'lower 0.95', 'upper 0.95', 3)
```

```
In [ ]: from latex_table_utils import build_latex_table
        build_latex_table(uniCoxLatex, 1, 1,
                './Results/GMM/tables/UnicariateCox.txt', 3)
```

11

1.11 Finding the correlation between the different volume components and the total volume for each treatment group

```
In [ ]: corr = mCRTDF.drop(['Duration','Event'],axis=1).corr()
        plt.figure(figsize=(8,6))
        sns.heatmap(corr,
                    xticklabels=corr.columns.values,
                    yticklabels=corr.columns.values,
                   annot=True,
                   fmt='.2f')
        plt.tight_layout()
        plt.savefig('./figures/Correlation/CRTCorr.pdf')
        plt.show()
```

```
In [ ]: corr = mNoCRTDF.drop(['Duration','Event'], axis=1).corr()
        plt.figure(figsize=(8,6))
        sns.heatmap(corr,
                    xticklabels=corr.columns.values,
                    yticklabels=corr.columns.values,
                   annot=True,
                   fmt='.2f')
        plt.tight_layout()
        plt.savefig('./figures/Correlation/NoCRTCorr.pdf')
        plt.show()
```

A.5 Test of DB, DB* and DB** cluster validity indices

In this section a printout from the jupyter notebook where the performance of the DB* and DB** implemented in as shown section A.4.2 is tested against DB.

Test of my DB* and DB** implementation

The purpose of this notebook is to test my implementation of the improved Davies-Bouldin indices proposed by Kim and Ramakrishna, 2005 [1]. The DB and DB* indices were found to perform well in the extensive study of Cluster Validation Indices (CVIs) by Arbelaitz et al (2013) [2].

```
In [12]: import sys, os
         sys.path.append(os.path.abspath(os.path.join('..', 'kMeans','pythonDataExploration')))

         import numpy as np
         import matplotlib.pyplot as plt
         from sklearn.metrics import davies_bouldin_score
         from sklearn.datasets import make_blobs
         from sklearn.cluster import KMeans

         from cluster_validity_indices import davies_bouldin_star_score, davies_bouldin_star_star_score
         %matplotlib notebook
         import seaborn as sns
         import pandas as pd

         figSavePath = 'New DB indices tests'

         try:
             os.mkdir(figSavePath)
         except:
             print("Directory already exists")

           Directory already exists
```

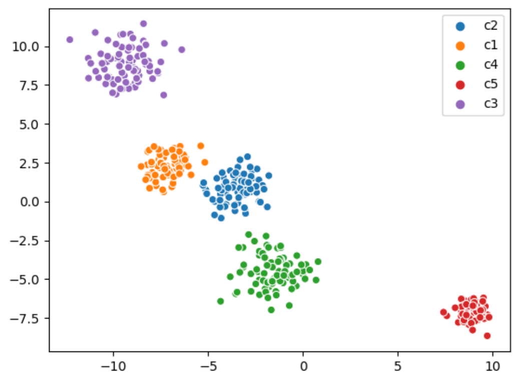
```
In [13]: os.path.join(figSavePath,'SaveName')

Out[13]:   'New DB indices tests\\SaveName'
```

The datasets

To make the testing simple I'll use some of the sample datasets included in the scikit-learn package namely the make_blobs datasets [2]. This makes isotropic gaussian clusters in two dimensions. The make_blobs functions, amoung other things, allows setting the number of samples, number of clusters and the standard deviation for each of the clusters. An example of the initialization of such a dataset is shown below.

```
In [14]: n_samples = 400
         X, y = make_blobs(n_samples=n_samples, centers= 5,
                                     cluster_std=[0.7, 0.8, 1.1, 1, 0.5])
         labelsMB = [f'c{i+1}' for i in y]
```

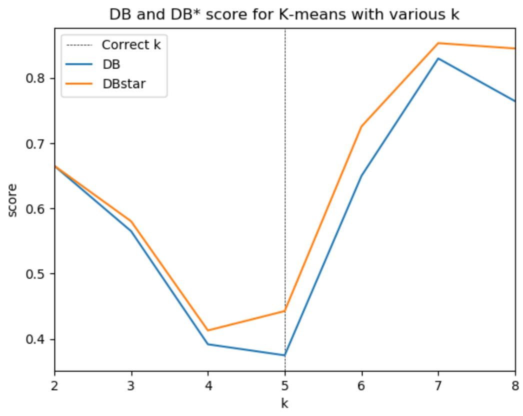
In [15]:
```
plt.figure()
sns.scatterplot(X[:,0],X[:,1], hue=labelsMB)
plt.savefig(os.path.join(figSavePath,'MakeBlobsExample.pdf'))
```



The centers of the clusters are set at random within a bounding box from -10 to 10 in both dimensions. Thus more or less overlap will occur between the clusters on different initializations of the dataset. Below is a test of the DB index implemented in sklearn toghether with my implementation of DB*.

```
In [16]: klist = np.arange(2,9,1)
         DB = []
         DBstar = []
         for k in klist:
             kmeans = KMeans(n_clusters=k, random_state=1).fit(X)
             labels = kmeans.labels_
             DBstar.append(davies_bouldin_star_score(X,labels))
             DB.append(davies_bouldin_score(X,labels))
         plt.figure()
         plt.axvline(x=5,c='k',linewidth = 0.5, label='Correct k', linestyle = '--')
         plt.plot(klist,DB, label="DB")
         plt.plot(klist, DBstar, label="DBstar")
         plt.legend()
         plt.xlabel('k')
         plt.ylabel('score')
         plt.title('DB and DB* score for K-means with various k')
         plt.xlim([2, 8])
         plt.savefig(os.path.join(figSavePath,'MakeBlobsExampleDBandDBstar.pdf'))
         plt.show()
```

```
C:\Users\bendi\Anaconda3\lib\site-packages\sklearn\metrics\cluster\unsupervised.py:342: RuntimeWarning: divide by zero encountered in true
_divide
  score = (intra_dists[:, None] + intra_dists) / centroid_distances
C:\Users\bendi\Anaconda3\lib\site-packages\sklearn\metrics\cluster\unsupervised.py:342: RuntimeWarning: divide by zero encountered in true
_divide
  score = (intra_dists[:, None] + intra_dists) / centroid_distances
C:\Users\bendi\Anaconda3\lib\site-packages\sklearn\metrics\cluster\unsupervised.py:342: RuntimeWarning: divide by zero encountered in true
_divide
  score = (intra_dists[:, None] + intra_dists) / centroid_distances
C:\Users\bendi\Anaconda3\lib\site-packages\sklearn\metrics\cluster\unsupervised.py:342: RuntimeWarning: divide by zero encountered in true
_divide
  score = (intra_dists[:, None] + intra_dists) / centroid_distances
C:\Users\bendi\Anaconda3\lib\site-packages\sklearn\metrics\cluster\unsupervised.py:342: RuntimeWarning: divide by zero encountered in true
_divide
  score = (intra_dists[:, None] + intra_dists) / centroid_distances
C:\Users\bendi\Anaconda3\lib\site-packages\sklearn\metrics\cluster\unsupervised.py:342: RuntimeWarning: divide by zero encountered in true
_divide
  score = (intra_dists[:, None] + intra_dists) / centroid_distances
C:\Users\bendi\Anaconda3\lib\site-packages\sklearn\metrics\cluster\unsupervised.py:342: RuntimeWarning: divide by zero encountered in true
_divide
  score = (intra_dists[:, None] + intra_dists) / centroid_distances
```



An extensive test of DB, DB* and DB**

Here an extensive test will be run using the make_blobs datasets. If there is any reason I should use DB *or DB** instead of DB in my analysis they at least need to on average outperform DB on datasets that are well behaved. The primary focus however, is to see whether my implementation of DB* and DB **are correct. If so, they should both outperform DB on average [2,3]. For these test 200 datasets will be made with 2,3,...,7 clusters with the make_blobs function making a total of 1200 datasets. The number of samples in each dataset will be an integer from a uniform probability distribution between 200 and 700 and the standard deviation of each cluster will be chosen as a random number between 1.2 and 0.5 also with a uniform probability distribution. As pointed out earlier, the cluster may in some cases be perfectly separated, and in other cases overlap completely or close to completely. This does not really matter as we should still, if the implementation of DB* and DB** are correct, see that they outperform DB on average across all the datasets. As performance measures the meas percentage error will be used.

```
In [143]:
cList = np.arange(2,8,1)
kList = np.arange(2,14,1)
absDiffDBstarstar = []
absDiffDBstar = []
absDiffDB = []
MPEDBstarstar = []
MPEDBstar = []
MPEDB = []
n_reps = 200 # number of datasets for each c in cList

np.seterr(all='ignore')

for nc in cList:
    # to see progress
    print(f'Step {np.argwhere(cList==nc)[0,0]+1} of {len(cList)}')
    print(f'Performing analalysis on {n_reps} datasets of {nc} clusters')
    for i in range(0,n_reps):
        X, y = make_blobs(n_samples=np.random.randint(200,701), centers= nc,
                          cluster_std=np.random.random(nc)*(1.2-0.5)+0.5)
        labels = []
        DBstar = []
        DB = []
        for k in kList:
            kmeans = KMeans(n_clusters=k).fit(X)
            labels.append(kmeans.labels_)
            DBstar.append(davies_bouldin_star_score(X,kmeans.labels_))
            DB.append(davies_bouldin_score(X, kmeans.labels_))
        DBstarstar = davies_bouldin_star_star_score(X, labels)

        absDiffDBstarstar.append(abs(nc - kList[np.argmin(DBstarstar)]))
        absDiffDBstar.append(abs(nc - kList[np.argmin(DBstar)]))
        absDiffDB.append(abs(nc-kList[np.argmin(DB)]))
        MPEDBstarstar.append(abs(nc - kList[np.argmin(DBstarstar)])*100/nc)
        MPEDBstar.append(abs(nc - kList[np.argmin(DBstar)])*100/nc)
        MPEDB.append(abs(nc-kList[np.argmin(DB)])*100/nc)
```

```
2
3

C:\Users\bendi\Anaconda3\lib\site-packages\sklearn\metrics\cluster\unsupervised.py:342: RuntimeWarning: invalid value encountered in true_
divide
  score = (intra_dists[:, None] + intra_dists) / centroid_distances

4
5

C:\Users\bendi\Anaconda3\lib\site-packages\sklearn\metrics\cluster\unsupervised.py:342: RuntimeWarning: invalid value encountered in true_
divide
  score = (intra_dists[:, None] + intra_dists) / centroid_distances
C:\Users\bendi\Anaconda3\lib\site-packages\sklearn\metrics\cluster\unsupervised.py:342: RuntimeWarning: invalid value encountered in true_
divide
  score = (intra_dists[:, None] + intra_dists) / centroid_distances

6
7
```

Results

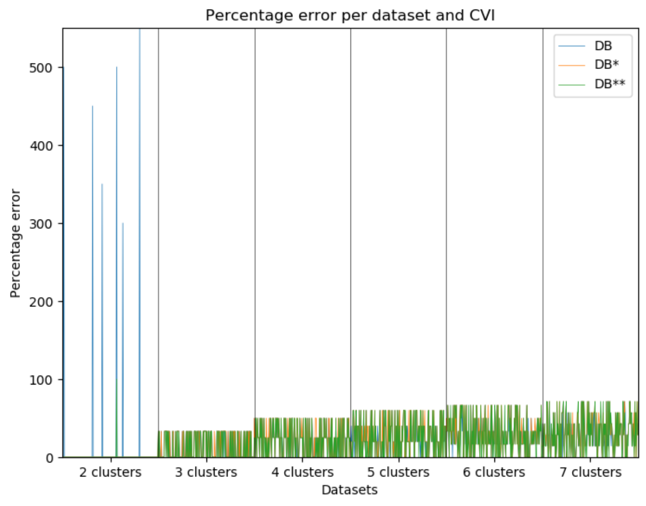
From the cell below we see that DB is outperformed by both DB* and DB **as reported in the original article [1]. DB** is the best performing of them.

