Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Joakim Olsen

Measuring Summary Quality using Weak Supervision

Master's thesis in Applied Physics and Mathematics
Supervisor: Arild Brandrud Næss
January 2021

NTNU
Norwegian University of
Science and Technology

Joakim Olsen

Measuring Summary Quality using Weak Supervision

Master's thesis in Applied Physics and Mathematics
Supervisor: Arild Brandrud Næss
January 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

In this work, we analyse real estate condition reports and their corresponding summaries. Studies have suggested that many real estate buyers do not bother to read the full reports, and only read the summaries instead. This is problematic considering the following two facts: Firstly, we are aware that some of the summaries are not very good, and secondly, as many as 10% of real estate transactions end in conflict. We do not know how many low-quality summaries there are, but since the conflict rate is so high, we wish to investigate the extent of this problem. Hence, the objective of this work is to explore ways to automatically measure summary quality in an objective way, by using mathematical, statistical and machine learning methods. The objective is further to investigate the general summary quality for real estate condition reports, to determine whether poor summary quality can be a source of conflict.

We perform our analysis on a dataset of $96\,534$ real estate condition reports. We would like to make use of supervised learning methods, but the dataset is unlabelled. To remedy this challenge, weak supervision is employed. Thus, we first make a label model by using the weak supervision system Snorkel. From this label model, a labelled dataset of $81\,195$ real estate condition reports is obtained.

We then propose and implement various supervised model architectures for measuring summary quality. In particular, we investigate an approach where we map textual reports and summaries to a conceptual summary content space. In this vector space, the embedded reports and summaries should contain only key parts of the full, semantic content, such that summary quality can be measured by the cosine similarity between the embedded report and summary. We create such models by using the embedding techniques LSA, Word2vec and Doc2vec in combination with deep learning architectures like Feed-Forward Neural Networks, LSTM and CNN.

Our models are then trained on the previously obtained weak supervision labels. These labels are formulated as binary signals of quality, but we want our summary quality models to return a continuous quality score. To obtain this property, we construct an appropriate training objective, where we use a variation of the cosine embedding loss function.

Model performances are then evaluated on the weak supervision labels. Since the labels are binary signals of quality, we report the results by classification scores. In addition, we investigate the distribution of quality measures from the various models to investigate if they behave as requested. In general, we observe a substantial performance increase for all our weakly supervised models, compared to using unsupervised methods. In particular, we obtain a maximum accuracy of 89.5% for CNN-based models, compared to 72.6% for the best unsupervised model. Furthermore, by inspecting the distribution of quality measures, we find that models based on linear Feed-Forward Neural Networks and CNNs obtain the properties we request for a summary quality model.

Finally, we use the obtained models to measure the general summary quality in our complete dataset of $96\,534$ real estate condition reports. The results indicate that approximately 30% of the reports have a bad summary. Considering the fact that many only read the summaries, we therefore conclude that the high amount of bad summaries is likely a source of conflict in real estate transactions.

Samandrag

I dette arbeidet analyserer vi tilstandsrapportar for bustad, og deira samandrag. Studiar har antyda at mange kjøparar av bustad ikkje tek seg tid til å lese heile tilstandsrapportar, og berre les samandrag i staden. Dette er problematisk dersom vi tek i betraktning følgjande to fakta: For det første er vi klar over at nokre samandrag ikkje er særleg gode, og for det andre endar så mange som 10 % av bustadsal i konflikt. Vi veit ikkje heilt kor mange samandrag som faktisk har lav kvalitet, men sidan konfliktnivået er så høgt, så ynskjer vi å undersøke omfanget av dette problemet. Målet med denne oppgåva er derfor å utforske metodar for å automatisk måle kvaliteten til samandrag på ein objektiv måte, ved å bruke matematiske og statistiske metodar, samt maskinlæringsmetodar. Målet er vidare å undersøkje den generelle kvaliteten på samandrag, for å avgjere om dårleg kvalitet kan vere ei kjelde til konflikt.

Vi utfører vår analyse på eit datasett med 96 534 tilstandsrapportar for bustad. Vi ynskjer å bruke overvaka læring (eng: supervised learning), men datasettet vårt manglar ein "fasit" (informasjon om kvaliteten på ulike samandrag). For å handtere denne utfordringa tek vi i bruk *weak supervision*. Dermed lagar vi først ein modell for å lage fasit ved hjelp av *weak supervision*-systemet Snorkel. Frå denne modellen får vi ut eit datasett med fasit for 81 195 tilstandsrapportar.

Vi foreslår og implementerer så ulike overvaka modellarkitekturar for å måle kvaliteten på samandrag. Nærare bestemt undersøkjer vi ei tilnærming der vi avbilder teksten i rapportar og samandrag til eit vektorrom for samandragsinnhald. I dette vektorrommet burde vektoriserte rapportar og samandrag berre innehalde nøkkelinformasjon som er relevant for å måle kvalitet. Dermed kan kvaliteten målast som cosinus-likskapen mellom den vektoriserte rapporten og samandraget. Vi lager slike modeller ved å bruke vektoriseringsmetodane LSA, Word2vec og Doc2vec i kombinasjon med djup læringsarkitekturar som *feed-forward* nevrale nettverk, LSTM og CNN.

Desse modellane blir så trena på fasit-datasettet. Fasiten er formulert som binære kvalitetssignal, men vi ynskjer at modellane våre skal returnere ein kontinuerleg kvalitetsverdi. For å oppnå dette konstruerer vi eit passande treningsmål, der vi tek i bruk ein tapsfunksjon som baserer seg på cosinus-likskap, nemleg *cosine embedding loss*.

Prestasjonen til dei ulike modellane blir deretter vurdert på fasiten vi har fått frå *weak supervision*. Sidan denne fasitan består av binære kvalitetssignal, rapporterer vi resultata i form av klassifiseringsscorar. I tillegg undersøkjer vi fordelinga av kvalitetsmål frå dei ulike modellane for å finne ut om dei oppfører seg slik vi ynskjer. Generelt så observerer vi ei betydeleg auke i prestasjonen for modellane som er trena på fasit-datasettet, når vi samanliknar med uovervaka metodar. Nærare bestemt observerer vi ei maksimal treffsikkerheit på 89,5 % for CNN-baserte modellar, medan den beste uovervaka modellen får ei treffsikkerheit på 72,6 %. Ved å vidare undersøkje fordelinga av kvalitetsmål for dei ulike modellane, observerer vi at modellane basert på lineære *feed-forward* nevrale nettverk og CNN får dei eigenskapane vi ynskjer at kvalitetsmodellar skal ha.

Til slutt bruker vi modellane vi har laga til å måle den generelle kvaliteten på samandrag for det fullstendige datasettet med 96 534 tilstandsrapportar. Resultata indikerer at omtrent 30 % av tilstandsrapportane har eit dårleg samandrag. Tatt i betraktning at mange berre les samandraga, kan vi konkludere med at den store mengda dårlege samandrag sannsynlegvis er ei kjelde til konflikt ved kjøp av bustad.

Preface

This master's thesis concludes my time as a student at the Norwegian University of Science and Technology (NTNU), and completes my master's degree within the study programme Applied Physics and Mathematics, with main profile in Industrial Mathematics and specialization in statistics and machine learning. The work of this thesis has been carried out during the fall of 2020.

I would like to thank my supervisor, Arild Brandrud Næss, for giving me the opportunity to work with a very interesting research topic. Throughout this work, I have had a lot of freedom to explore my own ideas and interests, which has made the work very rewarding. I am also grateful for all the great guidance I have been given whenever I have needed it. I would also like to thank my co-advisor, Jo Eidsvik, who became my advisor for bureaucratic reasons, but who, nevertheless, stepped in and gave me very useful feedback.

Furthermore, I would like to thank the team at Vendu for facilitating the research topic, and for providing me with the data. In particular, I would like to thank Aleksander Bai and Annabelle Redelmeier for helping me with domain knowledge and useful data insight. I must also direct a big thanks to Pierre Lison at the Norwegian Computing Center (NR), who gave us the idea of using weak supervision in the first place, and also provided me with useful feedback.

Finally, I would like to thank my friends for five wonderful years as a student, and my incredible girlfriend Julie Røste for support, help, and for listening to my endless blabber about memory friendly iterators, code bugs, embedding techniques and whatnot. You all have made my time as a student both educational and very fun.

Joakim Olsen
Trondheim, January 2021

Abbreviations

AAN ACL Anthology Network.

BERT Bi-directional Encoder Representations from Transformers.

CNN Convolutional Neural Network.

CPM Concept-Project Matching.

D2v Doc2vec, which is the collective name of Paragraph Vector - Distributed Memory and Paragraph Vector - Distributed Bag-of-Words.

EL Embedding Layer.

EmbLayer Embedding Layer.

FFN Feed-Forward Neural Network.

FN False Negatives.

FP False Positives.

IDF Inverse Document Frequency.

LDA Latent Dirichlet Analysis.

LIKS Long words readability score.

LinTrans Linear Transformation.

LSA Latent Semantic Analysis.

LSTM Long Short-Term Memory.

LT Linear Transformation.

MSRP Microsoft Research Paraphrase.

NLP Natural Language Processing.

OVR Unique words readability score.

PV-DBOW Paragraph Vector - Distributed Bag-of-Words.

PV-DM Paragraph Vector - Distributed Memory.

ReLU Rectified Linear Unit.

RNN Recurrent Neural Network.

STS Semantic Textual Similarity.

TF Term Frequency.

TF-IDF Term Frequency-Inverse Document Frequency.

TG Condition degree.

TN True Negatives.

TP True Positives.

TSM Topic Space Matching.

W2v Word2vec, which is the collective name of Continuous Bag-of-Words and Continuous Skip-Gram.

Table of Contents

Chapter 1

Introduction

If you have ever bought real estate, then you have probably read a real estate condition report. Then you also know how long, technical and tedious such reports are to read. This is one of the reasons why the real estate condition reports also have a corresponding summary. One could argue that the longer and more difficult a report is, the more important it is to have a good summary.

And if you have skipped reading a condition report because it was too tedious, you should know that you are not alone. As we shall see, there are, in fact, many who do not bother to read the entire report. They must then rely on the summary to give them crucial information about the condition of the real estate they are buying. With this insight, consider the following example of an actual summary of a condition report: "Boligen er i god stand, kun enkelte anmerkninger."[1] It is clear that if there are summaries as little informative as this one, it is problematic that many buyers read only the summary rather than the full report.

The above example illustrates a fact that is the foundation of this work: There is interest in measuring summary quality for real estate condition reports in a fast, scalable and objective way. That will be the objective of this work. Hence, this work is a study of how large-scale summary quality can be measured for the real estate domain, by using mathematical, statistical and machine learning methods.

We will perform our analysis on an unlabelled dataset of real estate condition reports. This is challenging when working with statistical and machine learning methods, since these are data-driven, and often require labelled samples to learn from. To remedy this challenge, weak supervision will be employed to create a labelled dataset, such that weakly supervised learning can be applied. Thus, this work is also a study of how weak supervision can be used to improve performance in a setting where we traditionally would only be able to use unsupervised learning methods.

[1] English translation: "The real estate is in good condition, only a few remarks."

1.1 Background

Vendu is a startup company working with intelligent real estate solutions. In the fall of 2017, Vendu initiated a cooperation with Norsk Takst,[2] with the objective of analysing data from real estate condition reports.[3] This initiative has developed into a bigger research project where the goal is to make it easier for buyers to obtain and understand necessary information when buying real estate. The motivation behind this project comes from the fact that the buyer in a real estate transaction has to collect, read and understand an overwhelming amount of information. The aforementioned real estate condition report is a crucial part of this information.

A real estate condition report is a thorough and detailed description of the technical condition of a piece of real estate. In transactions, the condition report contains important information for the involved parts, especially the buyer. The report is, however, rather long and technical, and is therefore not an easy read. Studies have suggested that less than 50% of buyers actually read these condition reports.[4] The condition reports also have a corresponding summary, and in light of the above information, it is clear that this summary is important. In particular, many might resort to reading the summary only, in which case it needs to be of high quality.

We are, however, aware that some summaries are of low quality, as some of them contain very little information in general, and therefore summarize their condition report poorly. The example summary in the introduction above is one example of this. We do not know how many there are, but if there is a substantial amount of bad summaries, it is clear that a reading rate below 50% can be a source of conflict. And in fact, Huseiernes Landsforbund reported in 2017 that 10% of transactions did end in conflict.[5] This number is too high, and measures should be taken. One of these measures is to investigate the quality of the summaries in an objective way. In particular, we want to be able to identify bad summaries. We can then investigate how many bad summaries there are, and decide whether measures should be taken to improve summary quality.

The objective of this work is to create models that can analyse summary quality in more depth. The objective is further to analyse the summary quality across a dataset of real estate condition reports. This is a problem that belongs to the field of *Natural Language Processing* (NLP), which will be briefly introduced in the next section. Although NLP is a field within computer science, many of the popular models within it are mathematical and statistical in nature. Particularly, the use of artificial neural networks and deep learning have pushed performance on many tasks in the last years.

[2] Norsk Takst is the Norwegian tariff organization for real estate. `https://www.norsktakst.no/`

[3] By real estate condition report, we refer to the Norwegian "Tilstandsrapport". `https://www.norsktakst.no/norsk/finn-takstmann/bolig-tilstand/`

[4] See for example the article "Få leser tilstandsrapporter under boligjakten" in *Dagens Næringsliv*, 13.06.2017. `https://www.dn.no/privatokonomi/bolig/boligkjop/fa-leser-\tilstandsrapporter-under-boligjakten/2-1-102359`

[5] See the article "Konfliktnivået ved bolighandel må ned" by Huseiernes Landsforbund, 12.06.2017. `https://www.huseierne.no/nyheter/konfliktnivaet-ved-bolighandel-ma-ned/`

1.2 Natural Language Processing

NLP is a branch of artificial intelligence that deals with the processing of natural languages (e.g., English and Japanese). These are languages that have evolved naturally, in contrast to constructed languages (e.g., programming languages). Natural languages consist of several complex elements, like grammatical rules, a spoken language formed by a collection of sounds, and a written language formed by a collection of signs. There are a huge number of different tasks related to NLP, with some examples being optical character recognition, machine translation and automatic summarization.

The task of this work is to measure summary quality. This is a problem that has not been studied all that much, but it is very similar to a subfield of NLP called *document similarity*, which has been studied extensively. It is therefore appropriate with a short introduction to document similarity.

1.2.1 Document similarity

In linguistic theory, the meaning of a document, that is, the message that the document is trying to convey, is referred to as its *semantics*. The goal in document similarity is to measure how similar the semantics of documents are. Thus, the semantics of each document must in some way be modelled. This is generally done by making so-called *document embeddings*. This is a very central concept in this work, and will therefore be explained further in the following.

Document Embeddings

Document embeddings are mappings from documents to numerical vectors. In applications where the semantics are of interest, the idea is to use a mapping such that the resulting numerical vector represents the semantics of the input document. Hence, the dimensions of the numerical vector should correspond to different aspects of meaning, while the values of the vector elements should reflect to what extent these aspects are present in a document.

Once document embeddings have been obtained for the documents of interest, it is easier to measure document similarity. This is generally done by applying an appropriate distance measure between the document vectors. Thus, documents will be modelled as semantically similar if their semantic document embeddings are close to each other in the corresponding vector space.

There are many document embedding techniques, and they all map documents into unique vector spaces. However, some of these vector spaces have conceptual similarities. In order to better understand the embedding techniques, some of these spaces will now be further explored.

The Semantic Space

A semantic space is a vector space where a mathematical distance is equivalent to a measure of semantic similarity. The oldest, and perhaps most intuitive type of semantic space is the *word space*. This is a vector space where the dimensions correspond to the words in the vocabulary, and the vector values for a document creates a relation between the

document and the various words. The full vocabulary in a collection of documents is generally very large, and thus, the word space is very high dimensional. In this vector space, documents are modelled as semantically similar if they contain many of the same words.

Another very intuitive way of modelling semantics is by mapping documents to a *topic space*. In the topic space, the dimensions correspond to different topics, while the vector values for a document embedding relate the document to the different topics. Topic modelling is a well-developed branch of NLP, with robust and well-performing baseline models. The topics are generally not pre-defined, but instead *latent*, hidden concepts that the models are constructed to uncover. In this vector space, documents are modelled as semantically similar if they have a similar relation to the various topics.

In recent years, models based on deep learning have become increasingly popular for their ability to solve complex problems. This trend has reached NLP as well, and many new document embedding techniques are based on deep learning. Such techniques are mapping documents to a general semantic *feature space*. This space is not as intuitive as the word space or topic space, because of the black-box nature of deep learning models: These are performance-driven only, and it is therefore impossible to know exactly what the resulting features in the feature space represent. However, by using the resulting document embeddings for semantics-related tasks, it is clear that a good performance indicates that the feature space indeed forms a good representation of the semantics. In general, a *feature vector* is a numerical vector that represents an object, where the vector values correspond to various features. Word vectors and topic vectors are thereby also examples of feature vectors.

Now that the idea of document embeddings and semantic feature vectors have been introduced, we can start to sketch a solution to the real estate summary quality problem.

1.3 Solution Sketch to the Summary Quality Problem

As will be discussed in the next chapter, we are not aware of any previous work related to summary quality that is relevant for this work. Therefore, we will instead look to the field of document similarity for inspiration. After all, measuring document similarity is a quite similar task to that of measuring summary quality. The most important task for a summary is to reproduce the semantic content of the main document. Thus, a good summary should be semantically similar to its real estate condition report. This motivates us to measure summary quality by using some sort of document similarity measure. However, instead of using a general semantic vector space, we will explore the idea of using a different vector space that is specialized in measuring summary quality. This will be referred to as the *summary content space*.

1.3.1 The Summary Content Space

A real estate condition report contains a very large amount of information. A summary should not contain all of this information, or else it would become just as long and technical as the full report. Instead, the summary should contain only key parts of the full report. There are also other qualities that a good summary could have that are unrelated to the semantic similarity, like a language that is not too difficult and technical. A standard

measure of semantic document similarity is therefore not really suited for the task at hand, since this measure does not pay attention to which parts are important, and which parts are not.

Thus, instead of mapping reports and summaries to a general semantic vector space, it would be better if we could map the reports and summaries to a specific vector space that only includes the key information that a good summary should have. In such a vector space, embedded reports and summaries would only be close if the summaries actually contained this key information. In this space, a summary could also be moved further away from its report if it contains too much irrelevant information.

In this work, we will attempt to solve the summary quality problem by mapping reports and summaries to a vector space with these properties. This space will be referred to as the summary content space. Summary quality will then be measured by applying an appropriate distance measure between embedded reports and summaries. The main effort of this work will thereby be to develop models that can make such mappings to the summary content space. Before proceeding with specific solution proposals, the main challenges of this task will be introduced.

1.3.2 Challenges

There are mainly two big challenges with the above solution sketch:

1. There is no prior knowledge about summary quality. This means that the real estate condition reports are unlabelled, that is, there is no information about summary quality in the dataset.

2. Within the field of document similarity, most of the work is focused on shorter documents. However, the real estate condition reports are very long.

The first point above has a couple of very challenging implications. Firstly, this is a fact that, by traditional means, restricts us to using only unsupervised learning methods. And even though there are many powerful unsupervised methods to choose from, such methods are generally more suited for exploring data, rather than solving specific problems. In fact, supervised deep learning methods are becoming the new state-of-the-art on a wide range of problems, both within and outside the field of NLP. Ideally, we would like to apply supervised methods on the task at hand, but supervised methods require labelled data to learn from.

Secondly, the fact that the data is unlabelled makes model evaluation very difficult. It might be easy to construct any arbitrary measure of summary quality, yet, it is very difficult to determine whether this quality measure actually works as intended. To evaluate the constructed models, some knowledge about the truth is necessary.

In this work, the challenge of unlabelled data will be tackled by applying weak supervision. This is a rather new supervision concept, where labels are created from a set of rules, rather than manually by humans. This way, a large amount of labels can be made efficiently, but they are also expected to be noisy and more imprecise than manually made labels would be. Even so, weak supervision will allow us to apply supervised learning methods and evaluate models.

The second challenge of long documents also makes the summary quality problem difficult. We would like to look for inspiration for embedding techniques within the field of document similarity, but many embedding architectures are not really suited for very long documents. We must therefore be mindful in our choice of model architectures, such that the resulting models are able to capture necessary information from such long documents.

1.3.3 Weak Supervision

These days, more and more data is becoming available for machine learning to learn from, but unfortunately, as the amount of data grows, so does the amount of work required to label it. So much, in fact, that the work of labelling data is becoming the new bottleneck in developing many machine learning systems. Advanced supervised machine learning methods are there, ready and easy to use thanks to open-source libraries. However, without high-quality, labelled datasets to apply them on, they are of no use to us.

The task of measuring the summary quality for the real estate condition reports is a classic example of this. The data is there, but since the condition reports are so long and technical, it would require a tremendous amount of work by people with expert domain knowledge to manually make high-quality labels of summary quality. This is simply not an option in this work. As mentioned, this fact would traditionally restrict us to using only unsupervised methods.

Luckily for us, new methods are becoming available for tackling the challenge of labelling data as well. Weak supervision is one such method. Through this, expert domain knowledge can be applied by making labelling rules instead of manually labelling data samples, and thus, large amounts of data can be labelled with a much smaller effort.

The resulting weak supervision labels are likely to be of lower quality than a manually made set of labels would be. However, when training models on the weak supervision labels, we want to make sure that the machine learning models are given a more general input, without knowledge of the rules that were used to make the labels. Then, they should not be able to mimic the labelling rules, and thus, will have to find different, underlying patterns that can explain the weak supervision labels. And even if the labels are noisy and imperfect, the models should be able to pick up true patterns of summary quality, as long as the labels have at least some accuracy. In theory, by picking up the right patterns, the models might even become superior to the labels they are trained on.

1.3.4 Model Proposals

In this work, three main approaches will be investigated, all of which are based on mapping reports and summaries to the conceptual summary content space. Summary quality will then be measured by measuring the similarity between the embedded reports and summaries. The approaches will combine existing embedding techniques with supervised deep learning methods, where the latter will be trained on the weak supervision labels. The general idea behind these three strategies will now be introduced.

Semantic Feature Vectors as Input

A simple and natural approach is to make use of existing document embedding techniques. These should be able to capture the full semantic content of documents, which should be a good starting point for a summary quality model. The goal will then be to create a transformation from the semantic vector space to a summary content space.

In this approach, we must use semantic embedding techniques that are suitable for long documents. In this work, Latent Semantic Analysis (LSA), which is a topic modelling technique, and Doc2vec, which is an embedding technique based on artificial neural networks, will be applied. A transformation will then be made by sending the semantic document vectors through a fully connected Feed-Forward Neural Network (FFN). This way, we will obtain both linear and non-linear transformations from the full semantic vector space to a summary content space.

A Section-Based Approach

If we were to evaluate the quality of summaries as humans, we would probably proceed with a sentence- or section-based approach, since the point of a sentence in a summary generally is to summarize the content of one or more sentences in the original document. With this in mind, it makes sense to use a section-based approach.

Thus, we will split the report into sections, and the summary into sentences. Then, by applying an embedding technique to each sentence or section, the reports and summaries will be represented as sequences of semantic feature vectors. This way, models can distinguish sentences and sections from each other, and possibly learn a more informed measure of summary quality.

In this approach, the reports and summaries must be mapped from sequences of semantic feature vectors to the summary content space. This will be done using a Long Short-Term Memory (LSTM) network. This is a type of Recurrent Neural Network (RNN) that is frequently used in NLP on sequential data. Again, LSA and Doc2vec will be used as embedding techniques.

By splitting the long reports into sections, the documents to embed become much shorter. Thus, state-of-the-art embedding techniques like BERT (Devlin et al. 2018), which can only embed shorter documents, can be applied. We will not do that in this work, but it will be interesting to see if this section-based approach is promising for the summary quality problem. If that is the case, then a natural next step will be to apply more powerful embedding techniques than LSA and Doc2vec, which are mainly chosen for their ability to embed arbitrarily long documents.

Starting From Word Embeddings

As a final proposal, we will attempt to build a model based on *word embeddings*. Word embeddings are, similarly to document embeddings, numerical vectors that describe semantics. However, word embeddings describe the semantics of words instead of documents.