```
In [144]: print(f'Mean absolute difference DB:                {np.mean(absDiffDB):.3f}')
          print(f'Mean absolute difference DB*:               {np.mean(absDiffDBstar):.3f}')
          print(f'Mean absolute difference DB**:              {np.mean(absDiffDBstarstar):.3f}')
          print(f'Mean percentage error (MPE) DB:             {np.mean(MPEDB):.3f}')
          print(f'Mean percentage error (MPE) DB*:            {np.mean(MPEDBstar):.3f}')
          print(f'Mean percentage error (MPE) DB**:           {np.mean(MPEDBstarstar):.3f}')
          print(f'percentage error standard deviation DB:     {np.std(MPEDB):.3f}')
          print(f'percentage error standard deviation DB*:    {np.std(MPEDBstar):.3f}')
          print(f'percentage error standard deviation DB**:   {np.std(MPEDBstarstar):.3f}')
```

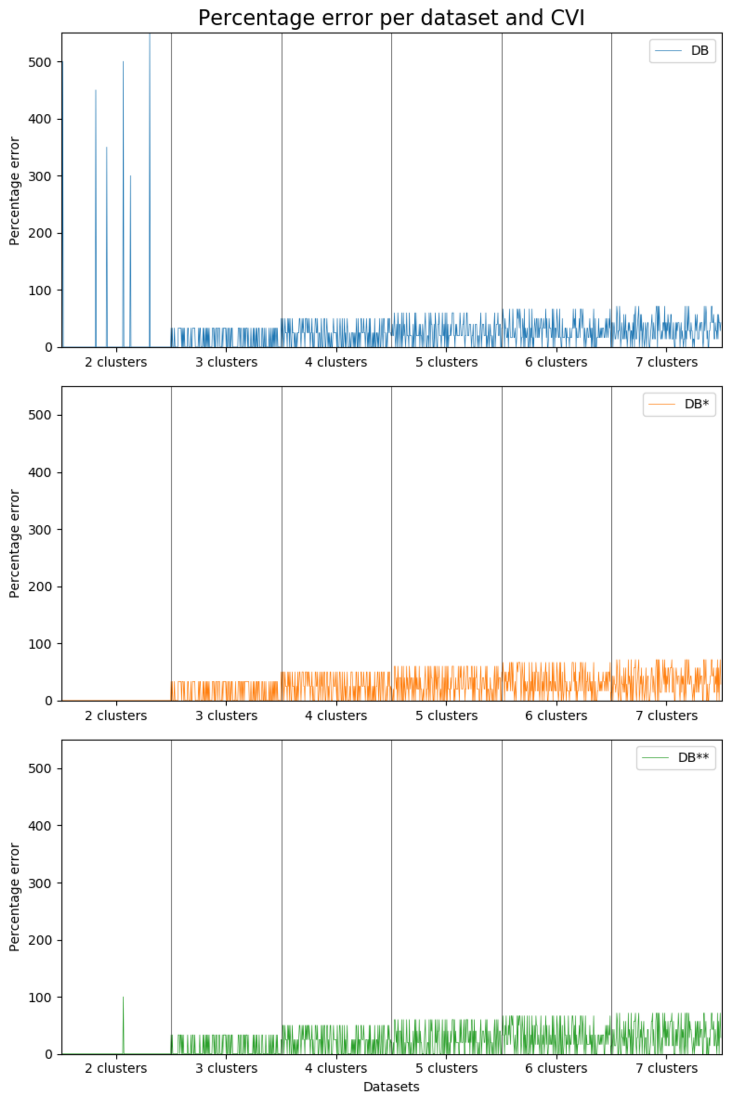
```
Mean absolute difference DB:                1.120
Mean absolute difference DB*:               1.160
Mean absolute difference DB**:              1.051
Mean percentage error (MPE) DB:             22.343
Mean percentage error (MPE) DB*:            21.779
Mean percentage error (MPE) DB**:           19.481
percentage error standard deviation DB:     36.495
percentage error standard deviation DB*:    22.325
percentage error standard deviation DB**:   22.439
```

Below is a plot showing percentage error on each dataset for each method. The methods can be seen all in the same plot, and as three different subplots. The datasets are grouped according to the number of clusters they have. The most noticeable trait is the fluctuationsof the DB index for datasets with two clusters. Both DB *and DB** seem to be more stable. This is might explain some of the reason why the standard deviation of the percentage error of DB is conciderably higher than those of DB* and DB**.

```
In [213]: plt.figure(figsize=(8,6))

          plt.plot(MPEDB, label='DB', linewidth=0.5)
          plt.plot(MPEDBstar, label='DB*', linewidth=0.5)
          plt.plot(MPEDBstarstar, label='DB**',linewidth=0.5)

          lw = 0.4
          for markerLinePosition in [n_reps*i for i in np.arange(len(cList)-1)+1]:
              plt.axvline(markerLinePosition, c='k', linewidth=lw)

          plt.xlim([0, len(MPEDBstar)])
          plt.ylim([0, np.max([np.max(MPEDBstar), np.max(MPEDBstarstar), np.max(MPEDB)])])
          plt.xticks(np.arange(n_reps/2,n_reps/2 + n_reps*len(cList),n_reps),[f'{i} clusters' for i in cList
          ])
          plt.legend()
          plt.title('Percentage error per dataset and CVI')
          plt.ylabel('Percentage error')
          plt.xlabel('Datasets')
          plt.savefig(os.path.join(figSavePath,'PercentageErrorPerDataset.pdf'))
```



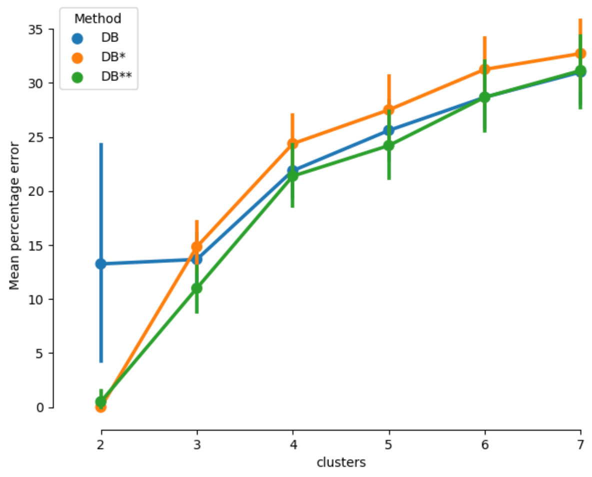
```
In [212]: plt.figure(figsize=(8,12))

          plt.subplot(311)
          plt.title('Percentage error per dataset and CVI',size=16)
          plt.plot(MPEDB, label='DB', linewidth=0.5)
          lw = 0.4
          for markerLinePosition in [n_reps*i for i in np.arange(len(cList)-1)+1]:
              plt.axvline(markerLinePosition, c='k', linewidth=lw)
          plt.xlim([0, len(MPEDBstar)])
          plt.ylim([0, np.max([np.max(MPEDBstar), np.max(MPEDBstarstar), np.max(MPEDB)])])
          plt.xticks(np.arange(n_reps/2,n_reps/2 + n_reps*len(cList),n_reps),[f'{i} clusters' for i in cList
          ])
          plt.legend()
          plt.ylabel("Percentage error")

          plt.subplot(312)
          plt.plot(MPEDBstar, label='DB*', linewidth=0.5,c="tab:orange")
          plt.ylim([0, np.max([np.max(MPEDBstar), np.max(MPEDBstarstar), np.max(MPEDB)])])
          plt.xlim([0, len(MPEDBstar)])
          plt.legend()
          for markerLinePosition in [n_reps*i for i in np.arange(len(cList)-1)+1]:
              plt.axvline(markerLinePosition, c='k', linewidth=lw)
          plt.xticks(np.arange(n_reps/2,n_reps/2 + n_reps*len(cList),n_reps),[f'{i} clusters' for i in cList
          ])
          plt.ylabel("Percentage error")