When using word embeddings as the starting point, it is important to keep in mind the length of the reports, and to choose an architecture accordingly. A common strategy in

NLP is to use an LSTM network over word embeddings, but since the documents in this case are so long, we do not expect RNNs to be the best approach. We will instead use a Convolutional Neural Network (CNN) over the word embeddings. This is a type of neural network that is often used in computer vision, but which has also given good results on text data.

For this approach, we require word embeddings for all the words in the vocabulary. In this work, we will use word embeddings trained on the weak supervision labels, as well as word embeddings from the word embedding technique Word2vec. Word2vec is a similar model to Doc2vec, but which creates word embeddings instead of document embeddings.

To summarize, in this work, weak supervision will first be employed on the real estate condition report dataset, in order to obtain noisy labels of summary quality. Then, various supervised architectures will be proposed, with the end goal of mapping reports and summaries to an appropriate summary content space. This way, summary quality can be measured by measuring similarity in the resulting vector space. The various architectures will first be evaluated on, and then applied to the real estate condition report dataset. Finally and hopefully, conclusions can be drawn about the general summary quality, and we can discuss whether or not measures should be taken to improve the summary quality.

In Chapter 2, previous work related to measuring summary quality will be presented. Then, the background theory of this work will be given in Chapter 3. This includes the theory behind relevant neural network architectures like FNN, LSTM and CNN, as well as relevant embedding techniques like LSA, Word2vec and Doc2vec. Weak supervision will also be thoroughly presented in Chapter 3. In Chapter 4, the experimental setup for this work will be given. This includes information about our dataset, how we implement weak supervision, and details about our model proposals for measuring summary quality. Then, our results will be presented and discussed in Chapter 5, while conclusions will be given in Chapter 6.

Chapter 2

Previous Work

In this chapter, we will first present previous work on summary quality. However, as we will see, most of this work is not very relevant to the problem at hand. Therefore, results on a similar task, namely document similarity, will also be presented. Finally, we will do a brief survey of previous work in weak supervision.

2.1 Previous Work on Summary Quality

Summary quality is a task within NLP that has not been given too much attention on its own. It has, however, been studied quite a lot in relation to a different task, namely *automatic text summarization*. In particular, the text summarization task requires an evaluation system for the proposed summarization methods. These systems must, by definition, be systems of summary quality.

Outside of the automatic text summarization context, however, there is little work on summary quality, and to the best knowledge of this author, there is no previous work on summary quality for real estate condition reports. The other most relevant previous work is instead found within the field of document similarity, which will be investigated in Section 2.2.

2.1.1 Summary Quality in Automatic Text Summarization

An overview of evaluation systems in automatic text summarization is given by Lloret, Plaza, and Aker (2018). Here, they distinguish between evaluating *readability*, *non-redundancy* and *content coverage*.

The points concerning readability and non-redundancy are important in the automatic text summarization context since automatically generated summaries can be terribly written from a grammatical perspective, even if the content is good. In our context, however, we can trust that the readability and non-redundancy of the summaries are decent since they are written by humans and not automatically generated.

The main focus in this work will therefore be on content coverage. Lloret, Plaza, and Aker (2018) present most of the content coverage evaluation systems that have been pro-

posed and used in automatic text summarization during the last two decades. Common for pretty much all of these systems, is that they compare the quality of generated summaries not with the original document, but instead with one or more reference summaries. This fact immediately makes such systems of no use in this work, since there are no reference summaries to compare the real estate summaries to.

The most used evaluation methods in the automatic summarization context, are various ROUGE-scores (Recall-Oriented Understudy for Gisting Evaluation). ROUGE-scores are based on finding matching n-grams, that is, co-occurring sequences of n words in the documents. An overview of various ROUGE-scores is given by Lin (2004). However, these are also generally applied between a generated summary and a reference summary, and they are therefore not meant to be used on a long report and a short summary. Therefore, this evaluation metric will not be investigated in this work.

Thus, twenty years of previous work on summary quality is of little use to us. We will therefore, instead, look to the field of document similarity for baselines and inspiration.

2.2 Previous Work on Semantic Similarity

There is a vast amount of previous work related to measuring the semantic similarity of documents. In this section, the general development, with some important baseline models, will briefly be presented. Results on a few tasks will also be given. This includes results on the Microsoft Research Paraphrase (MSRP) corpus (Dolan, Quirk, and Brockett 2004), the Semantic Textual Similarity (STS) benchmark (Cer et al. 2017), the Concept-Project Matching (CPM) task (Gong et al. 2019) and the ACL Anthology Network (AAN) data (Liu et al. 2017).

The MSRP corpus and STS benchmark contain pairs of short documents, the AAN data contains pairs of long, scientific reports, and the CPM data contains pairs of long descriptions and short summaries. All pairs are labelled by how similar they are, either by a binary signal of whether or not they are similar, or as a similarity score on a given scale which describes how similar they are. A collection of results on these datasets are given in Table 2.1. The results on the binary signals are reported by accuracy (acc.), precision (pre.), recall (rec.) and/or F_1-score as defined in (4.9), while the results on the similarity score labels are reported by Pearson correlation (r) which we assume to be known, or Spearman's rank correlation (r_s), which is defined among others by Liu et al. (2017). Note that for the AAN data, the labels are given both on a 5-level scale (5lev.) and as a binary signal (2lev.). In both cases, they use Spearman's rank correlation r_s to report performance.

The results on these datasets indicate how well the models are able to capture the semantics of the various documents. Since the real estate condition reports are so long, results on the CPM and AAN data are of particular interest.

2.2.1 The Old Baseline: Bag-of-Words

The bag-of-words approach, which will be introduced in Section 3.3.1, has been a solid baseline for a long time. In particular, Term Frequency-Inverse Document Frequency (TF-IDF), presented on page 25, has proven to be a tough baseline to beat. E.g., results using

TF-IDF are reported by Vrbanec and Meštrović (2020) for the MSRP corpus, and by Liu et al. (2017) for the AAN corpus. These are shown in Table 2.1. As the results show, TF-IDF appears to be a good baseline, without being particularly noteworthy.

Table 2.1: A collection of semantic similarity results on the MSRP corpus, AAN data, STS benchmark and CPM task. Results on the MSRP and CPM tasks are measured by classification scores, while the AAN and STS tasks are measured by correlation with the true similarity. In both cases, a higher score indicates better model performance.

Method	**MSRP**		**AAN**		**STSb**		**CPM**		
	acc.	F1	5lev.	2lev.	r	r_s	pre.	rec.	F1
TF-IDF	70.6	81.3	51.9	24.5					
LSA	73.6[6]	**81.8**[6]							
LDA	73.3[7]	80.9[7]	53.7	25.0					
Word2vec	69.0[6]	80.3[6]	51.7[6]	**40.4**[6]	56.5[6]		64.3[8]	73.5[8]	67.9[8]
Doc2vec	65.5	79.2	**54.1**	32.7	**64.9**		61.5	84.3	69.5
BERT	**76.0**					**79.2**			
TSM							**75.8**	**88.5**	**81.8**

2.2.2 A New Paradigm: Topic Modelling

The first high-performing topic modelling technique, LSA, was first introduced in 1990 (Deerwester et al. 1990). Later, in 2003, Latent Dirichlet Analysis (LDA) was introduced (Blei, Ng, and Jordan 2003). Both of these are very important unsupervised baseline models that have been frequently used, especially in data mining, since they can retrieve a lot of information from unlabelled data. They can, however, also be used to create document embeddings.

LSA and LDA are applied to the MSRP corpus by Rus, Niraula, and Banjade (2013). LDA is also included as a baseline on the AAN corpus by Liu et al. (2017). The results are given in Table 2.1 and show that LSA and LDA have given good results on document similarity tasks.

In the author's project thesis (Olsen 2020), both LSA and LDA were considered as document embedding techniques for the real estate domain. In this preliminary work, we achieved good results with LSA, but found to our surprise that LDA was not very well suited for embedding real estate condition reports. For this reason, we will not use LDA in this work. LSA will, on the other hand, be used as a document embedding technique, and will therefore be introduced in Section 3.3.

2.2.3 The New Baseline: Neural Networks

In the last ten years, a wide range of models based on artificial neural networks have been introduced. Such models have become the new baseline for a wide range of tasks, thanks

[6]Document vector obtained by averaging word vectors.
[7]Distances calculated by word matching.
[8]Distances calculated using Word Movers Distance.

to their ability to solve complex problems. The word-embedding technique Word2vec (Mikolov et al. 2013b) was one of the first neural network-based models that arrived, and soon after, a natural extension to documents arrived with Doc2vec (Le and Mikolov 2014).

Results for the MSRP corpus are given for Word2vec and Doc2vec by Vrbanec and Meštrović (2020). Both Word2vec and Doc2vec are also included as baseline models by Liu et al. (2017) for the AAN corpus. The results are given in Table 2.1. As the table shows, the results are not too convincing for the MSRP corpus. For the AAN data, however, Doc2vec outperforms the other models for the 5 level task, whereas averaging word vectors with Word2vec is surprisingly effective for the 2 level task.

Furthermore, Gong et al. (2019), include Doc2vec and Word2vec as baseline models for the CPM task. Gong et al. (2019) also propose their own model, which we refer to as Topic Space Matching (TSM). The results are given in Table 2.1 and show that their own model significantly outperforms Word2vec and Doc2vec.

Finally, results with Word2vec and Doc2vec for the STS benchmark are presented by Cer et al. (2017). These results are also presented in Table 2.1, and show that Doc2vec significantly outperforms Word2vec on this task. These results are based on the work of Lau and Baldwin (2016), where Word2vec and Doc2vec are compared to an n-gram baseline for two tasks: STS tasks across 5 domains, as well as the dataset of Hoogeveen, Verspoor, and Baldwin (2015). The authors find that both Word2vec and Doc2vec outperform the n-gram baseline for both tasks.

Lau and Baldwin (2016) further compare Doc2vec with two other state of the art models: Skip-Thought (Kiros et al. 2015) and Paragram-Phrase (Wieting et al. 2016). They find that Skip-Thought performs poorly for both tasks. They also find that Doc2vec outperforms Paragram-Phrase for the dataset of Hoogeveen, Verspoor, and Baldwin (2015), while this is reversed for the STS tasks. The Paragram-Phrase model is based on averaging word vectors, and the documents in the dataset of Hoogeveen, Verspoor, and Baldwin (2015) are longer than the documents for the STS tasks. Lau and Baldwin (2016) therefore argue that the strategy of averaging word vectors is more suitable for shorter documents.

In summary, both Word2vec (when averaging word vectors) and Doc2vec have obtained good results on document similarity tasks, both for long and short documents. Some of the results do, however, indicate that Doc2vec is more suitable than Word2vec for longer documents. In the author's project thesis (Olsen 2020), Word2vec and Doc2vec were also investigated as document embedding techniques for the real estate domain. Doc2vec achieved good results, but as with LDA, we found that Word2vec was not very well suited for embedding real estate condition reports. These observations are in accordance with the arguments of Lau and Baldwin (2016), and for this reason, Word2vec will only be used as a word embedding technique in our work, while Doc2vec will be used as a document embedding technique in addition to LSA. Both Word2vec and Doc2vec will therefore be presented in Section 3.3.

2.2.4 State of the Art: RNN, LSTM and Attention

In the last few years, new neural network architectures have been proposed, pushing the state of the art even further. Particularly, mechanisms like recurrent neural networks, LSTM and attention have led to substantial progress, and there have been too many publications to include them all. Instead, we will restrict ourselves to what is arguably the most

successful of these new methods, namely the Bi-directional Encoder Representations from Transformers (BERT).

BERT was first introduced by Devlin et al. (2018). It is first and foremost a word embedding technique, but BERT can also be used as a document embedding technique for documents shorter than 500 words. Reimers and Gurevych (2019) report results on the MSRP data, as well as on the STS benchmark. These are given in Table 2.1 and show that BERT significantly outperforms all the other models on these two tasks.

Extensions of BERT are among the current state-of-the-art for a wide range of tasks, also in the document similarity domain. BERT will, however, not be considered in this work, due to the complexity of training this model. There are pre-trained versions available, but there are not many Norwegian versions that have been thoroughly tested, and it is unclear if they would be adequate for the real estate domain.

2.3 Previous Work on Weak Supervision

Weak supervision was first introduced by Ratner et al. (2016), and later expanded by Ratner et al. (2017) and Ratner et al. (2019). The effectiveness of using weak supervision is also investigated by Ratner et al. (2017). In their work, they apply weak supervision on four relation extraction tasks and one sentiment analysis task, which are tasks within the field of NLP, as well as on one image classification task.