          plt.subplot(313)
          plt.plot(MPEDBstarstar, label='DB**',linewidth=0.5, c="tab:green")
          plt.ylim([0, np.max([np.max(MPEDBstar), np.max(MPEDBstarstar), np.max(MPEDB)])])
          plt.xlim([0, len(MPEDBstar)])
          plt.legend()
          for markerLinePosition in [n_reps*i for i in np.arange(len(cList)-1)+1]:
              plt.axvline(markerLinePosition, c='k', linewidth=lw)
          plt.xticks(np.arange(n_reps/2,n_reps/2 + n_reps*len(cList),n_reps),[f'{i} clusters' for i in cList
          ])
          plt.xlabel("Datasets")
          plt.ylabel("Percentage error")
          plt.tight_layout()
          plt.savefig(os.path.join(figSavePath,'PercentageErrorPerDatasetSubplots.pdf'))
```



Due to the number of datasets it is a bit difficult to really be able to discern all the properties of the CVIs using the plot above. Below a pointplot is made. This plots points at the mean percentage error of the CVIs on datasets containing the same amount of clusters. The errorbars are the 95 percent confidence intervals around the mean.

In [146]:
```python
cLabel = np.array([n_reps*[c] for c in cList]).reshape(-1)
cLabel = np.concatenate((cLabel,cLabel,cLabel))
dfDict = {'clusters': cLabel, 'MPE': np.concatenate((MPEDB,MPEDBstar,MPEDBstarstar),axis=0),
          'Method': len(MPEDB)*['DB'] + len(MPEDBstar)*['DB*'] + len(MPEDBstarstar)*['DB**'] }
```

In [211]:
```python
plt.figure(figsize=(8,6))
sns.pointplot(x='clusters', y='MPE', hue='Method', data=df, ci=95)
sns.despine(offset=0, trim=True)
plt.ylabel('Mean percentage error')
plt.savefig(os.path.join(figSavePath,'MPEperClusterGroup.pdf'))
```



References

[1] Minho Kim, R.S. Ramakrishna, New indices for cluster validity assessment, Pattern Recognition Letters, Volume 26, Issue 15, 2005, Pages 2353-2363, ISSN 0167-8655, https://doi.org/10.1016/j.patrec.2005.04.007 (https://doi.org/10.1016/j.patrec.2005.04.007). (http://www.sciencedirect.com/science/article/pii/S016786550500125X (http://www.sciencedirect.com/science/article/pii/S016786550500125X))

[2] Olatz Arbelaitz, Ibai Gurrutxaga, Javier Muguerza, Jesús M. Pérez, Iñigo Perona, An extensive comparative study of cluster validity indices, Pattern Recognition, Volume 46, Issue 1, 2013, Pages 243-256, ISSN 0031-3203, https://doi.org/10.1016/j.patcog.2012.07.021 (https://doi.org/10.1016/j.patcog.2012.07.021). (http://www.sciencedirect.com/science/article/pii/S003132031200338X (http://www.sciencedirect.com/science/article/pii/S003132031200338X))

[3] Scikit-learn, Sklearn.datasets.make_blobs, https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html (https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html), retrieved 27.04.2019

A.6 Finding the optimal k for the k-means algorithm

In the k-means algorithm k is a required input parameter. Some of the difficulties in using the k-means algorithm thus arises in trying to select the optimal k for a given dataset. A more in depth discussion about this can be found in section about Cluster Validity Indices (CVIs) in the theory 2.4.2. The chosen CVIs for report were the Calinski-Harabasz (CH) Davies-Bouldin family of indices, DB, DB* and DB** and the Silhouette score (Sil). These are all defined in the relevant section of the theory 2.4.2. The CVIs were computed for both a dataset including polynomial interaction features up to and including the third order in DWI features with b-values 25, 50, 100 and $500\,\mathrm{s\,mm^{-2}}$ and a dataset without interaction feature.