The results show that the use of weak supervision substantially outperforms other alternatives when supervised learning is not available, by an average performance increase of 132%. The results further show that while the weak supervision labels give pretty good results on their own, the results can be improved even further by training supervised methods on the weak supervision labels. This shows that when training supervised learning methods on the noisy and imprecise weak supervision labels, the methods might actually learn to pick up other and better patterns than the labels do, and thus, outperform the labels that the models are trained on. Finally, the results show that the weak supervision performance approaches the performance of standard supervised learning, and comes within an average of 3.6% of the performance whenever hand-labelled training sets are available.

Ratner et al. (2017) also describe a workshop where they compare 7 hours of work on weak supervision for one person to 7 hours of work with hand-labelling a training set. The results from this workshop show that the weak supervision approach substantially outperforms the standard supervised approach by an average performance increase of 45.5%. This shows that the weak supervision approach might be a more efficient way to spend time when building machine learning systems, rather than creating traditionally labelled datasets.

Currently, there is little work with weak supervision related to document similarity or summary quality, but weak supervision has been applied to a wide range of tasks. Promising results have also been reported by industrial giants like Google (Bach et al. 2019) and Intel (Bringer et al. 2019).

To the best of our knowledge, there is no previous work that addresses the possible downsides of using weak supervision. So far, the results seem to indicate that when using weak supervision, the performance will improve compared to using other, unsupervised alternatives. The results also indicate that the weak supervision performance might approach

the performance of using hand-made labels. However, we cannot from these indications conclude that this will always be the case.

In this work, we will not have any ground truth labels to compare the weak supervision labels to, and thus, we must be careful with how we interpret our results. The previous good results obtained with weak supervision are only an indication that we with weak supervision might obtain good results for our task as well. Therefore, any conclusions we can draw about summary quality in this work can only be indicative in nature.

Chapter 3

Theory

In this chapter, the relevant theoretical background will be presented. First, a formal measure of document similarity, given document embedding vectors, will be introduced. Then, the theory and intuition behind relevant deep learning methods will be briefly presented. This includes fully connected FFN, LSTM and CNN. Then, the theory and intuition behind relevant embedding techniques will be given. This includes LSA, Word2vec and Doc2vec. Finally, the theory behind weak supervision will be thoroughly presented.

3.1 Document Distance

The notion of document embeddings was introduced in Section 1.2.1. These embeddings enable us to define a formal measure of document similarity. Since the objects of interest now are numerical vectors, similarity can be measured by a mathematical distance.

In mathematics in general, the Euclidean distance is by far the most widely used distance measure. In NLP, however, results often show that this is not the best choice of distance measure. Instead, *cosine similarity*, which measures the cosine of the angle between the document embedding vectors is commonly used.

The objective of this work is to measure the quality of summaries. This will be done by first mapping reports and summaries to the conceptual summary content space, which was described in Section 1.3.1. Then, the quality of the summaries will be measured by the similarity between the embedded report and summary. More specifically, the cosine similarity measure will be applied.

3.1.1 Cosine Similarity

The *cosine similarity* measures the cosine of the angle θ between two non-zero vectors $\mathbf{a}$ and $\mathbf{b}$ in an inner product space. Mathematically, it is defined as

$$\cos \text{sim}(\mathbf{a}, \mathbf{b}) = \cos\theta = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}||\,||\mathbf{b}||}, \tag{3.1}$$

where $\mathbf{a} \cdot \mathbf{b}$ is the dot product between $\mathbf{a}$ and $\mathbf{b}$ and $||\cdot||$ is the Euclidean norm. This gives values of $\cos\theta$ limited by $-1 < \cos\theta < 1$, where 1 means perfect similarity, 0 means they are orthogonal and -1 means they are completely opposite.

3.2 Deep Learning

In this work, several strategies will be applied in order to map reports and summaries to the summary content space described in Section 1.3.1. This includes supervised deep learning architectures like FFN, LSTM and CNN. To better understand the model architectures that will be proposed in this work, a brief introduction to these methods is necessary. In this work, CNN will also be used together with an embedding layer, and thus, the embedding layer will also be introduced.

3.2.1 Feed-Forward Neural Networks

FFNs are the simplest form of artificial neural networks, where a set of input features are passed forward through layers of transformation. In its simplest form, the transformations are linear, in which case the network is equivalent to a linear transformation.

However, in such networks, so-called *activation functions* are generally applied to the output of each layer. These activation functions are generally non-linear, and by combining a sufficient amount of non-linear transformations, results have shown that feed-forward networks can learn to imitate any arbitrary function.

Let $\mathbf{x} = (x_1, \ldots, x_K)^{\mathrm{T}}$ be a set of input features. The output of a fully connected network layer can then be described mathematically by

$$\mathbf{y} = \sigma(\mathbf{z}), \quad \text{where} \quad \mathbf{z} = \mathbf{W}\mathbf{x}. \tag{3.2}$$

Here $\mathbf{z} = (z_1, \ldots, z_L)^{\mathrm{T}}$ is an L-dimensional linear transformation of $\mathbf{x}$, $\sigma(\cdot)$ is an activation function and $\mathbf{W} = (\mathbf{w}_1, \ldots, \mathbf{w}_L)^{\mathrm{T}}$ is a $L \times K$ weight matrix, where K and L are hyperparameters for the neural network. Figure 3.1 illustrates a fully connected neural network layer. FFNs are typically several fully connected layers stacked on top of each other.

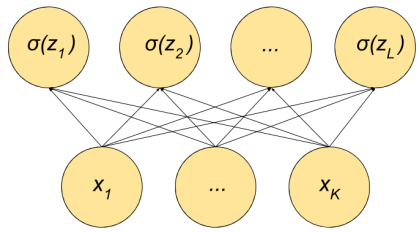

Figure 3.1: Illustration of a fully connected neural network layer. Each neuron represents a feature value, while the edges represents weights.

Learning the Weights

A neural network generally has a very high amount of trainable weights. For each layer in a feed-forward neural network, there is a weight matrix $\mathbf{W}_i$ with $L_i \cdot K_i$ weights that must be determined, where K_i and L_i are the numbers of input and output neurons, respectively, for the i-th layer.

Learning the weights is done by minimizing some loss function $l\big(h(\mathbf{x}), y\big)$, where $h(\mathbf{x})$ is the neural network, and y is the correct value that the network should output. This training objective makes neural networks a supervised architecture since they require the correct output value y to be known for each training sample $\mathbf{x}$. Without any labels y, it is virtually impossible to learn the weights of the model.

Minimization of the loss function $l\big(h(\mathbf{x}), y\big)$ is generally done by using some variation of *stochastic gradient descent*. In this algorithm, gradient descent steps are taken after looking only at a few data samples at a time. Thus, the true gradient given the entire training dataset is never calculated. Instead, gradient descent steps are taken after computing the gradient of the loss function on B training samples at a time. The training is performed in batches, where B is referred to as the batch size. In this work, we will use an optimizer algorithm called Adam. The details will not be given here, but can be found in the work of Kingma and Ba (2014).

When the network consists of many layers with hundreds of neurons, the gradient becomes rather complicated. The *error backpropagation algorithm*, which was introduced by Rumelhart, Hinton, and Williams (1986), is a systematic way of calculating this gradient and is generally used to train artificial neural networks. The details will not be given in this work, but can be found for example in the original paper or Jurafsky and Martin (2019).

3.2.2 Activation Functions

Several different activation functions can be applied in neural networks. In this work, five activation functions will be used: Linear, Rectified Linear Unit (ReLU), Sigmoid, hyperbolic tangent (tanh) and SoftMax. Linear and ReLU will be used directly in proposed model architectures, Sigmoid and tanh are used in LSTM, which will also be used and explained afterwards, while SoftMax is a classification function that is used in Word2vec and Doc2vec, which will be presented in Section 3.3. All activation functions are visualized in Figure 3.2.

Linear Activation Function

Linear activation functions in neural networks are equivalent to using no activation function, and are therefore the simplest form of activation function. The fully connected network layer becomes $\mathbf{y} = \mathbf{z} = \mathbf{W}\mathbf{x}$. Thus,

$$\sigma_{\text{linear}}(z_i) = z_i. \tag{3.3}$$

If a neural network consists of only linear layers, then the network is equivalent to a linear transformation.

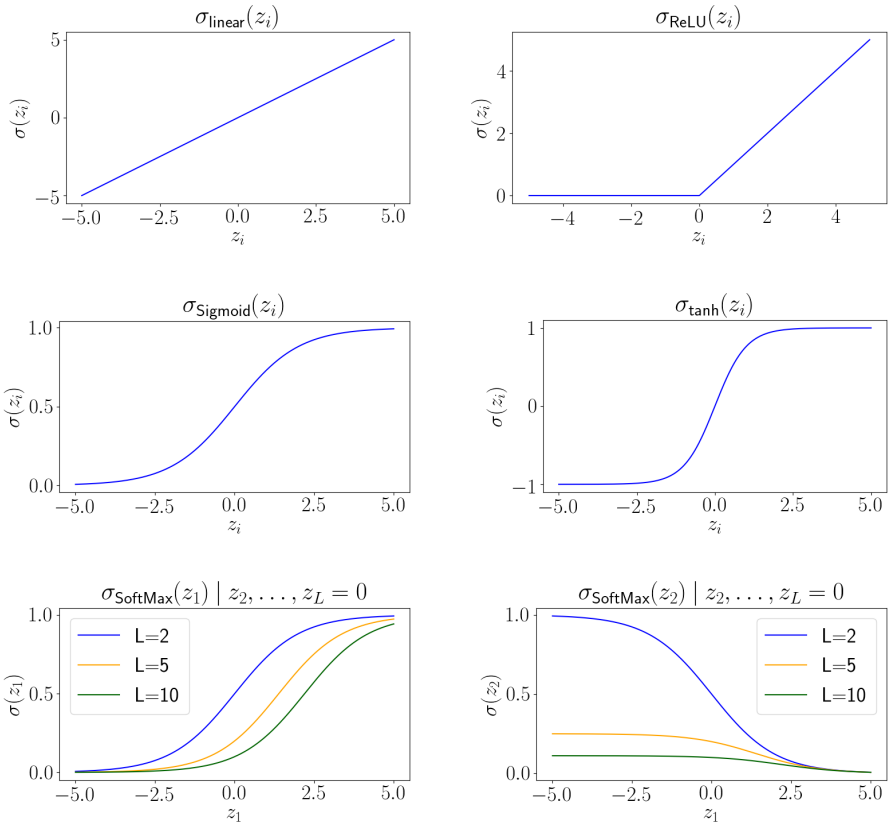

Figure 3.2: Visualizations of relevant activation functions. Note that in the visualizations of the SoftMax activation function, we investigate how $\sigma(z_1)$ and $\sigma(z_2)$ behave when z_1 vary, under the assumption that $z_2, \ldots, z_L = 0$.

ReLU Activation Function

ReLU, on the other hand, is a simple form of non-linear function. This activation function is given by

$$\sigma_{\text{ReLU}}(z_i) = \max(0, z_i). \tag{3.4}$$

This might not seem like a very powerful function, but thanks to its non-linearity, applying in millions of computations enables the neural network to imitate pretty much any arbitrary function. ReLU has several advantages compared to other activation functions, like computational simplicity and a simple gradient, and is currently the most popular activation function in deep neural networks.

Sigmoid Activation Function

The Sigmoid activation function is given by

$$\sigma_{\text{Sigmoid}}(z_i) = \frac{1}{1 + \exp(-z_i)}. \tag{3.5}$$

This is another non-linear function, which has the nice property that it maps the input to the range $[0, 1]$. It is this property that makes it suitable for LSTM networks, which will be explained in Section 3.2.4.

tanh Activation Function

In the LSTM cell, the tanh activation function, given by

$$\sigma_{\tanh}(z_i) = \tanh z_i = \frac{\exp(z_i) - \exp(-z_i)}{\exp(z_i) + \exp(-z_i)}. \tag{3.6}$$

is also used. This function maps the input to the range $[-1, 1]$, and thereby limits the output values of the network layer such that it cannot have extreme values. As we shall see, this is desirable in the LSTM cell, which is why this activation function is employed there.

SoftMax Activation Function

The last activation function that will be used in this work is the SoftMax activation function. This is actually a classification function. We will not perform classification in this work, but this activation function is used in the embedding techniques Word2vec and Doc2vec, which will be introduced in Section 3.3. The SoftMax activation function is given by

$$\sigma_{\text{SoftMax}}(z_i) = \frac{\exp(z_i)}{\sum_{l=1}^{L} \exp(z_l)}, \tag{3.7}$$

where L is the dimensionality of the vector $\mathbf{z} = (z_1, \ldots, z_L)^{\mathrm{T}}$ that SoftMax is employed on. Note that $\sum_{l=1}^{L} \sigma_{\text{SoftMax}}(z_l) = 1$. This gives the SoftMax activation function the qualities of a probability distribution, with nice interpretability. In particular, when used in a classification setting, the output of a SoftMax layer can be viewed as class probabilities.

3.2.3 The Embedding Layer

Machine learning methods generally require numerical input. Documents, which are sequences of words, must therefore be transformed to a numerical representation before they can be processed by such methods. This is why there is so much focus on embedding techniques in NLP. A common way to make word embeddings is to apply a so-called embedding layer. In this work, the embedding layer will be used in combination with CNNs. It is also an important building block in Word2vec, and thus, a short explanation is appropriate.

The embedding layer takes a sequence of words as input, and then assigns a numerical vector to each word in the input sequence. Thus, documents are mapped from a sequence of words $w_1, \ldots, w_T$ to a sequence of numerical vectors $\mathbf{x}_1, \ldots, \mathbf{x}_T$. To make this mapping, a fully connected neural network layer with linear activation functions is applied. This will be done in the following way.

Let $w^1, w^2, \ldots w^V$ denote the words in the vocabulary, that is, all unique words that occur in the dataset. Note that the superscript is used to denote that w^v is the v-th word in the vocabulary, while the subscript is used to denote that w_t is the t-th word in a sequence. Then, we let the word w^v be described by a so-called one-of-V vector $\mathbf{w}^v$, such that $\mathbf{w}^v$ is a vector of length V, where the v-th element is one, whilst all other elements are zero.

We then define the embedding layer as a fully connected neural network layer with V input neurons, and K output neurons, where K is the wanted dimensionality of the word embeddings. Thus, we get

$$\mathbf{x}_t = \mathbf{W}^{\mathrm{E}}\mathbf{w}_t, \tag{3.8}$$

where $\mathbf{W}^{\mathrm{E}}$ is the $K \times V$ weight matrix of the embedding layer, and $\mathbf{w}_t$ is the one-of-V vector of the word w_t. Note that the superscript of $\mathbf{W}^{\mathrm{E}}$ is used to denote a specific type of matrix, which in this case is an embedding matrix.

The embedding layer is generally combined with some other neural network architecture, applied to a dataset with a given loss function. Thus, the embedding matrix $\mathbf{W}^{\mathrm{E}}$ can be learned at the same time as the network is trained, by using the error backpropagation algorithm. Note also that since the words are represented as one-of-V vectors, the columns of the $K \times V$ weight matrix $\mathbf{W}^{\mathrm{E}}$ will, after training, contain K-dimensional word embeddings for all V words in the vocabulary.

3.2.4 Long Short-Term Memory

LSTM is a type of RNN. These are networks made for processing sequential data, where the output from the last element is fed into the network, together with the input for the next element. This makes RNNs able to take the previous elements into consideration when processing new elements, which makes them suitable for data like time series and texts. Since the meaning of a word often is dependent on the previous words in the sequence, RNNs can create powerful context-aware embeddings.

Many recurrent network architectures do, however, have trouble with understanding long-range dependencies. In text data, the true meaning of a word is often dependent on words that are quite far away in the sequence. Consider for example the sentence “The animal didn’t cross the street because it was too tired”. The word “it” is very dependent on “the animal”, however, there are quite a few words in-between, and in conventional RNNs, information like “the animal” is often lost when the network reaches words that are dependent on it.

LSTM is a type of RNN that was specifically designed for dealing with this long-range dependency issue. It was introduced by Hochreiter and Schmidhuber (1997) and has become a go-to architecture for embedding text data. A brief introduction to LSTM will now be given; more details can be found in Hochreiter and Schmidhuber (1997).

An LSTM layer consists of an LSTM cell which sequentially processes an input sequence $\mathbf{x}_1, \ldots, \mathbf{x}_T$, and creates an output sequence $\mathbf{h}_1, \ldots, \mathbf{h}_T$. The cell has an internal

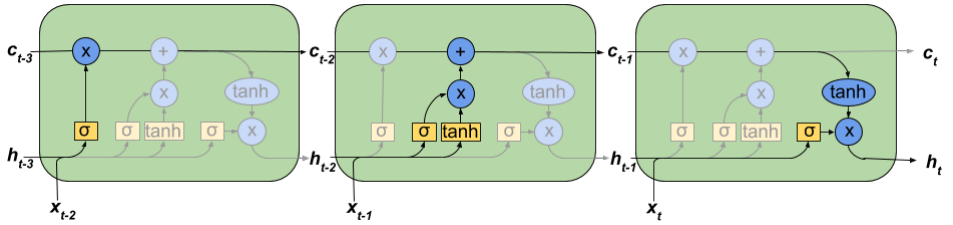

Figure 3.3: Illustration of an LSTM cell executed on three sequence steps. In the left cell, the forget gate is highlighted, in the middle cell, the update gate is highlighted, while in the right cell, the output gate is highlighted. In this figure, a yellow box represents a fully connected network layer with either Sigmoid activation function (σ) or tanh activation function (tanh). Furthermore, the blue circles represent pointwise calculations, two arrows going together means the vectors are concatenated, and one arrow splitting in two means the vector is copied.

cell state, denoted $\mathbf{c}_t$, which is a vector of information from previous elements, such that the network can "remember" information over long distances. Then, for each sequence element, the cell takes as input the vector $\mathbf{x}_t$, as well as the output from the last element, $\mathbf{h}_{t-1}$. The LSTM cell will then update its own cell state $\mathbf{c}_t$, and then combine $\mathbf{c}_t$, $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ to create the output $\mathbf{h}_t$.

The LSTM cell is illustrated in Figure 3.3. The process above is performed in three separate parts of the cell, namely the forget gate, update gate and output gate. These will now be explained.

The Forget Gate

In the forget gate, information from $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ is combined, to decide what the cell should include, and what it should forget from its last cell state $\mathbf{c}_{t-1}$. This is done by first concatenating $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$, then applying a fully connected layer with a Sigmoid activation function, and finally performing pointwise multiplication between the output of the Sigmoid layer and the last cell state $\mathbf{c}_{t-1}$.

Since the Sigmoid activation function outputs a number between 0 and 1, this results in the new cell state $\mathbf{c}_t$ including certain parts of the old cell state $\mathbf{c}_{t-1}$, whilst forgetting other parts. It is thereby the content of $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ that decides what the internal cell state $\mathbf{c}_t$ should forget from the last cell state $\mathbf{c}_{t-1}$. Thus, the LSTM network is able to learn what it should remember, and what it should forget, based on new input elements in the sequence. The forget gate is highlighted in the leftmost cell in Figure 3.3.

The Update Gate

In the update gate, the concatenated input of $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ is being used to determine what new information the cell state $\mathbf{c}_{t-1}$ should get from the input $\mathbf{x}_t$. This is done by sending the concatenated vector of $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ through two separate fully connected network layers: One with a Sigmoid activation function, and one with a tanh activation function.

The job of the tanh layer is to transform the information in $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ to an appropriate cell state representation $\mathbf{c}_t^*$. The information in the dimensions of $\mathbf{c}_t^*$ must correspond to the information in the last cell state $\mathbf{c}_{t-1}$, such that the information in $\mathbf{c}_t^*$ can be added to the internal cell state by pointwise summing the vectors. By using the tanh activation function, the output is limited between -1 and 1, which is a good property that puts a limit on the influence of single sequence elements.

Note that $\mathbf{c}_t^*$ then contains *all* of the information from $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$, in an appropriate cell state representation. The job of the Sigmoid layer is then to determine *which* parts of $\mathbf{c}_t^*$ that should be allowed through to the new, internal cell state $\mathbf{c}_t$, and which parts should be excluded. Again, since the Sigmoid activation function outputs a number between 0 and 1 in all output dimensions, this can in practice be done by pointwise multiplying $\mathbf{c}_t^*$ with the output of the Sigmoid layer. The result of this pointwise multiplication is finally added to the last cell state $\mathbf{c}_{t-1}$, such that a new cell state $\mathbf{c}_t$ is formed. The update gate is highlighted in the centre cell in Figure 3.3. Note that both the forget gate and the update gate happens in each timestep. Thus, the last cell state $\mathbf{c}_{t-1}$ is first processed in the forget gate, and then in the update gate before the new cell state $\mathbf{c}_t$ is obtained.

The Output Gate

The output gate uses the updated internal state $\mathbf{c}_t$ together with the input $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ to determine what the output $\mathbf{h}_t$ should be. In this process, the concatenated vector of $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ is again sent through a fully connected Sigmoid layer. Pointwise multiplication is then performed between the output of the Sigmoid layer, and tanh of the internal cell state $\mathbf{c}_t$.

Thus, the tanh function again makes sure that the output is not too extreme, while the sigmoid layer determines which part of the internal cell state the LSTM should output for $\mathbf{h}_t$. The output gate is highlighted to the right in Figure 3.3.

When using LSTM on document-related tasks, the last hidden state of the LSTM layer, $\mathbf{h}_T$, is normally used as the basis for solving the document-level task.

3.2.5 Convolutional Neural Networks

In the previously introduced neural network architectures, the layers have been fully connected, that is, there is a weight from all input neurons to all output neurons in the layers. While this gives the network layers a lot of flexibility, there are some issues with this connectivity.

Firstly, fully connected neural networks tend to *overfit* to the training data, that is, learn patterns from the data that, in reality, is noise. This will generally increase the performance on the training data, but make performance worse for new, unseen data samples. There are several techniques available for preventing overfitting, often referred to as *regularizations*, but the best regularization technique is often simply to reduce the complexity of the networks.

Secondly, the full connectivity between network layers results in a very high amount of trainable weights. This makes networks more expensive to train, especially with respect to memory. CNNs are networks where this full connectivity is removed, and thus, they can be seen as regularizations of fully connected feed-forward networks. This way, the

number of trainable weights is also reduced significantly. CNNs does this simplification by applying mathematical convolutions instead of matrix multiplications.

Convolutions

More specifically, a CNN layer is a collection of N_F convolutions, applied between the input $\mathbf{x}$, and a set of N_F filters $\mathbf{W}_1, \ldots, \mathbf{W}_{N_F}$. Note that the subscript of $\mathbf{W}_n$ is used to denote the n-th filter in a collection of N_F possible. Thus, the CNN layer can be described mathematically by

$$\mathbf{y}_n = \sigma(\mathbf{z}_n) \quad \text{where} \quad \mathbf{z}_n = (\mathbf{x} * \mathbf{W}_n) \quad \text{for} \quad n = 1, 2, \ldots, N_F, \tag{3.9}$$

where $\mathbf{y}_1, \ldots, \mathbf{y}_{N_F}$ are the layer outputs, $\sigma(\cdot)$ is an activation function and $(*)$ denotes the convolution operator.

In practice, the objects $\mathbf{x}$ and $\mathbf{W}_1, \ldots \mathbf{W}_{N_F}$ are tensors, that is, multidimensional arrays, which for CNNs normally are in 1D (vectors) or 2D (matrices). The filters $\mathbf{W}_1, \ldots \mathbf{W}_{N_F}$, often illustrated as windows and which typically have a much smaller size than the input $\mathbf{x}$, then "slide" over the input $\mathbf{x}$, and for each window position $\mathbf{i}$, the output value $z_{n,\mathbf{i}}$ is calculated as the sum of pointwise multiplications between the tensors $\mathbf{x}$ and $\mathbf{W}_n$, that is, the convolution. This results in N_F output tensors $\mathbf{z}_1, \ldots, \mathbf{z}_{N_F}$ with almost the same size as the input $\mathbf{x}$. An activation function is then applied to obtain the final CNN layer output.