The are CVIs are computed for k-means clustering solutions with $k \in \{1, 2 \ldots 21\}$ with the exception of DB** that is only calculated up to $k_{max} = 20$ due to the fact that it relies on the clustering solution for with $k = k_{max} + 1$. CH and the DB familiy of indices were computed using the full dataset which is feasible due to the fact that both of these indices have linear time complexity in the number of samples. The Sil was calculated using 20000 samples when interaction features were included and 5000 samples when these were omitted. The Sil calculations were repeated 20 times with randomly sampled dataset to look at the variability due to a particular subsampling of the full dataset.

The scikit-learn implementation of the CH, DB and Sil index was used. No publicly available DB* or DB** implementations were found and thus one was implemented. The implemented DB* and DB** index can be found in section A.4.2 of the appendix. A small performance evaluation of the DB* and DB** implementations can be found in section A.5 of the appendix where DB* and DB** are shown to on average outperform DB in simple clustering tasks. For more information about the testing procedures and datasets tested refer to the relevant section of the appendix.

Result for the dataset without interaction feature is presented in figures 30, 31 and 32. From the figures it can be seen that the optimal k is two across almost all criteria. The same conclusion can be drawn from figures 33, 34 and 35 showing the CH, DB, DB* and DB** and Sil calculated for the dataset without including polynomial interaction features. In the figures showing the CH and DB family of indices the scores scaled to have a minimum of zero and a maximum of one since only the relative size of the selected CVI for different k is of importance in determining the optimal k. In the Sil scores, no such rescaling is performed since the Sil score by definition ranges from zero to one.

The only criterea for which an optimal k of two is not indicated is the DB score for the dataset without interactions shown in figure 34 where an optimal k of four is indicated.

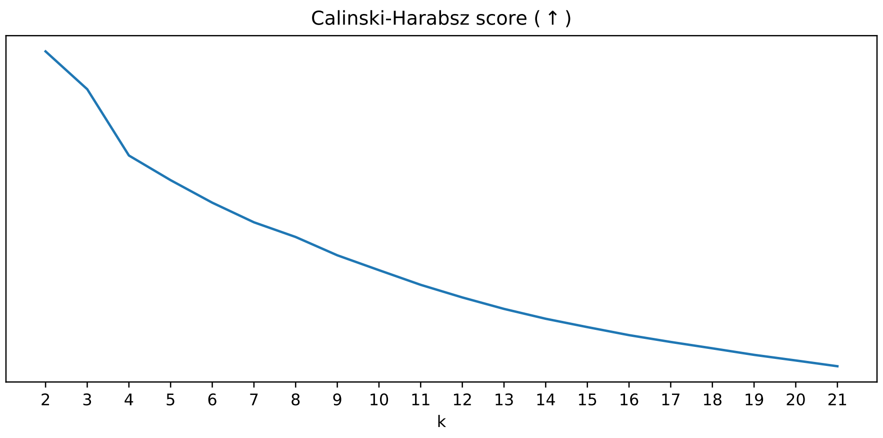


Figure 30: Calinski-Harabasz index for the dataset including polynomial interaction features.

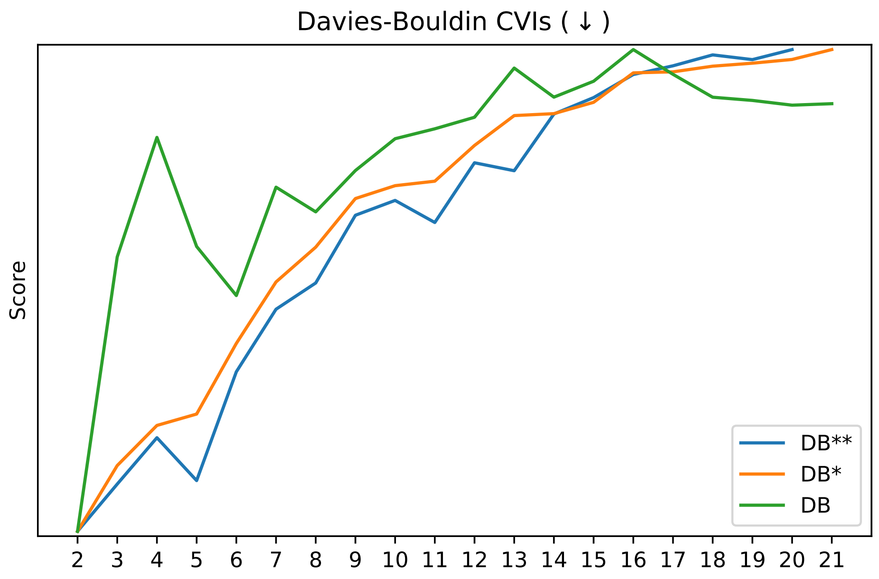


Figure 31: Davies-Bouldin index for the dataset including polynomial interaction features.

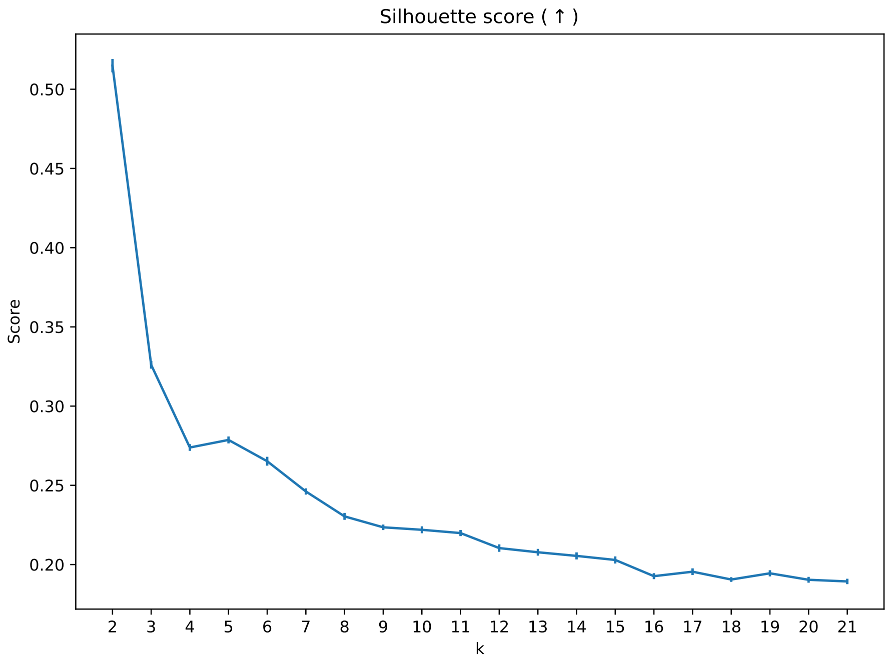


Figure 32: Silhouette score for the dataset including polynomial interaction features.

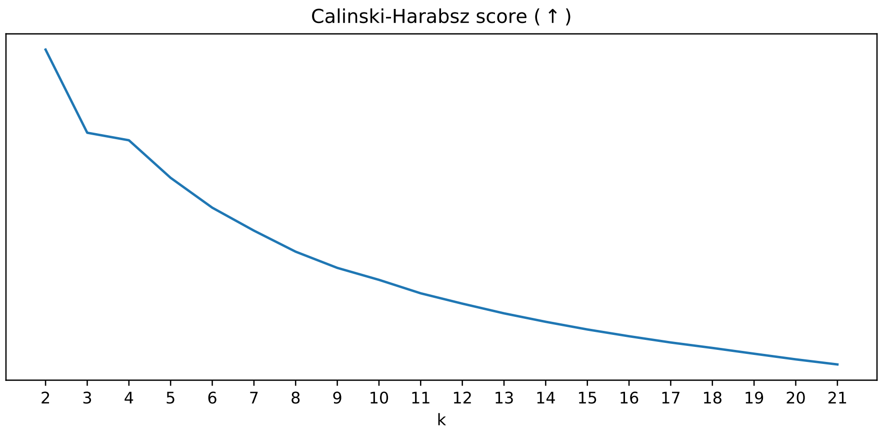


Figure 33: Calinski-Harabasz index for the dataset without including polynomial interaction features.

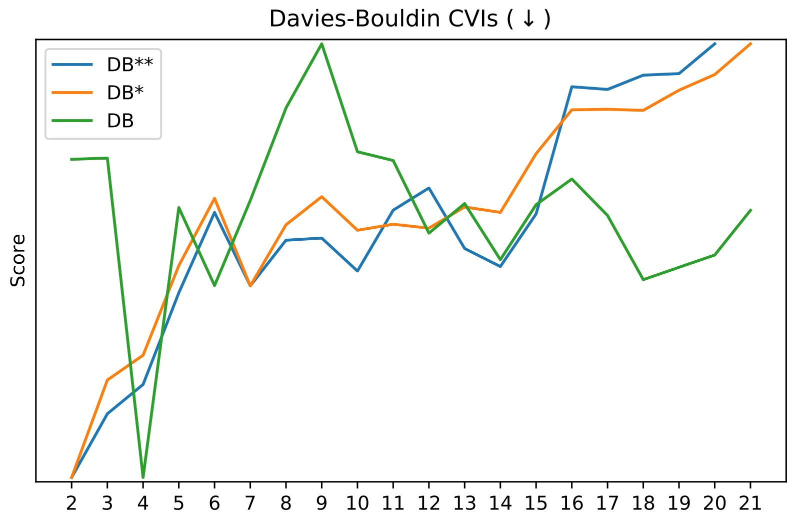


Figure 34: Davies-Bouldin index for the dataset without including polynomial interaction features.

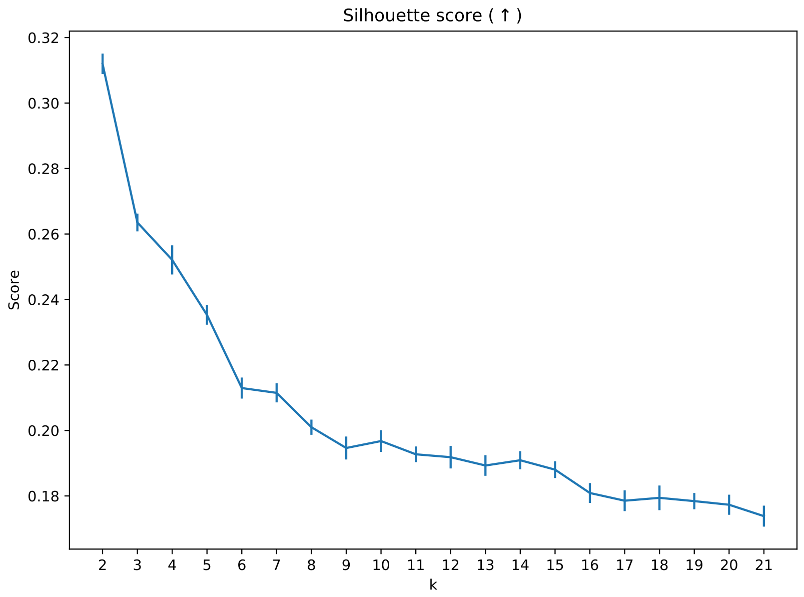
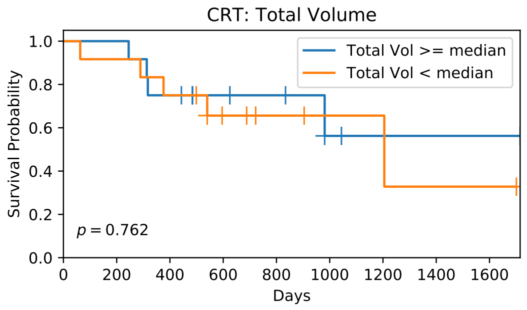


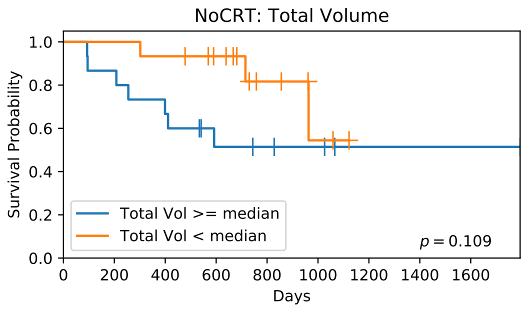
Figure 35: Silhouette score for the dataset without including polynomial interaction features.

A.7 Kaplan-Meier estimates with logrank test for the total tumour volume

The Kaplan-Meier estimates with logrank tests for the total volume partitioned according to the median tumour volume for each of the treatment groups is presented in figure 36.



(a)



(b)

Figure 36: Kaplan-Meier (KM) estimate for the total volume tumour partitioned according to the median tumour volume for each treatment group. Censored patients are indicated with crosses on the respective survival curves. The logrank test fails to show any significant association between the total volume and survival for either of the treatment groups(CRT: $p > 0.7$, No CRT: $p > 0.1$)

A.8 One component vs. rest histograms for the GMM clustering solution

Below histograms of one component of the GMM clustering solutions is plotted against all others mapped backed onto the original, z-scored feature axes. This is done to see whether any feature in particular is important in the distinction between component, and to see whether any of the properties of the tissue associated to a component can be assessed. The plots are given as referance for the reader.