The output will have a slightly smaller size than the input since the filter windows cannot be moved outside of the edges of the input tensor $\mathbf{x}$. This is often remedied by surrounding the input tensor with the appropriate amount of zeros such that the output becomes the same size as the input, which is referred to as *padding*.

Intuition

As the notation implies, $\mathbf{W}_1, \ldots, \mathbf{W}_{N_F}$ consist of trainable weights. Thus, instead of having weights between each input and output neurons, the same weights are reused by "sliding" them over the input neurons.

The intuition behind this type of networks is clear: The filters $\mathbf{W}_1, \ldots, \mathbf{W}_{N_F}$ are tensors with trainable weights that are meant to pick up data patterns. In the training process, the filters are trained to become experts at identifying specific structures. And since this type of network is invariant to translation, that is, the filters are moved without being rotated, it does not matter where in the input a pattern might be. If a pattern exists, and there is a filter that is able to pick up this kind of pattern, it will be reflected somewhere in the output $\mathbf{y}_1, \ldots, \mathbf{y}_{N_F}$.

This is a particularly nice quality when working with long documents: If the pattern we are searching for is anywhere in the document, it should be reflected by an extreme value somewhere in the output, no matter where it is. It can then be determined if a pattern exists in a document, for example by evaluating the maximum of the output. This will be done for N_F different filters, which should be able to identify at least N_F different patterns. This way, the network can investigate the existence of many different patterns, regardless of the length of the document.

3.3 Embedding Techniques

In this work, existing embedding techniques will be utilized in our attempt to map reports and summaries to the summary content space, as described in Section 1.3.1. In particular, the topic modelling technique LSA, and the neural network-based technique Doc2vec will be used as document embedding techniques. These are chosen for their ability to embed arbitrarily long documents. Various architectures based on the semantic feature vectors from these methods will later be proposed and implemented.

Additionally, the word embedding technique Word2vec will be utilized together with CNNs. Thus, to fully understand the models that will be implemented in this work, it is necessary with an understanding of how these embedding techniques work. The following section will therefore explain the theory and intuition behind the embedding techniques that will be used in this work.

3.3.1 Bag-of-Words

The easiest way to vectorize documents is by embedding them into the word space (described in Section 1.2.1). Then, documents are represented by a vector where each dimension corresponds to a word in the vocabulary. A value is assigned to each vector element, where different methods have different ways to assign this value. Such methods can, since they ignore the word order, be thought of as taking the words of a document and putting them in a bag. Hence, they are referred to as *bag-of-words* methods.

Although bag-of-words methods will not be used on their own in this work, they are an important building block in LSA. A brief introduction will therefore first be given. In this section, *one-hot encodings*, *frequency vectors* and TF-IDF will be discussed. The latter, TF-IDF, is by far the most used, and will also be used in this work as the basis for LSA. The first two will, however, be included for explanatory purposes.

One-Hot Encodings

The one-hot encoding is, perhaps, the simplest possible way to make vector representations of documents. The one-hot encoding does not pay attention to how many times a word occurs in a document, it only reflects whether the word is present in the document or not.

Let $w^1, w^2, \ldots, w^V$ define the vocabulary of the textual dataset (which we refer to as the corpus), and $\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_M$ be the documents in the corpus. The one-hot encoding for document $\mathbf{d}_i$ is then given by $\mathbf{z}_i = (z_i^1, z_i^2, \ldots, z_i^V)^{\mathrm{T}}$, where $z_i^v = 1$ if $w^v \in \mathbf{d}_i$, and $z_i^v = 0$ otherwise.

The entire corpus can then be represented in a matrix $\mathbf{Z}_{\text{one-hot}}$, defined by

$$\mathbf{Z}_{\text{one-hot}} = \begin{bmatrix} z_1^1 & \ldots & z_M^1 \\ \vdots & \ddots & \vdots \\ z_1^V & \ldots & z_M^V \end{bmatrix} \quad \text{where} \quad z_i^v = \begin{cases} 1 & w^v \in \mathbf{d}_i \\ 0 & w^v \notin \mathbf{d}_i. \end{cases} \tag{3.10}$$

The matrix $\mathbf{Z}_{\text{one-hot}}$ is sometimes referred to as a *term-document* matrix, since it forms a relation between the words (terms) and the documents.

Frequency Vectors

A frequency vector is a representation of a document that describes how many times each word occurs in the document. It therefore contains more information than the one-hot encoding.

Let again $w^1, w^2, \ldots, w^V$ define the vocabulary of the corpus, and $\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_M$ be the documents in the corpus. The frequency vector for document $\mathbf{d}_i$ is then given by $\mathbf{z}_i = (z_i^1, z_i^2, \ldots, z_i^V)^{\mathrm{T}}$, where z_i^v is the frequency of the word w^v in document $\mathbf{d}_i$, i.e., $z_i^v = \sum_{j=1}^{N_i} I(w_{ij} = w^v)$. Here, w_{ij} is the j-th word in the i-th document, and $I(\cdot)$ is the indicator function, which equals one if the condition inside is true, and zero otherwise.

Similarly to the one-hot encoding, the entire corpus can now be represented as a matrix $\mathbf{Z}_{\text{freq}}$, defined by

$$\mathbf{Z}_{\text{freq}} = \begin{bmatrix} z_1^1 & \cdots & z_M^1 \\ \vdots & \ddots & \vdots \\ z_1^V & \cdots & z_M^V \end{bmatrix} \quad \text{where} \quad z_i^v = \sum_{j=1}^{N_i} I(w_{ij} = w^v). \tag{3.11}$$

As $\mathbf{Z}_{\text{freq}}$ forms a relation between the words (terms) and the documents, like $\mathbf{Z}_{\text{one-hot}}$ did, this is another version of a *term-document* matrix.

TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) the most commonly used bag-of-words representation. Instead of only counting the occurrences of the words in the documents, it assigns a weight to each word for each document. This weight is a product of the *term frequency* and the *inverse document frequency*, hence the name.

The term frequency will, similarly to the frequency vectors, give higher weights to words that occur often in the documents. This weight is, however, multiplied by the inverse document frequency, which attempts to give a higher weight to rare words, thereby making common words less important. This is an important quality since many words are very common, yet do not contribute to the semantics of a text. Thus, TF-IDF is a representation that emphasizes the words that make each document unique.

The most common definition of term frequency is the raw count of the word occurrences, as given in above, under "Frequency Vectors". Hence, the term frequency for the word w^v in document $\mathbf{d}_i$ is given by

$$\text{TF}_{\mathbf{d}_i}(w^v) = \sum_{j=1}^{N_i} I(w_{ij} = w^v), \tag{3.12}$$

where w_{ij} is the j-th word in the i-th document. Thus, if a word occurs many times in a document, the term frequency will contribute to a higher weight.

The most common definition of *inverse document frequency* for a word w^v is given by the logarithm of the total number of documents divided by the number of documents where w^v appears, i.e.,

$$\text{IDF}(w^v) = \log \frac{M}{m^v} \quad \text{where} \quad m^v = \sum_{i=1}^{M} I(w^v \in \mathbf{d}_i). \tag{3.13}$$

Here, M is the total number of documents in the corpus, while m^v is the number of documents where the word w^v appears. Hence, rare words will get higher IDF values.

The TF-IDF representation for document $\mathbf{d}_i$ is then given by $\mathbf{z}_i = (z_i^1, z_i^2, \ldots, z_i^V)^{\mathrm{T}}$ where $z_i^v = \mathrm{TF}_{\mathbf{d}_i}(w^v) \cdot \mathrm{IDF}(w^v)$. The entire corpus can now be represented in a matrix $\mathbf{Z}_{\text{TF-IDF}}$, given by

$$\mathbf{Z}_{\text{TF-IDF}} = \begin{bmatrix} z_1^1 & \cdots & z_M^1 \\ \vdots & \ddots & \vdots \\ z_1^V & \cdots & z_M^V \end{bmatrix} \quad \text{where} \quad z_i^v = \mathrm{TF}_{\mathbf{d}_i}(w^v) \cdot \mathrm{IDF}(w^v), \tag{3.14}$$

where $\mathrm{TF}_{\mathbf{d}_i}(w^v)$ and $\mathrm{IDF}(w^v)$ are given in (3.12) and (3.13) respectively. $\mathbf{Z}_{\text{TF-IDF}}$ also forms a relation between words (terms) and documents, and is therefore a third example of a *term-document* matrix.

3.3.2 Latent Semantic Analysis

LSA is an unsupervised method that involves discovering latent (hidden) topics across a collection of documents. The topic vectors of documents can also be seen as semantic document embeddings, and this method will be used as an embedding technique in this work. LSA was first introduced by Deerwester et al. (1990). The starting point for the analysis is a bag-of-words representation. As mentioned, bag-of-words representations do not pay attention to the context of words, and thus LSA assumes that the semantics of a document is defined only by which words are present.

The bag-of-words representation contains a lot of semantic information. However, as the vocabulary size V and the number of documents M grow, the information in a term-document matrix $\mathbf{Z}$ will become huge and sparse. LSA handles this by applying dimensionality reduction using truncated singular value decomposition. The result is a low-rank approximation to the bag-of-words representation which, as it turns out, also creates a relation from the words and documents to a set of latent topics.

Information Matrices

As mentioned before, the bag-of-words vectors forms so-called *term-document* matrices, since they form a relation between the terms (words) and the documents. Now, let $\mathbf{A}$ denote a term-document matrix, and let the term-document matrix be given by the one-hot encoding $\mathbf{A} = \mathbf{Z}_{\text{one-hot}}$, as presented in (3.10) in Section 3.3.1. Now, $\mathbf{A}$ is a $V \times M$ matrix such that if the word w^v appears in the document $\mathbf{d}_i$, then $\mathbf{A}_{v,i} = 1$, otherwise $\mathbf{A}_{v,i} = 0$.

Further, let $\mathbf{B}$ be given by $\mathbf{B} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$, which is an $M \times M$ matrix. Now, if the documents $\mathbf{d}_i$ and $\mathbf{d}_j$ have b words in common, then $\mathbf{B}_{i,j} = b$. Since this matrix forms a relation between documents, this is referred to as the *document-document* matrix.

Finally, let $\mathbf{C}$ be given by $\mathbf{C} = \mathbf{A}\mathbf{A}^{\mathrm{T}}$, which is a $V \times V$ matrix. Then, if word w^v and w^u appear together in c documents, then $\mathbf{C}_{v,u} = c$. This matrix forms a relation between words, or terms, and is therefore referred to as the *term-term* matrix.

The matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ form the foundation for the singular value decomposition. It will further be shown that by applying singular value decomposition to $\mathbf{A}$, a *document-topic* matrix and a *topic-term* matrix will be formed.

In the example above, $\mathbf{A}$ is given by the one-hot encoding. Although this is fine, most applications of LSA is based on the TF-IDF representation defined in (3.14) in Section 3.3.1, i.e., $\mathbf{A} = \mathbf{Z}_{\text{TF-IDF}}$. In this work, whenever we use LSA, we use $\mathbf{A} = \mathbf{Z}_{\text{TF-IDF}}$.

Results from Linear Algebra

The *document-document* matrix $\mathbf{B}$ and the *term-term* matrix $\mathbf{C}$ have a few properties that are important for the singular value decomposition:

First, the spectral theorem for symmetric matrices states that the eigenvalues of a symmetric matrix are real, and that the eigenvectors form an orthonormal basis. It is easy to see that both $\mathbf{B}$ and $\mathbf{C}$ are symmetric, since

$$\mathbf{B} = \mathbf{A}^{\mathrm{T}}\mathbf{A} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}} \quad \text{and} \quad \mathbf{C} = \mathbf{A}\mathbf{A}^{\mathrm{T}} = \left(\mathbf{A}\mathbf{A}^{\mathrm{T}}\right)^{\mathrm{T}} = \mathbf{C}^{\mathrm{T}}. \tag{3.15}$$

Furthermore, let $\mathbf{M}$ be a square $n \times n$ matrix with orthonormal columns. Then $\mathbf{M}^{\mathrm{T}}\mathbf{M} = \mathbf{I}_{n\times n}$. It follows that

$$\mathbf{M}^{\mathrm{T}}\mathbf{M} = \mathbf{I}_{n\times n} = \mathbf{M}^{-1}\mathbf{M} \implies \mathbf{M}^{\mathrm{T}} = \mathbf{M}^{-1} \implies \mathbf{M}\mathbf{M}^{\mathrm{T}} = \mathbf{M}\mathbf{M}^{-1} = \mathbf{I}_{n\times n}. \tag{3.16}$$

Finally, let $\mathbf{M}$ be any matrix. Now $\mathbf{M}$ is positive semi-definite if $\mathbf{v}^{\mathrm{T}}\mathbf{M}\mathbf{v} \geq 0$ for all vectors $\mathbf{v} \neq 0$. It can then be seen that $\mathbf{B}$ is positive semi-definite, since

$$\mathbf{v}^{\mathrm{T}}\mathbf{B}\mathbf{v} = \mathbf{v}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{v} = (\mathbf{A}\mathbf{v})^{\mathrm{T}}(\mathbf{A}\mathbf{v}) \geq 0. \tag{3.17}$$

These results will be used to see that the *document-document* matrix $\mathbf{B}$ and the *term-term* matrix $\mathbf{C}$ play an important role in the singular value decomposition.