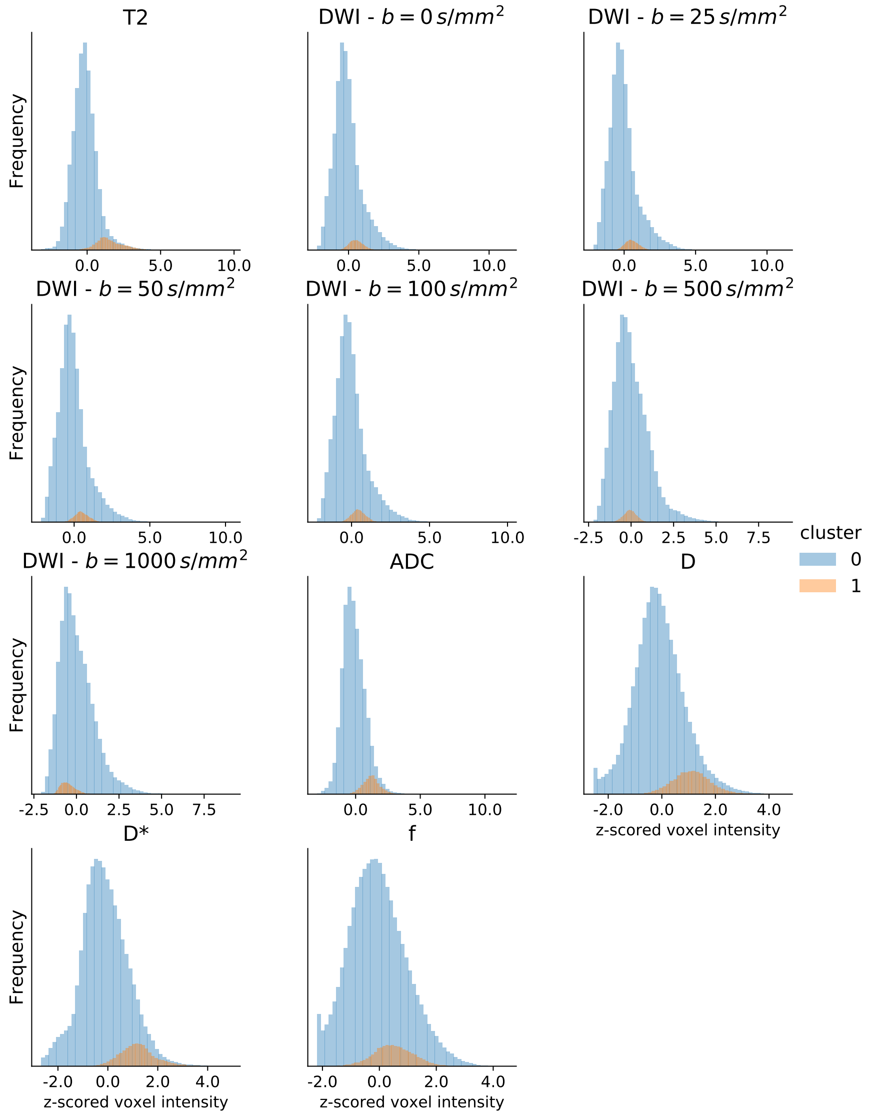


Figure 37: Histogram showing component 1 against all other components grouped together mapped along the original z-scored feature axes.

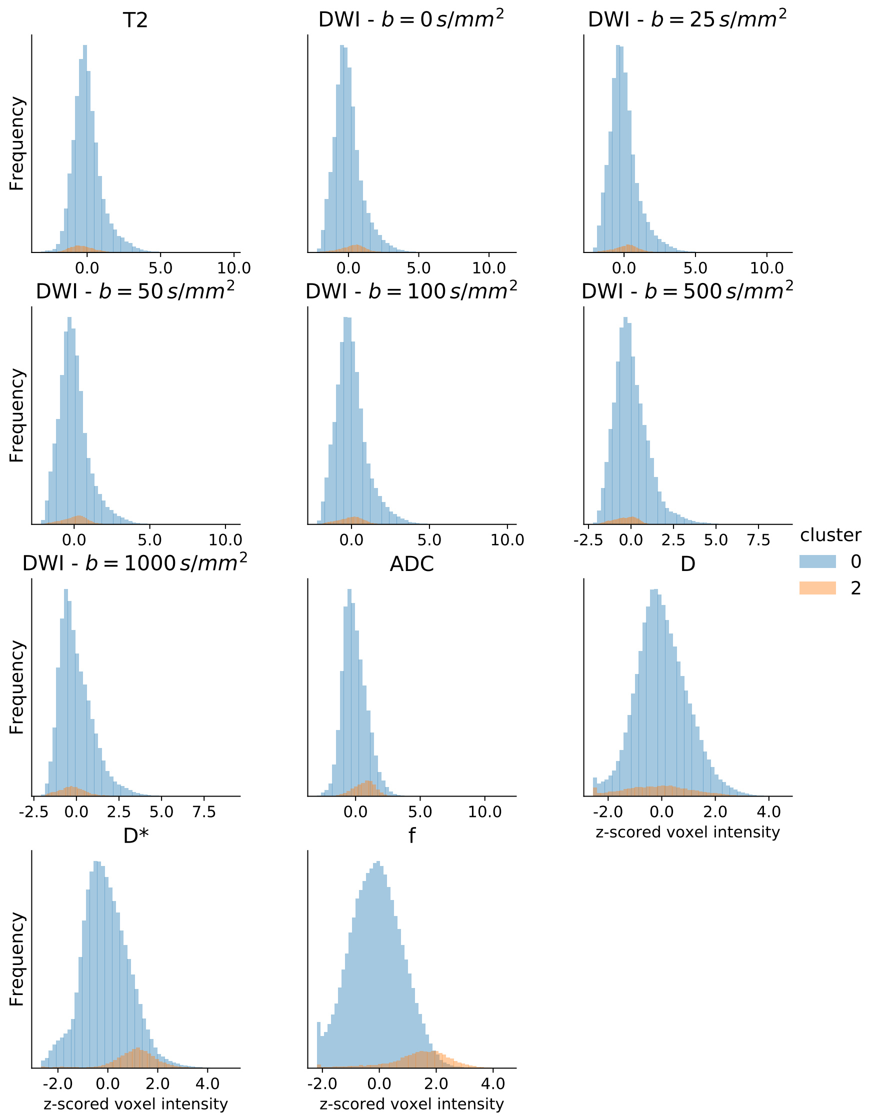


Figure 38: Histogram showing component 2 against all other components grouped together mapped along the original z-scored feature axes.

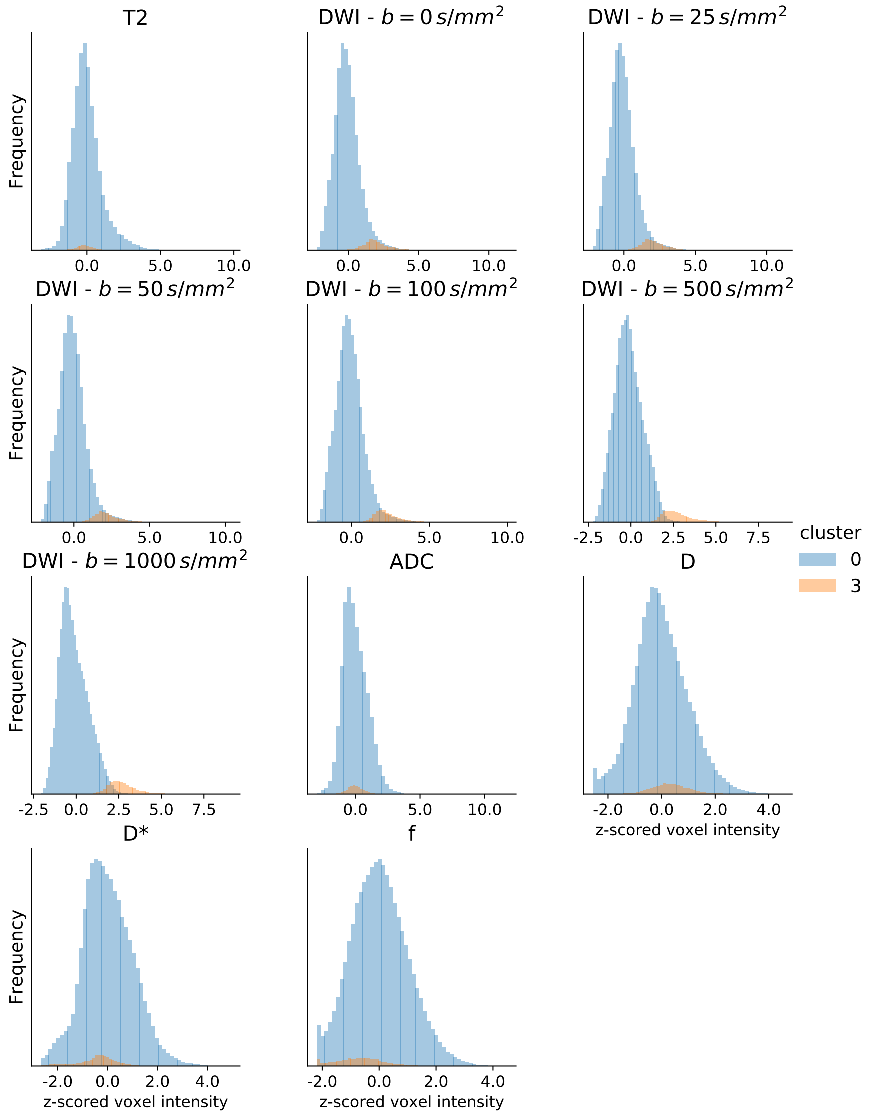


Figure 39: Histogram showing component 3 against all other components grouped together mapped along the original z-scored feature axes.

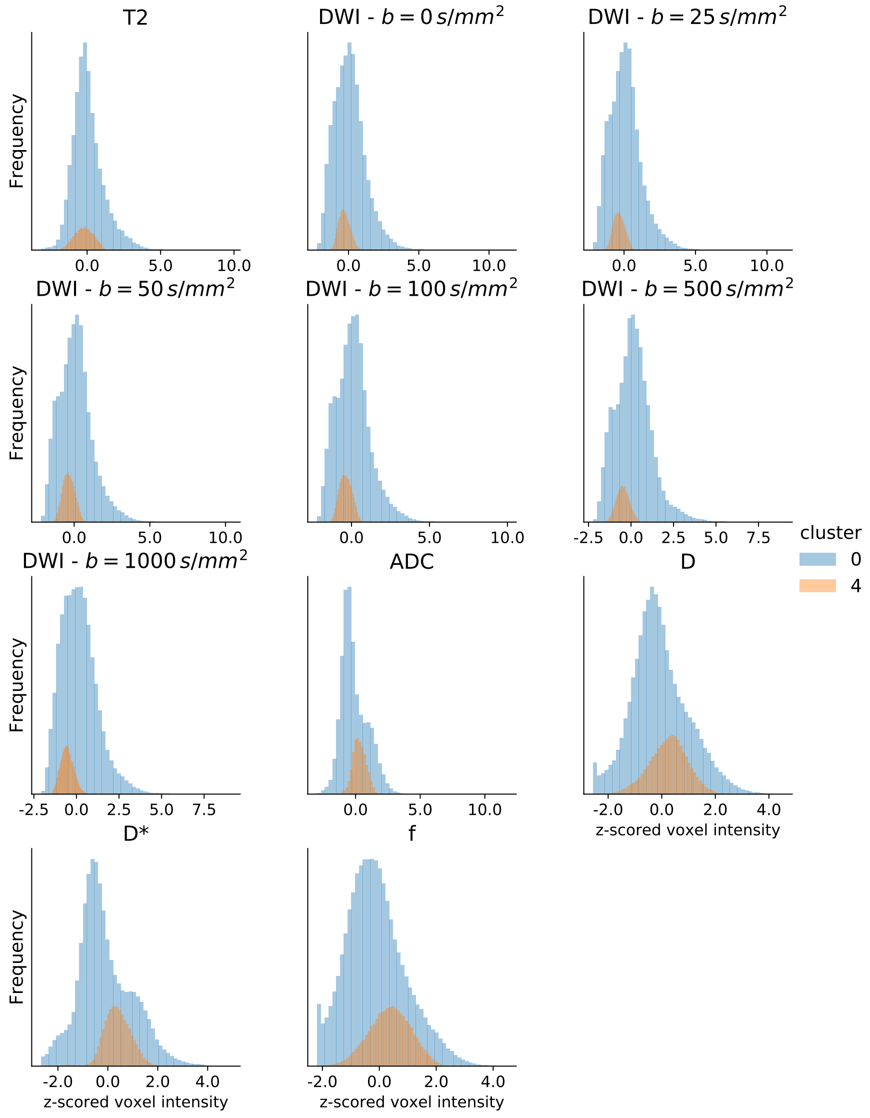


Figure 40: Histogram showing component 4 against all other components grouped together mapped along the original z-scored feature axes.

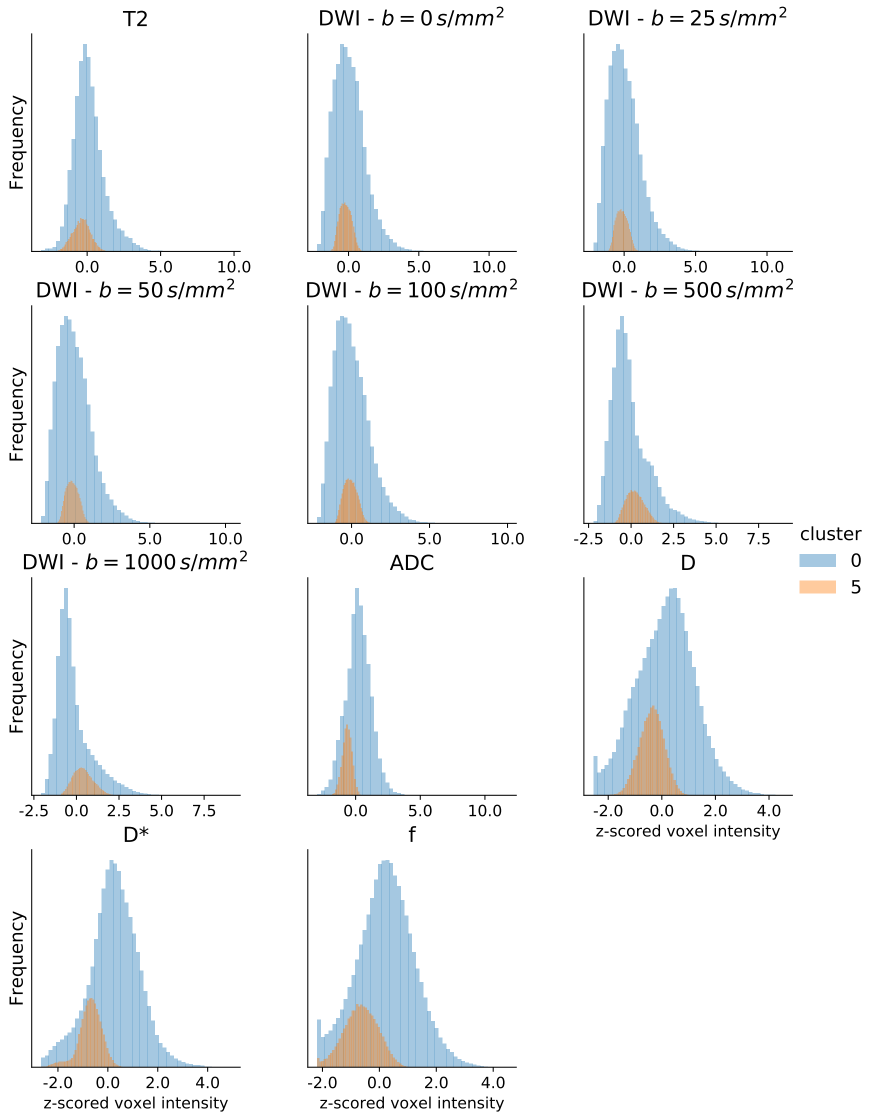


Figure 41: Histogram showing component 5 against all other components grouped together mapped along the original z-scored feature axes.

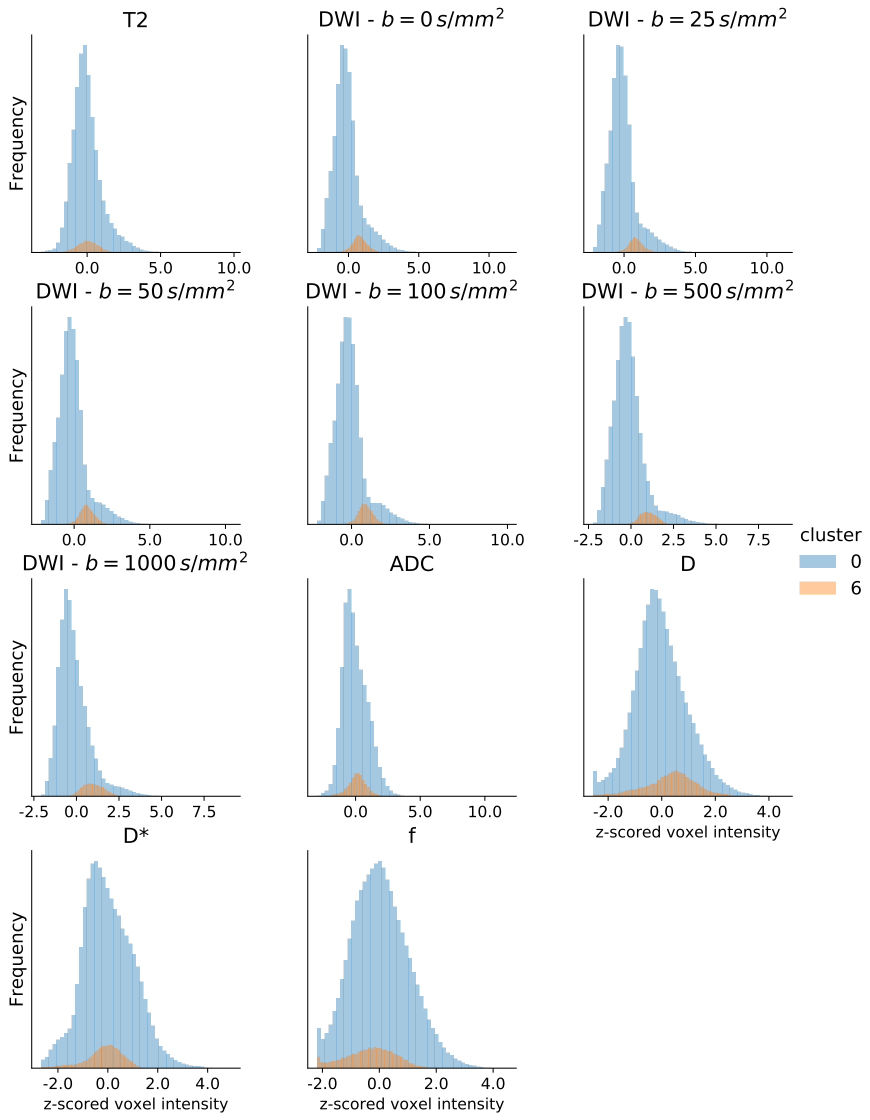


Figure 42: Histogram showing component 6 against all other components grouped together mapped along the original z-scored feature axes.

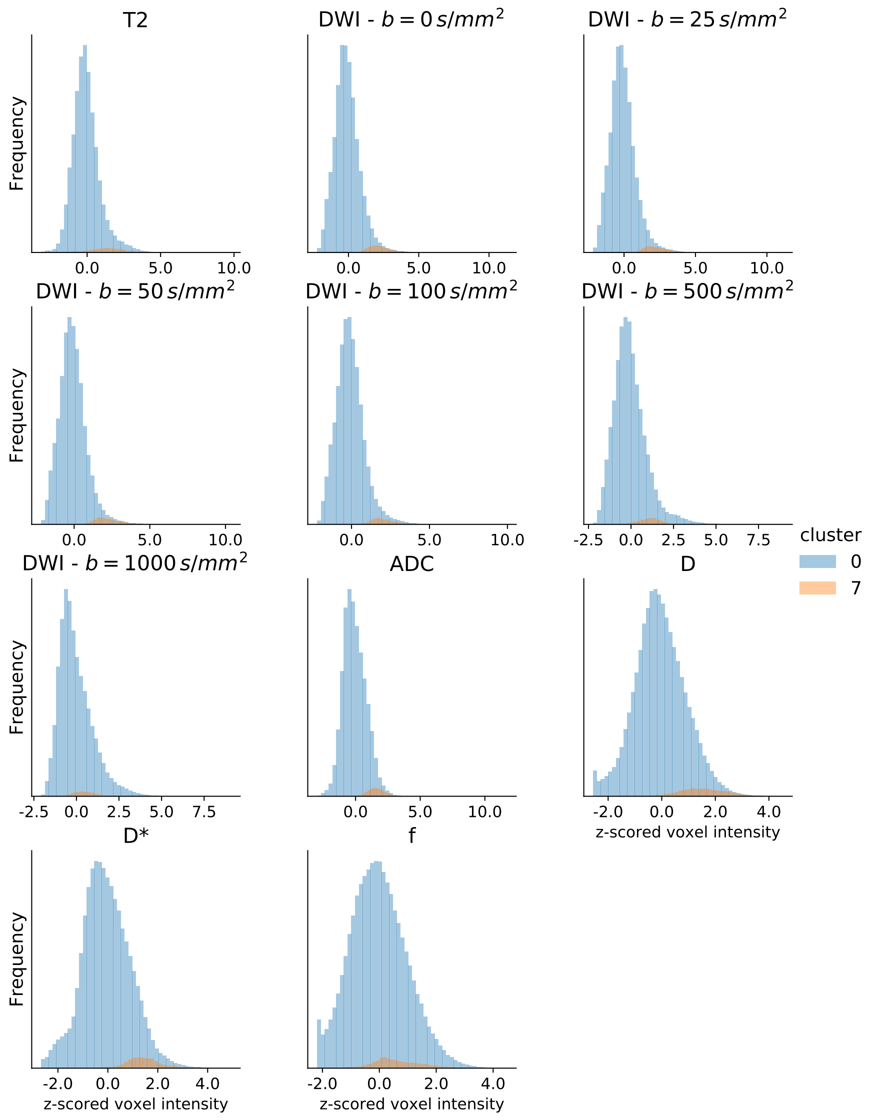


Figure 43: Histogram showing component 7 against all other components grouped together mapped along the original z-scored feature axes.

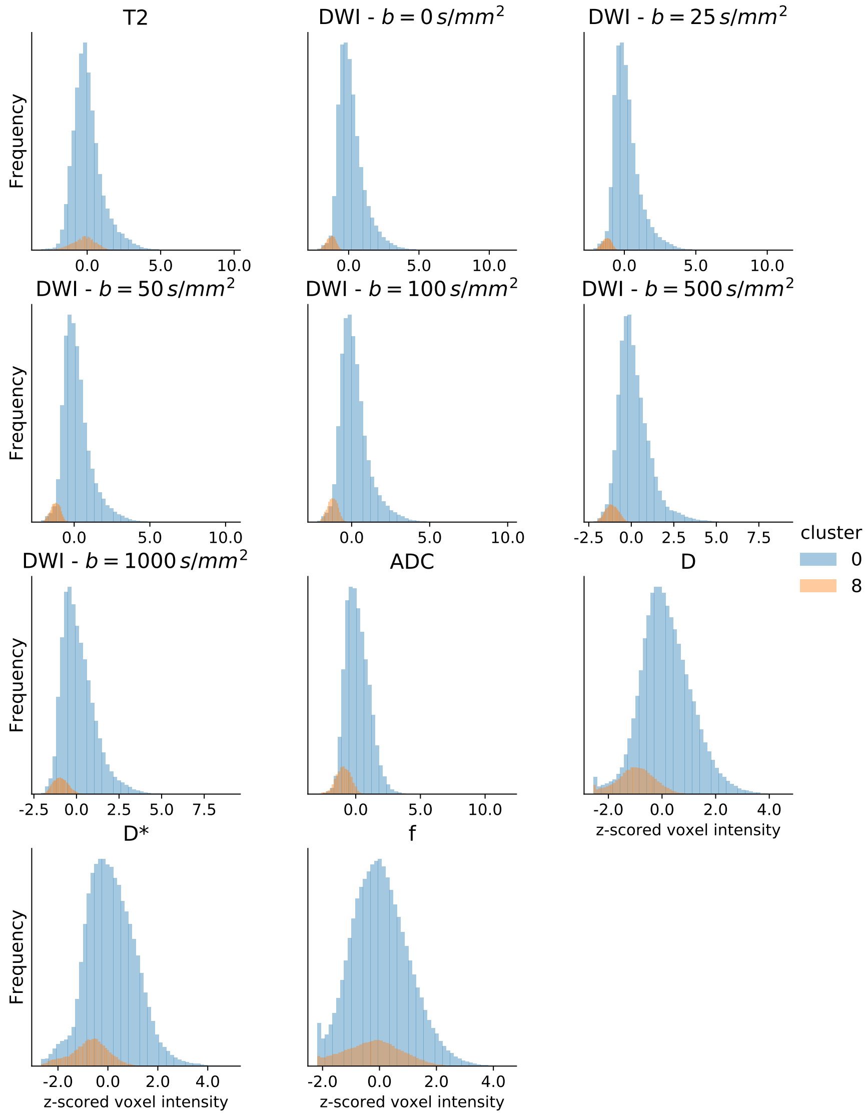


Figure 44: Histogram showing component 8 against all other components grouped together mapped along the original z-scored feature axes.

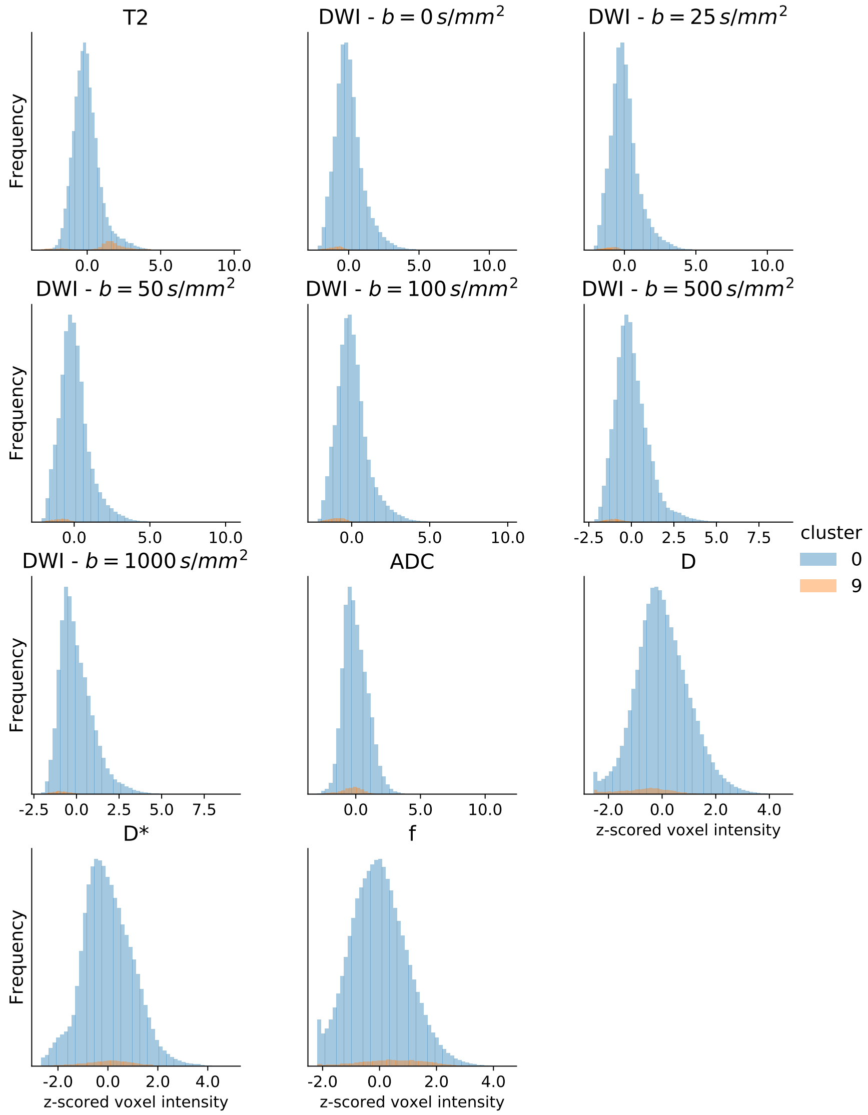


Figure 45: Histogram showing component 9 against all other components grouped together mapped along the original z-scored feature axes.

A.9 Survival Curves Of the Gaussian Mixture Model (GMM) components

Underneath a collection of all survival curves made for each of the 9 Gaussian mixture model components and for each of the two patient groups. The patients within one treatment group was partitioned according to the median component volume into low and high volume of a given component. After Bonferroni correction no significant difference in progression free survival (PFS) was found for the CRT patients. For the non CRT patient a significant difference was found for components 6 and 7. No caption will be given in the figures, as this section is in addition to the information in the figures themselves are regarded as such. A more detailed is discussion is given in the results section of the report where the survival curves shown to have a significant difference in PFS are repeated.

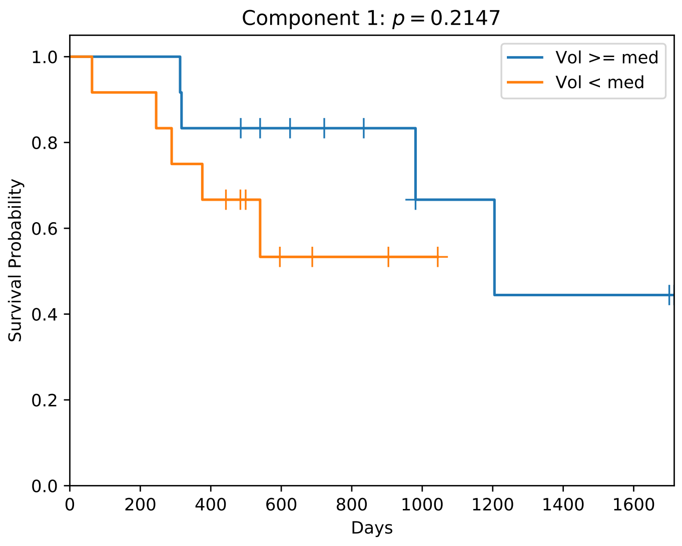


Figure 46

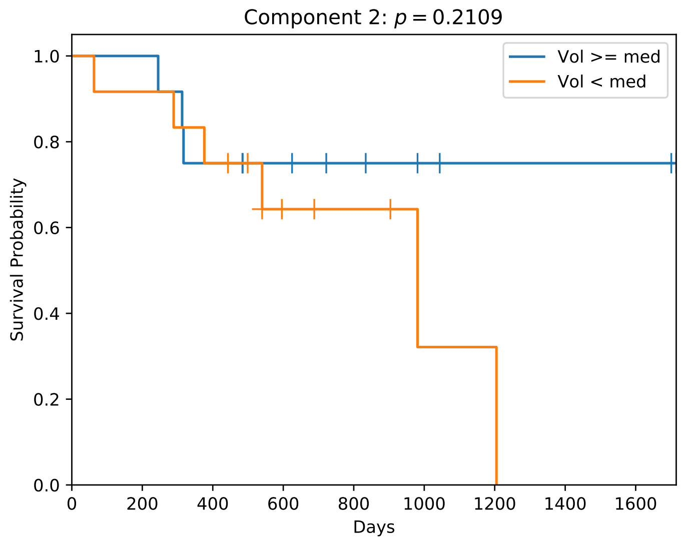


Figure 47

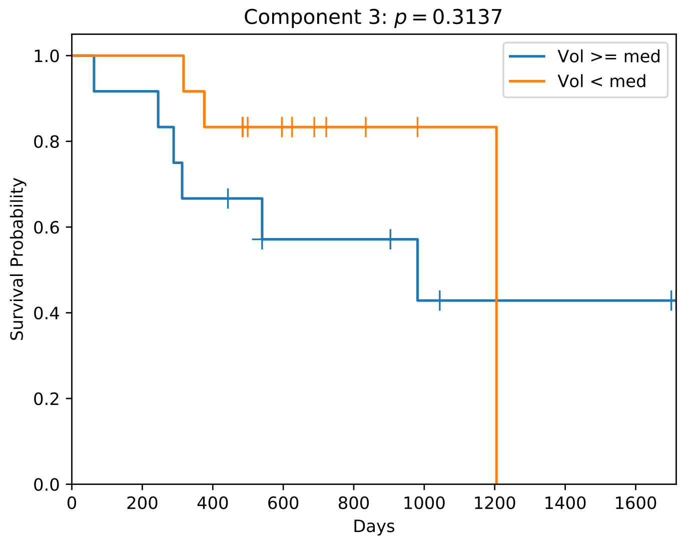


Figure 48

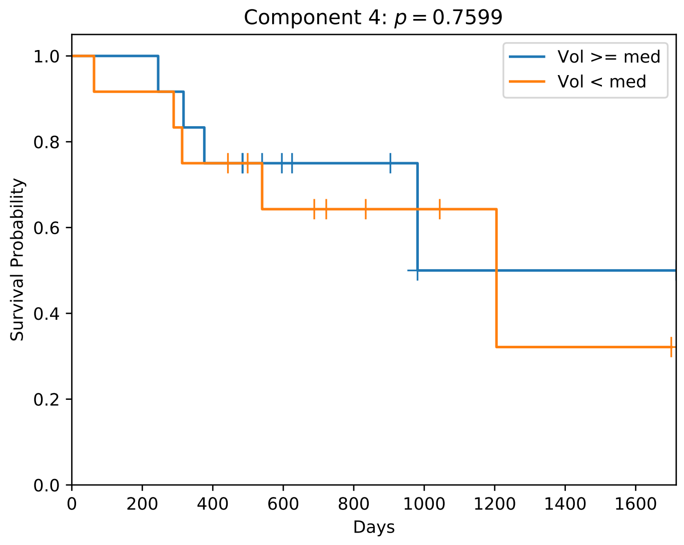


Figure 49

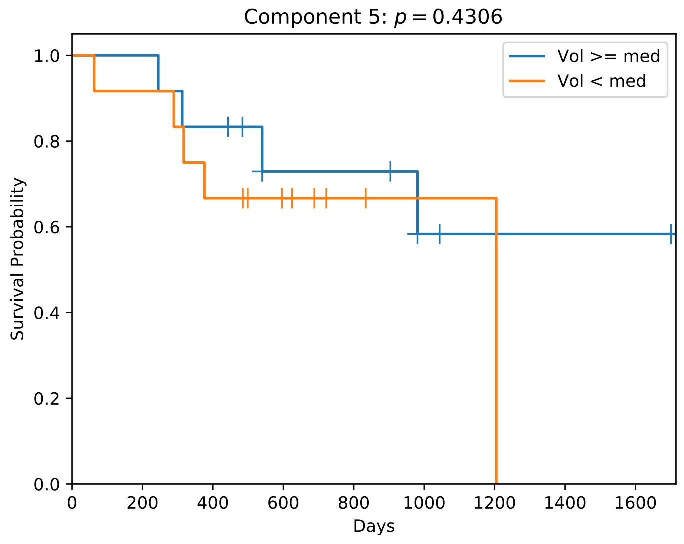


Figure 50

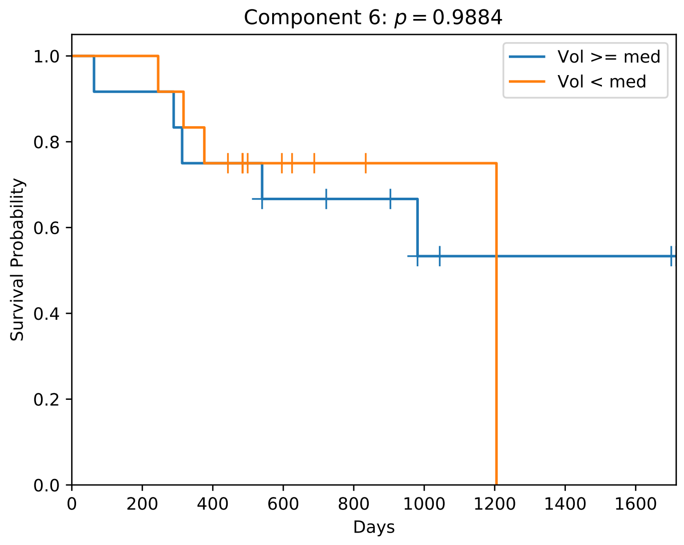


Figure 51

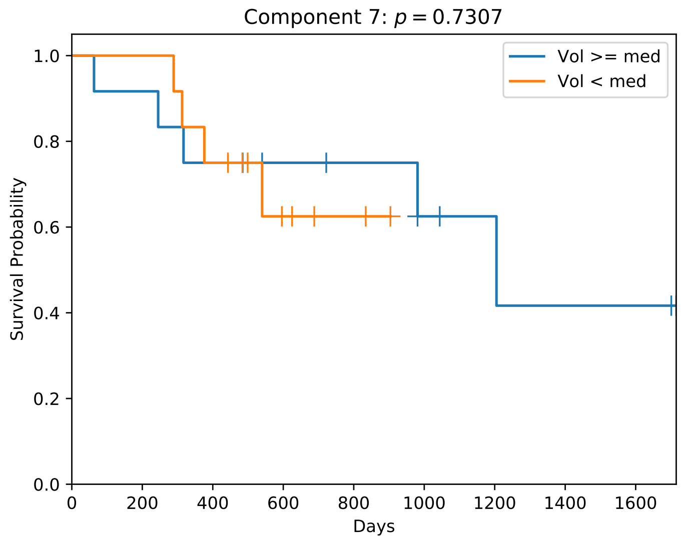


Figure 52

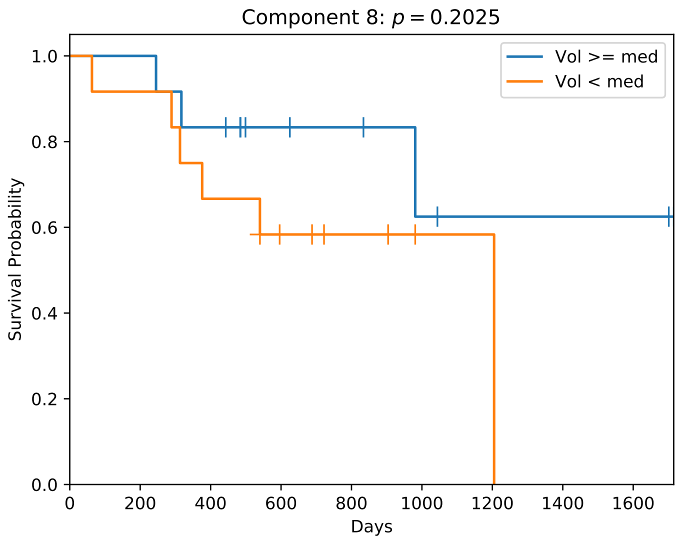


Figure 53

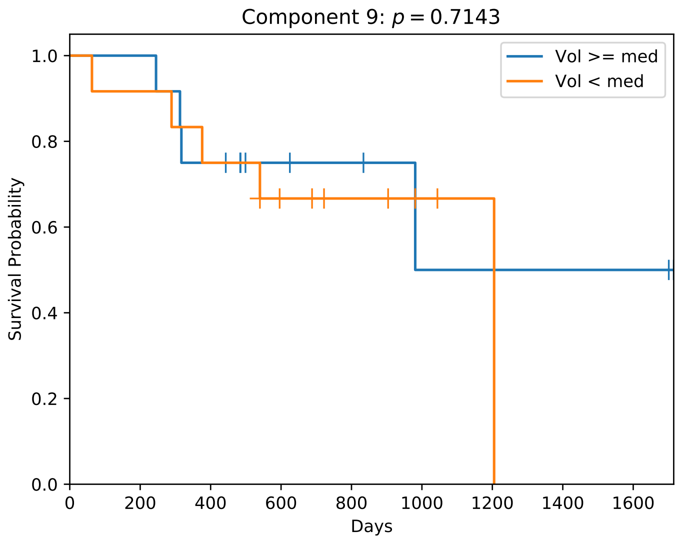


Figure 54

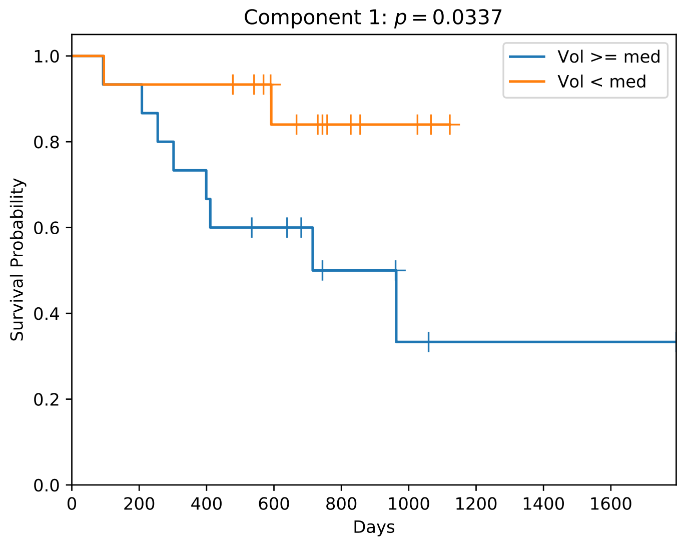


Figure 55

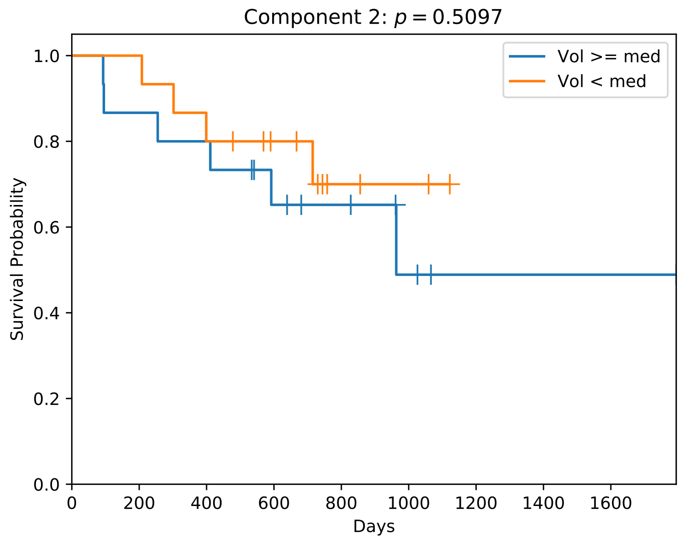


Figure 56

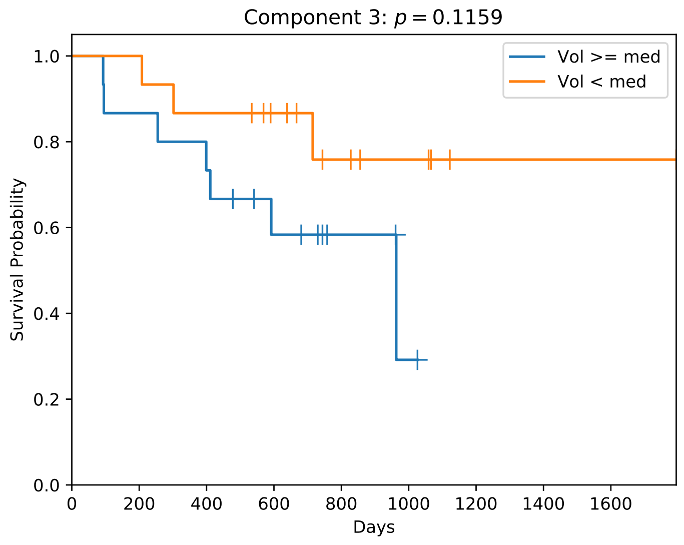


Figure 57

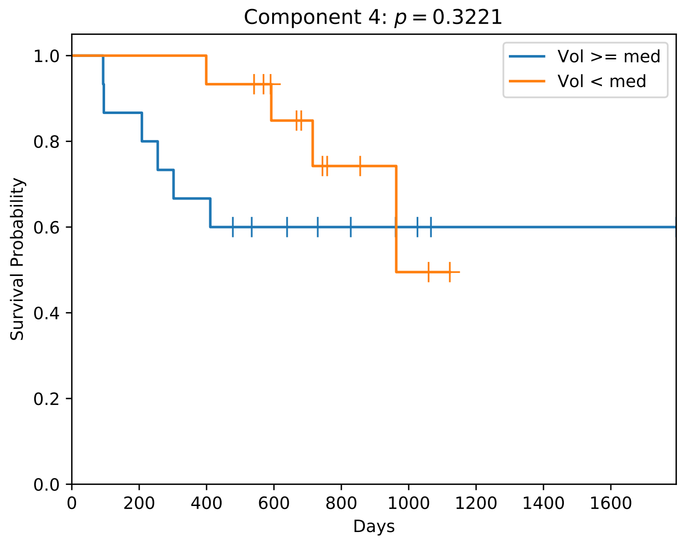


Figure 58

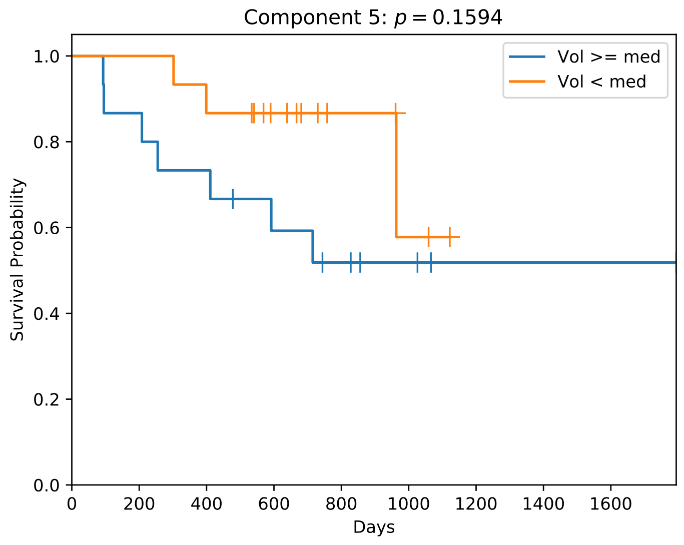


Figure 59

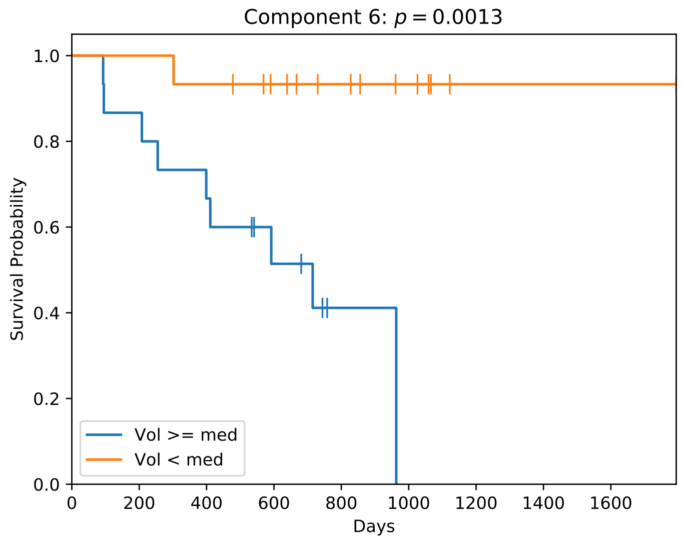


Figure 60

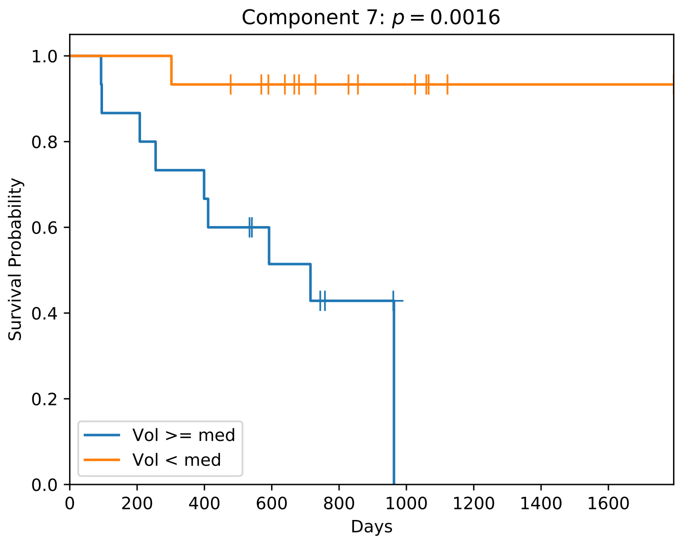


Figure 61

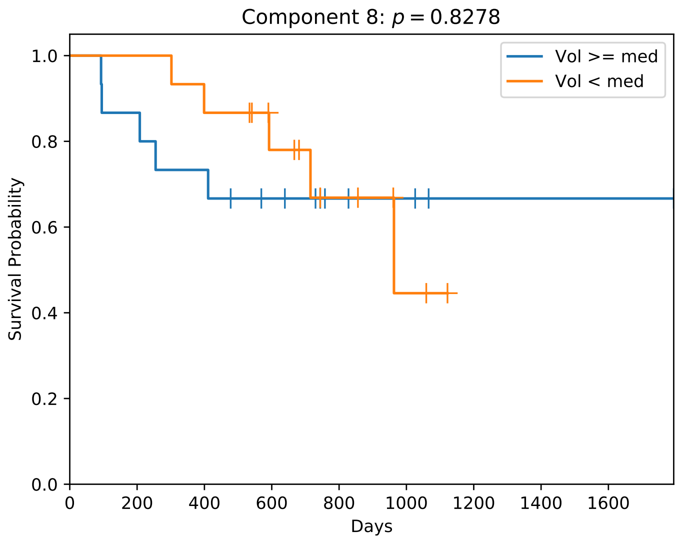


Figure 62

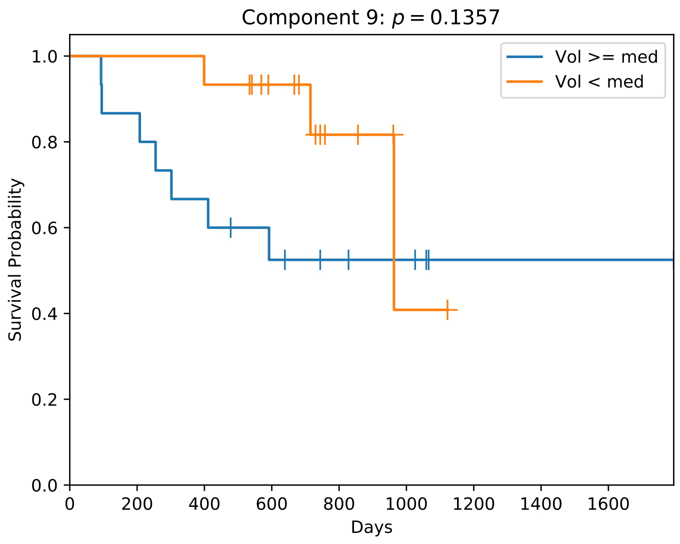

Component 9: $p = 0.1357$
Vol >= med
Vol < med
Survival Probability
Days
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
1200
1400
1600

Figure 63

A.10 Simulation study - Finding the optiman number of components in a GMM clustering solution using BIC on non-Gaussian clusters

Simulation study - Selecting the correct number of components in Gaussian Mixture Modelling using BIC on non-Gaussian clusters

May 27, 2019