Singular Value Decomposition

Singular value decomposition is a matrix factorization which decomposes a matrix $\mathbf{M}$ into three matrices, $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$. In LSA, singular value decomposition is used on the *term-document* matrix $\mathbf{A}$. The following shows how the document-document matrix $\mathbf{B}$ and the term-term matrix $\mathbf{C}$ form the foundation of the singular value decomposition:

Let the eigenvalues of $\mathbf{B}$ be given by $\sigma_1^2, \sigma_2^2, \ldots, \sigma_M^2$ such that $\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_M^2$. Let further $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M$ be the eigenvectors of $\mathbf{B}$ corresponding to the eigenvalues $\sigma_1^2, \sigma_2^2, \ldots, \sigma_M^2$, such that $\mathbf{B}\mathbf{x}_i = \sigma_i^2\mathbf{x}_i$. Since $\mathbf{B}$ is symmetric, as shown in (3.15), it follows from the spectral theorem for symmetric matrices that $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M$ are orthonormal.

Let us now define

$$(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_M) = \left(\frac{1}{\sigma_1}\mathbf{A}\mathbf{x}_1, \frac{1}{\sigma_2}\mathbf{A}\mathbf{x}_2, \ldots, \frac{1}{\sigma_M}\mathbf{A}\mathbf{x}_M\right). \tag{3.18}$$

These vectors have a few nice properties:

1. The vectors $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_M$ are orthonormal, since

$$\mathbf{y}_i^{\mathrm{T}}\mathbf{y}_j = \left(\frac{1}{\sigma_i}\mathbf{A}\mathbf{x}_i\right)^{\mathrm{T}}\frac{1}{\sigma_j}\mathbf{A}\mathbf{x}_j = \frac{1}{\sigma_i\sigma_j}\mathbf{x}_i^{\mathrm{T}}\mathbf{B}\mathbf{x}_j = \frac{\sigma_j}{\sigma_i}\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases}$$

2. The vectors $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_M$ are the M first eigenvectors of $\mathbf{C}$, with eigenvalues $\sigma_1^2, \sigma_2^2, \ldots, \sigma_M^2$, since

$$\mathbf{C}\mathbf{y}_i = \frac{1}{\sigma_i}\mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x}_i = \frac{1}{\sigma_i}\mathbf{A}\mathbf{B}\mathbf{x}_i = \frac{\sigma_i^2}{\sigma_i}\mathbf{A}\mathbf{x}_i = \sigma_i^2\mathbf{y}_i \tag{3.19}$$

3. The following relation holds:

$$\mathbf{y}_i^{\mathrm{T}}\mathbf{A}\mathbf{x}_j = \sigma_j\mathbf{y}_i^{\mathrm{T}}\mathbf{y}_j = \begin{cases}\sigma_j & i = j \\ 0 & i \neq j.\end{cases} \tag{3.20}$$

Now, $\mathbf{C}$ is a $V \times V$-matrix, and therefore has V eigenvectors $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_V$. Since $\mathbf{C}$ is symmetric, as shown in (3.15), it follows again from the spectral theorem for symmetric matrices that $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_V$ are orthonormal.

Let $\mathbf{U} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_V]$ and $\mathbf{V} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M]$. It is now clear from (3.20) that the following relation holds:

$$\mathbf{U}^{\mathrm{T}}\mathbf{A}\mathbf{V} = \mathbf{\Sigma}, \tag{3.21}$$

where $\mathbf{\Sigma}$ is a $V \times M$ matrix with $\mathbf{\Sigma}_{i,j} = \sigma_i$ for $i = j$, and $\mathbf{\Sigma}_{i,j} = 0$ for $i \neq j$. Note further that since $\mathbf{U}$ and $\mathbf{V}$ are square matrices with orthonormal columns, it follows from (3.16) that $\mathbf{U}\mathbf{U}^{\mathrm{T}} = \mathbf{I}_{V\times V}$ and $\mathbf{V}\mathbf{V}^{\mathrm{T}} = \mathbf{I}_{M\times M}$. Thus, (3.21) can be written as

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}. \tag{3.22}$$

This shows that the singular value decomposition of the *term-document* matrix $\mathbf{A}$ is obtained from the eigenvectors of the *document-document* matrix $\mathbf{B}$ and the *term-term* matrix $\mathbf{C}$. The last matrix $\mathbf{\Sigma}$ contains the square root of the eigenvalues of $\mathbf{B}$, which are referred to as the singular values. It follows from (3.17) that $\mathbf{B}$ is positive semi-definite, thus the eigenvalues are non-negative, i.e., $\sigma_1^2, \sigma_2^2, \ldots, \sigma_M^2 \geq 0$. The singular values are therefore also real.

Interpretation of the Singular Value Decomposition

An interesting question that now arises is how this new representation can be interpreted. To answer this, some conceptual understanding of eigenvectors and eigenvalues is necessary.

A matrix $\mathbf{M}$ is, in reality, a linear transformation that explains how a space is stretched in a number of directions. The eigenvectors represent the directions in which the space is stretched, while the eigenvalues represent how much it is stretched in the directions of the corresponding eigenvectors.

However, the *document-document* matrix $\mathbf{B}$ and the *term-term* matrix $\mathbf{C}$ are not really linear transformations, they are rather matrices of information. So the eigenvectors represent the "directions of information", while the eigenvalues represent how much information there is in the corresponding directions. So what are "directions of information"? To answer this, the eigenvectors and eigenvalues of $\mathbf{B}$ and $\mathbf{C}$ are investigated further.

An eigenvector of $\mathbf{B}$, $\mathbf{x}_i$, is a vector of length M, one value for each document. It can be seen as a representation of a topic, where the topic is described by how much it is

related to each of the documents. The corresponding eigenvalue σ_i^2 then describes how much this topic is present in the collection of documents.

Similarly, an eigenvector of $\mathbf{C}$, $\mathbf{y}_i$, is a vector of length V, one value for each word. It can also be seen as a representation of a topic, where the topic is described by how much it is related to each of the words. For the M first eigenvectors of $\mathbf{C}$, $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_M$, the eigenvalues are the same as for $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M$, as seen in (3.19).

Thus, the interpretation of the singular value decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$ is the following: The decomposition reveals a set of M latent topics. The singular values $\sigma_1, \sigma_2, \ldots, \sigma_M$ describe how much the topics are present in the documents and words. The topics themselves are described both by how much they are related to the documents, in the columns of $\mathbf{V}$, and how much they are related to the words, in the columns of $\mathbf{U}$. Essentially, $\mathbf{U}$ is a *term-topic* matrix that forms a relation from the terms to the topic, while $\mathbf{V}$ is a *topic-document* matrix that forms a relation from the topics to the documents.

Dimensionality Reduction

Since $\mathbf{U}$ is a $V \times V$ matrix, $\mathbf{\Sigma}$ is a $V \times M$ matrix, and $\mathbf{V}$ is an $M \times M$ matrix, then $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$ is still a high-dimensional representation. Dimensionality reduction can now be obtained by using truncated singular value decomposition. This involves approximating $\mathbf{A}$ by using only sub-matrices of $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{\Sigma}$.

The *term-topic* matrix $\mathbf{U}$ is a representation of the words in the topic space, while the *topic-document* matrix $\mathbf{V}$ is a representation of the documents in the topic space. As discussed above, the singular values $\sigma_1, \sigma_2, \ldots, \sigma_M$ describe how much the topics are present in the words and documents. One can obtain a lower-dimensional approximation of $\mathbf{A}$ by only considering the most significant topics.

The singular values are ordered such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_M$. Define $\mathbf{\Sigma}_K$ as the upper left square sub-matrix of $\mathbf{\Sigma}$, that is, a $K \times K$ diagonal matrix with σ_i on the diagonal for $i = 1, 2, \ldots, K$. The matrix $\mathbf{\Sigma}_K$ now contains the K most significant singular values (topics) present in the documents and words. By further defining $\mathbf{U}_K = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_K]$ and $\mathbf{V}_K = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K]$, $\mathbf{U}_K$ now becomes a representation of the words in the K dimensional reduced topic space, while $\mathbf{V}_K$ becomes a representation of the documents in the same space.

A low-rank approximation of $\mathbf{A}$ now becomes $\mathbf{A} \approx \mathbf{U}_K\mathbf{\Sigma}_K\mathbf{V}_K^{\mathrm{T}}$. Note that since $\mathbf{U}_K$ is a $V \times K$ matrix, $\mathbf{\Sigma}_K$ is a $K \times K$ matrix, and $\mathbf{V}_K^{\mathrm{T}}$ is a $K \times M$ matrix, this is still a $V \times M$ approximation. This is a good approximation in the context of topic modelling since it is based on a representation of the words and documents in the most significant K dimensional topic space.

Application

Based on the low-rank approximation $\mathbf{A} \approx \mathbf{U}_K\mathbf{\Sigma}_K\mathbf{V}_K^{\mathrm{T}}$, there are several nice applications. The low-rank *topic-document* matrix $\mathbf{V}_K$ is now a representation of the M documents in the K dimensional reduced topic space. In the context of document similarity, the rows of $\mathbf{V}_K\mathbf{\Sigma}_K$ can be seen as feature vectors for the documents, where the features represent how much a document is related to the different topics. Note that $\mathbf{V}_K$ is multiplied by $\mathbf{\Sigma}_K$, so that more important topics get a higher weight in the resulting feature

vectors.

Let now $\mathbf{b}$ be the bag-of-words representation of a new, unseen document $\mathbf{d}$. It is then possible to obtain a K-dimensional feature vector $\mathbf{z}$ for the document by calculating $\mathbf{z} = \mathbf{U}_K^{-1}\mathbf{b}$, since this creates a representation of the new document in the K-dimensional topic space, corresponding to $\mathbf{V}_K\mathbf{\Sigma}_K$. In this work, $\mathbf{z} = \mathbf{U}_K^{-1}\mathbf{b}$ will be used as the LSA document embedding for input documents $\mathbf{d}$, with corresponding bag-of-words representation $\mathbf{b}$. Note that it is possible to train LSA on a corpus, and then find document embeddings for new, unseen documents.

The low-rank *term-topic* matrix $\mathbf{U}_K$ is further a representation of the V words in the K dimensional reduced topic space. Since semantically similar words should be related to the same topics, similar words should have a similar representation in the topic space. Thus, by comparing words in the low-dimensional space, one could find semantic relations between words, like synonyms. Also, by investigating which words that are most relevant to the different topics, one could extract a lot of information about the various latent topics.

3.3.3 Word2vec

Word2vec is a strong baseline word-embedding technique that will be used in this work together with CNNs. The model was originally presented by (Mikolov et al. 2013b). Here, they presented two different versions, namely Continuous Bag-of-Words and Continuous Skip-Gram. They are both based on a similar architecture, but the input and training objectives are different. In this work, the Skip-Gram architecture will be employed, and therefore presented in this work.

The model architecture in question is a fully connected feed-forward neural network, the theory of which was presented in Section 3.2.1. Since it is based on a neural network, this model requires labelled data to train on. However, Word2vec is a so-called *self-supervised* method, that is, it creates its own labels and training objective from unlabelled data. This is a good feature for the problem at hand since word embeddings can be learned independently of the noisy weak supervision labels.