```
In [83]: %matplotlib inline
         import numpy as np
         from sklearn.mixture import GaussianMixture
         import pandas as pd
         import matplotlib.pyplot as plt
         import seaborn as sns
         from joblib import Parallel, delayed
```

1 The Cluster

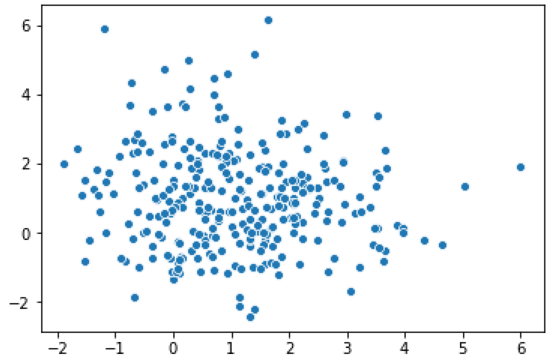
To make the example as simple as possible only a single cluster will be generated. This cluster will be made using an exponential distribution and a multivariate Gaussian distribution added together. The exponential distribution used is described by the rate $\lambda = 1$. The multivariate Gaussian distribution has mean vector (0,0) and covariance matrix equal to the identity matrix (spherical covariance).

2 300 samples

First the cluster is made using 300 samples. The GMM is used to cluster for number of components $k \in \{1, 2, \ldots 20\}$ and the BIC is calculated for each k. The process is repeated 5 times due to the randomness in the initialization of the components

```
In [111]: dist = np.random.exponential(scale=1.0, size=(300,2))
          dist2 = np.random.multivariate_normal([0,0],
                                        [[1, 0], [0, 1]], size=300)
          dist = dist+dist2
```

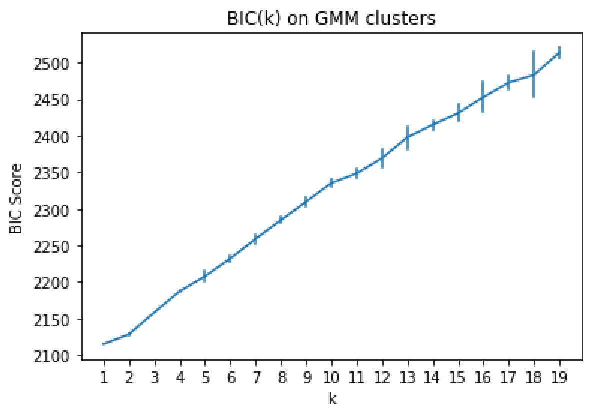
```
In [113]: sns.scatterplot(x=dist[:,0],y=dist[:,1])
          plt.show()
```



```
In [114]: gm = GaussianMixture()

In [115]: def fitGaussianMixture(X, k):
              return(GaussianMixture(n_components=k,
                                     max_iter=300, n_init=3).fit(X))

In [116]: kList = np.arange(1,20)
          reps = 5

In [117]: mixMods = Parallel(n_jobs=-2,
                             verbose=10)(delayed(fitGaussianMixture)(dist,
                                  k) for k in kList for r in range(reps))

[Parallel(n_jobs=-2)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=-2)]: Done   2 tasks      | elapsed:    2.5s
[Parallel(n_jobs=-2)]: Done   7 tasks      | elapsed:    2.6s
[Parallel(n_jobs=-2)]: Done  12 tasks      | elapsed:    2.7s
[Parallel(n_jobs=-2)]: Done  19 tasks      | elapsed:    2.9s
[Parallel(n_jobs=-2)]: Batch computation too fast (0.2000s.) Setting batch_size=2.
[Parallel(n_jobs=-2)]: Done  26 tasks      | elapsed:    3.1s
[Parallel(n_jobs=-2)]: Done  43 tasks      | elapsed:    3.9s
[Parallel(n_jobs=-2)]: Done  61 tasks      | elapsed:    5.1s
[Parallel(n_jobs=-2)]: Done  83 tasks      | elapsed:    8.0s
[Parallel(n_jobs=-2)]: Done  95 out of  95 | elapsed:   10.2s finished


In [118]: BICtest = [mixMod.bic(dist) for mixMod in mixMods]
```

2

```
In [119]: plt.figure()
          sns.lineplot(x=np.repeat(kList,reps), y=BICtest,
                       ci='sd', err_style='bars')
          plt.xlabel('k')
          plt.ylabel('BIC Score')
          plt.title('BIC(k) on GMM clusters')
          plt.xticks(kList)
          plt.show()
```



We can see that when using a limited number of samples the BIC find the correct number of clusters in the data being one.

3 1000 samples

Now using a dataset of 1000 samples from the same probability distribution function. The settings are the same as for the above.

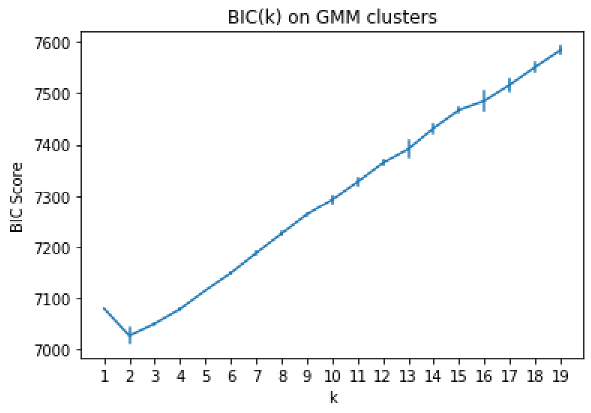
```
In [120]: dist = np.random.exponential(scale=1.0, size=(1000,2))
          dist2 = np.random.multivariate_normal([0,0], [[0, 1],[1,0]], size=1000)
          dist = dist+dist2
```

```
C:\Users\bendi\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: RuntimeWarning: covariance i
```

```
In [121]: mixMods = Parallel(n_jobs=-2,
                             verbose=10)(delayed(fitGaussianMixture)(dist,
                                  k) for k in kList for r in range(reps))

[Parallel(n_jobs=-2)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=-2)]: Batch computation too fast (0.0230s.) Setting batch_size=16.
[Parallel(n_jobs=-2)]: Done   2 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-2)]: Done  22 tasks      | elapsed:    1.5s
[Parallel(n_jobs=-2)]: Done  70 out of  95 | elapsed:    5.2s remaining:    1.8s
[Parallel(n_jobs=-2)]: Done  95 out of  95 | elapsed:    6.3s finished
```

```
In [122]: BICtest = [mixMod.bic(dist) for mixMod in mixMods]
```

```
In [123]: plt.figure()
          sns.lineplot(x=np.repeat(kList,reps), y=BICtest,
                       ci='sd', err_style='bars')
          plt.xlabel('k')
          plt.ylabel('BIC Score')
          plt.title('BIC(k) on GMM clusters')
          plt.xticks(kList)
          plt.show()
```



```
In [124]: mixMods[np.argmin(BICtest)].n_components
```

4

```
Out[124]: 2
```

We can now see that the BIC reaches a global minimum of 2 even though we know there is only one cluster in the dataset.

4 30000 samples

Now clustering 30000 samples from the same pdf and calculating the BIC.

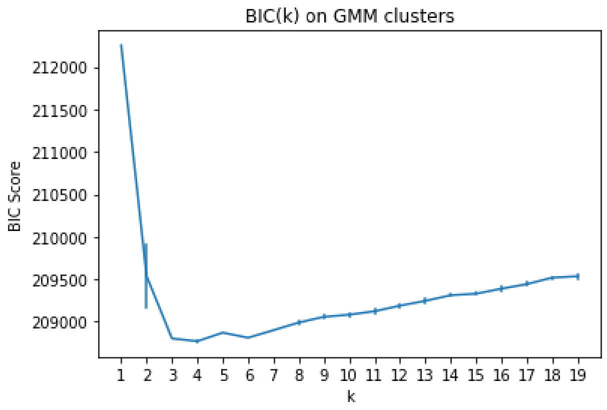
```
In [ ]: dist = np.random.exponential(scale=1.0, size=(30000,2))
        dist2 = np.random.multivariate_normal([0,0], [[0, 1],[1,0]], size=30000)
        dist = dist+dist2
```

```
In [103]: mixMods = Parallel(n_jobs=-2,
                             verbose=10)(delayed(fitGaussianMixture)(dist,
                                  k) for k in kList for r in range(reps))
```

```
[Parallel(n_jobs=-2)]: Using backend LokyBackend with 3 concurrent workers.
[Parallel(n_jobs=-2)]: Batch computation too fast (0.1309s.) Setting batch_size=2.
[Parallel(n_jobs=-2)]: Done   2 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-2)]: Done   8 tasks      | elapsed:    1.7s
[Parallel(n_jobs=-2)]: Batch computation too slow (2.0203s.) Setting batch_size=1.
[Parallel(n_jobs=-2)]: Done  18 tasks      | elapsed:    6.7s
[Parallel(n_jobs=-2)]: Done  27 tasks      | elapsed:   13.4s
[Parallel(n_jobs=-2)]: Done  34 tasks      | elapsed:   21.5s
[Parallel(n_jobs=-2)]: Done  43 tasks      | elapsed:   39.7s
[Parallel(n_jobs=-2)]: Done  52 tasks      | elapsed:   59.4s
[Parallel(n_jobs=-2)]: Done  63 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-2)]: Done  74 tasks      | elapsed:  2.1min
[Parallel(n_jobs=-2)]: Done  87 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-2)]: Done  95 out of  95 | elapsed:  3.3min finished
```

```
In [104]: BICtest = [mixMod.bic(dist) for mixMod in mixMods]
```

```
In [105]: plt.figure()
          sns.lineplot(x=np.repeat(kList,reps), y=BICtest,
                       ci='sd', err_style='bars')
          plt.xlabel('k')
          plt.ylabel('BIC Score')
          plt.title('BIC(k) on GMM clusters')
          plt.xticks(kList)
          plt.show()
```



```
In [110]: mixMods[18].n_components

Out[110]: 4
```

We see that the BIC now has it minimum at 4 components.

5 Discussion

The reason why we see that the number of components predicted by the BIC when more samples are included in the dataset is due to the fact that the BIC consistent in selecting the true model. That is when the number of samples $n \to \infty$ the probability of selecting the true model $\to 1$ given it is among the candidate models considered [1, 2]. The natural reason behind why we do not see a convergence of the BIC towards one component as the number of samples increase is here due to the fact that the true model is not part of the candidate models considered.

In many clustering task when using GMM the components are rarely ensured to be purely gaussian. Thus when clustering based on a large number of samples, the same trend as highlighted in this small simulation study might appear. A solution for this problem can be to use other mixtures where the true model is in the candidates considered. Another solution find a reasonable number of components instead of best solution based on the BIC. That is to acknowledge that the true model likely is not in the candidates tested.

As stated by Xu et al [3]: "The goal of the clustering is the separate a finite unlabelled data set into a finite discrete set of "natural", hidden data structures, rather than to provide an accurate characterization of unobserved samples generated the same probability distribution." and as stated by Everitt [4] that a clustering solution should not be based on its usefulness, rather than by in terms of whether it is true or false.

References

[1] S. I. Vrieze. "Model selection and psychological theory: a discussion of the differences between the Akaike information criterion (AIC) and the Bayesian information criterion (BIC)". In: *Psychol Methods* 17.2 (June 2012), pp. 228–243.

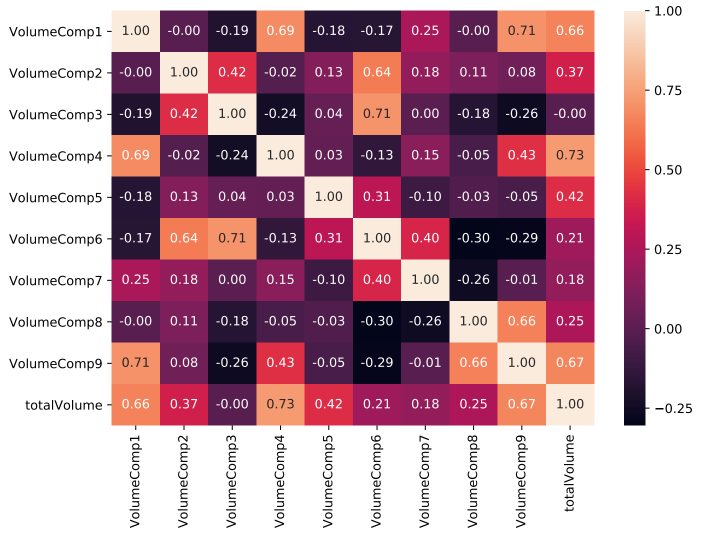
[2] Yuhong Yang. "Can the strengths of AIC and BIC be shared? A conflict between model indentification and regression estimation". In: *Biometrika* 92.4 (2005), pp. 937–950.

[3] Rui Xu and Donald C. Wunsch. *Clustering*. IEEE Series on Computational Intelligence. Wiley-IEEE Press, 2009. ISBN: 9780470276808. URL: http://search.ebscohost.com/login.aspx?direct=true&db=e230xww&AN=254099&site=ehost-live.

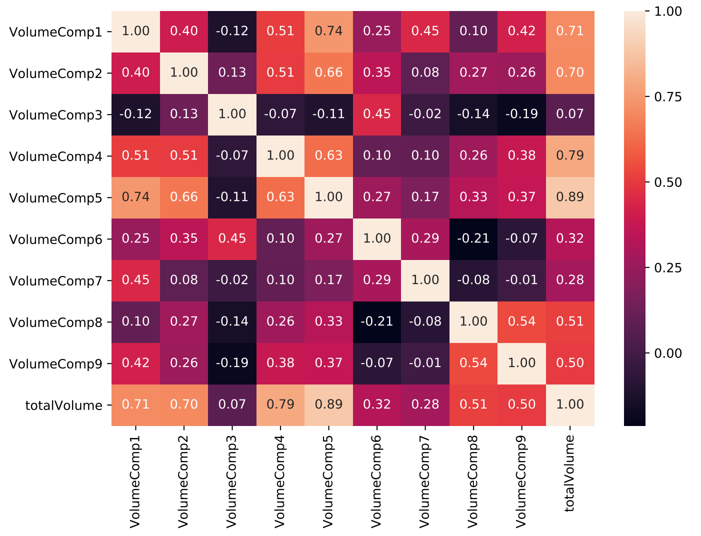
[4] Brian S Everitt. *Cluster Analysis*. eng. 5th edition. Vol. v.886. Wiley Series in Probability and Statistics. Chicester, 2010. ISBN: 1-280-76795-2.

A.11 Correlation heatmaps

In the following subsection the Pearson correlation coefficient is calculated between the volume of each component including the total volume. The results are presented as heatmaps. This is performed both for the CRT and the No CRT treatment groups and the results are presented in figure 64a and 64b respectively. A strong correlation (Persons correlation coefficient > 0.7) was found between certain volume components and between certain volume components and the total volume.



(a) CRT treatment group



(b) No CRT treatment group

Figure 64: Correlation matrix for the volume components and the total volume for both treatment groups. The correlation metric used is the Pearson correlation coefficient.

A.12 Dice-Sørensen Coefficient

The Sørensen-Dice similarity coefficient (DSC) [1] given two sets $\boldsymbol{X}$ and $\boldsymbol{Y}$ can be defined as

$$\mathrm{DSC} = \frac{2|\boldsymbol{X} \cap \boldsymbol{Y}|}{|\boldsymbol{X}| + |\boldsymbol{Y}|}. \tag{64}$$

The coefficient ranges from zero, when there is no overlap between the elements of the two sets, to one when there is completer overlap between the two sets.

References

[1] Lee R. Dice. "Measures of the Amount of Ecologic Association Between Species". In: *Ecology* 26.3 (1945), pp. 297–302. DOI: 10.2307/1932409. eprint: https://esajournals.onlinelibrary.wiley.com/doi/pdf/10.2307/1932409. URL: https://esajournals.onlinelibrary.wiley.com/doi/abs/10.2307/1932409.