The goal of Word2vec is to create a K-dimensional semantic feature vector $\mathbf{z}^v$ for all the words in the vocabulary $w^1, \ldots, w^V$. Note again that the superscript of w^v is used to denote the v-th word in the vocabulary, while the subscript of w_t will be used to denote the t-th word in a sequence $w_1, \ldots, w_T$. The architecture that the Word2vec Skip-Gram model employs to achieve this will now be further elaborated.

Skip-Gram Neural Network Architecture

In Word2vec, K-dimensional feature vectors $\mathbf{z}^1, \ldots, \mathbf{z}^V$ will be obtained by employing an embedding layer, which was described in Section 3.2.3. In the Skip-Gram architecture, the input to the network is a single word w_t in a sequence $w_1, \ldots, w_T$, and thus, the Word2vec embedding layer has V input neurons and K output neurons, and a $K \times V$ trainable embedding matrix $\mathbf{W}^{\mathrm{E}}$. The idea is then that after the training process, the embedding matrix $\mathbf{W}^{\mathrm{E}}$ will constitute the word embeddings, that is, $\mathbf{W}^{\mathrm{E}} = (\mathbf{z}^1, \ldots, \mathbf{z}^V)$.

The embedding matrix $\mathbf{W}^{\mathrm{E}}$ must, however, be trained. To do this, a dummy classification layer will be added on top of the embedding layer. This classification layer is not really of interest and is only included so that the embedding matrix $\mathbf{W}^{\mathrm{E}}$ can be trained.

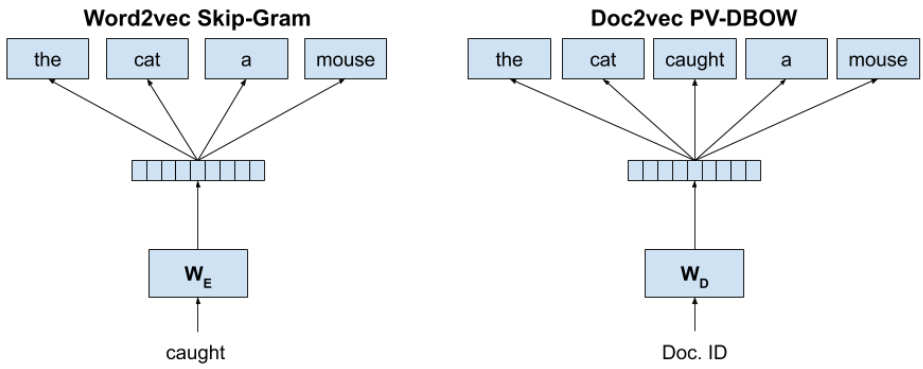

Figure 3.4: Illustration of the architecture for Word2vec Skip-Gram (left), and Doc2vec PV-DBOW (right). In this example, the context window size is given by $R = 2$.

In the Skip-Gram model, this dummy classification task is to correctly predict the $2R$ surrounding words given the input, i.e., $(w_{t-R}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+R})$. Thus, the output classification layer has K input neurons, and since there are V potential words to predict in the output, there are V output neurons. Finally, the SoftMax activation function, as defined in Section 3.2.2, is employed in the output classification layer.

In the Skip-Gram model, the value R is, in each training step, sampled as a random number from $\{1, \ldots, C\}$, where C is a hyperparameter for the model. By assigning a higher probability to lower numbers, the model gives higher importance to the words that are closer to the input word. The Skip-Gram architecture is illustrated to the left in Figure 3.4.

Training

As previously mentioned, there are $V \times K$ weights between the input and the hidden layer. However, for each training iteration, only a small part of this matrix is active at a time. In particular, for the Skip-Gram model, only K weights are active at a time for each of the $2R$ output predictions. The training complexity in this part of the network is therefore quite cheap.

For the output layer, however, the training complexity is $C \times K \times V$, and since the vocabulary V is large, this is quite expensive. To remedy this, a technique called *negative sampling* was introduced by Mikolov et al. (2013a). Negative sampling is a technique where only a few of the weights in the output layer are updated at a time. This is done by randomly selecting a small number of words, and only updating the weights corresponding to these words, in addition to the input word, for each training iteration. They find that negative sampling outperforms other training schemes. Therefore, Continuous Skip-Gram with negative sampling is the model used when referring to Word2vec in the rest of this work.

Intuition

The idea behind Word2vec is based on *the distributional hypothesis* (Harris 1954). This is the idea that similar words appear in similar contexts. That is, if two words are semantically similar, they should tend to be surrounded by the same words across the documents.

The Continuous Skip-Gram model is then particularly intuitive. Under this hypothesis, if the words w^v and w^u are semantically similar, they should have similar output probabilities in the neural network of Word2vec. And in the Skip-Gram model, the output probabilities are predicted based on the K-dimensional feature vectors $\mathbf{z}^v$ and $\mathbf{z}^u$. So, in order to have similar output probabilities, $\mathbf{z}^v$ and $\mathbf{z}^u$ must also be similar. Thus, the Word2vec Skip-Gram model results in word embeddings $\mathbf{z}^1, \ldots, \mathbf{z}^V$, such that semantically similar words have similar embeddings.

3.3.4 Doc2vec

Doc2vec, or Paragraph Vector which was the original name first presented by Le and Mikolov (2014), is a document embedding technique which generates semantically meaningful feature vectors for documents of arbitrary length. It will therefore be used in this work as a baseline model, as well as together with FFN and LSTM architectures, in order to map reports and summaries to the summary content space, as described in Section 1.3.1.

This model arrived soon after Word2vec as a natural extension from words to documents. It is based on a fully connected feed-forward neural network which is similar to that of Word2vec, but which is made for embedding documents instead of words. The authors presented two variants, namely Distributed Memory (PV-DM) and Distributed Bag-of-Words (PV-DBOW). These are similar to Continuous Bag-of-Words and Continuous Skip-Gram, respectively. In this work, the PV-DBOW version will be employed, and therefore explained further.

The goal of Doc2vec is to obtain a semantic feature vector $\mathbf{z}_i$ for each document in a corpus $\mathbf{d}_1, \ldots, \mathbf{d}_M$. A document is in general represented as a sequence of words, but in the PV-DBOW architecture, documents are instead represented by one-of-M vectors, similar to words in the embedding layer. That is, the document $\mathbf{d}_i$ will be represented as a vector of length M where the i-th element is 1, while the rest is zero. This document representation will be useful when extending the Word2vec Skip-Gram model to documents.

PV-DBOW Neural Network Architecture

The Doc2vec version PV-DBOW is, despite its name, more similar to the Word2vec Continuous Skip-Gram model than to the Continuous Bag-of-Words method. In the Word2vec Skip-Gram model, surrounding context words are predicted based on an input word. In PV-DBOW, words in a context window are instead predicted based on an input document. Hence, the word embedding matrix $\mathbf{W}^{\mathrm{E}}$ in the input layer is replaced with a document embedding matrix $\mathbf{W}_{\mathrm{D}}$.

Since the documents $\mathbf{d}_1, \ldots, \mathbf{d}_M$ now are represented as one-of-M vectors, the input layer in PV-DBOW has M input neurons and K output neurons, and a $K \times M$ trainable weight matrix $\mathbf{W}^{\mathrm{D}}$. The idea is that the weight matrix $\mathbf{W}^{\mathrm{D}}$ after training

will contain K-dimensional feature vectors for the documents in the input corpus, i.e., $\mathbf{W}_{\mathrm{D}} = (\mathbf{z}_1, \ldots, \mathbf{z}_M)$.

Again, as with Word2vec, the input layer must be trained, and for that, a dummy classification task will be employed. In PV-DBOW, this will be done the following way: For each training iteration, a word window $(w_{t-R}, \ldots, w_{t+R})$ will be randomly sampled from the input document $\mathbf{d}_i$. The model objective is then to predict the words in the sampled word window based only on the embedded document $\mathbf{z}_i$. Thus, the output layer of Doc2vec has K input neurons and V output neurons, and the SoftMax activation function is again employed, as in Word2vec. This model architecture is illustrated to the right in Figure 3.4. A similar training scheme of that to Word2vec is employed to train the model, and obtain feature vectors $\mathbf{z}_1, \ldots, \mathbf{z}_M$.

Intuition

In the PV-DBOW architecture, word windows are predicted based on the embedded input documents. This means that two documents $\mathbf{d}_i$ and $\mathbf{d}_j$ will be modelled as similar if they contain many of the same words, and in particular, if they contain similar windows of words. Thus, the embeddings are slightly more context-aware than LSA, which only takes into account which words are present in documents. However, in the Doc2vec scheme, the frequent words of a document will in general be assigned a higher predicted probability than less frequent words. Thus, the model will mainly reflect which words are present, which is why the name "Distributed Bag-of-Words" was given to this version.

Finally, once the Doc2vec model is trained, the columns in $\mathbf{W}_{\mathrm{D}}$ now contains K-dimensional feature vectors for all the M documents in the training corpus. These are trained such that the resulting document embeddings $\mathbf{z}_1, \ldots, \mathbf{z}_M$ should be semantically meaningful. Furthermore, the document embedding $\mathbf{z}$ for a new, unseen document $\mathbf{d}$ can be obtained by keeping the weights in the output layer fixed, and then train a new embedding vector $\mathbf{z}$ on the new document, by randomly sampling windows of words from $\mathbf{d}$, and then find the embedding vector $\mathbf{z}$ that maximizes the prediction quality of the Doc2vec output layer.

3.4 Supervision

Traditionally, in the field of machine learning, there has been a distinction between *supervised* and *unsupervised* learning methods. This comes naturally from the distinction between *labelled* data, where each data sample is accompanied by a label that defines some characteristic of the sample, and *unlabelled* data, where no such additional information about the data samples is given.

In the real estate condition report data, each sample consists of a real estate condition report and its summary. This data is unlabelled since it does not include any labels that are relevant for the task of measuring summary quality. Due to the length of the real estate condition reports, it would require a large amount of work, from people with professional domain knowledge, to create labels for this dataset. This is not an option for this work.

This fact would, traditionally, restrict us to using only unsupervised machine learning methods. However, in NLP, a large amount of *self-supervised* methods have been devel-

oped. In addition, machine learning has seen a new supervision paradigm in the last few years, namely *weak supervision*. These concepts reduce the gap between unsupervised and supervised learning, and makes it possible to use more complex methods to solve the summary quality problem, despite the restrictions that the unlabelled data impose.

In this section, a brief introduction to various supervision alternatives for the problem at hand will be discussed. Then, the theory and intuition behind weak supervision will be thoroughly presented, since this is the supervision alternative we choose to pursue in this work.

3.4.1 Unsupervised Learning

Unsupervised learning methods are algorithms that can discover patterns and relations from unlabelled data. Typical examples are clustering algorithms, which involve grouping of data samples, and principal component analysis, which involves changing the basis of the data. The latter can be used both to perform dimension reductions and to analyse the importance of different features in data samples.

Within the field of document similarity, both bag-of-words methods and LSA, as presented in Section 3.3.1 and 3.3.2, respectively, are unsupervised methods, since they explore unlabelled data. In particular, LSA can reveal a lot of information about unlabelled data, since this model discovers latent topics in the input documents. The meaning of these topics can furthermore be analysed by investigating the relationship between topics and words, and the importance of different topics are described by the singular values in the singular value decomposition. Finally, both methods create embeddings, which can be used to perform clustering analyses, as well as calculating semantic document similarity.

These kinds of algorithms can be very useful, but we never quite know what kind of insight we might obtain. They are therefore very suitable for exploring unlabelled data, but it might be difficult to know how to utilize this new, discovered insight when solving specific problems. It is therefore often difficult to apply such methods when we are solving particular problems.

The summary quality problem is a good example of this. With LSA, we can obtain document embeddings, and with an appropriate distance measure, we can measure the semantic similarity between reports and summaries. However, we can not expect the full semantic similarity to be a sufficient measure of summary quality, since some parts of the report will be more important to include in a summary than others. And even if we have gained a lot of insight when creating document embeddings, the topics are still latent, and thereby difficult to understand. Applying this insight to solve the summary quality problem in an unsupervised manner is thus not straightforward.

3.4.2 Self-Supervised Learning

Self-supervised methods are methods that actually are based on supervised architectures, but which create their own labels and training objectives so that they can be applied to unlabelled data. Thus, they are in many ways similar to unsupervised methods.

Both Word2vec and Doc2vec, presented in Section 3.3.3 and 3.3.4 are self-supervised methods. Their self-made training scheme makes them suitable for creating word and document embeddings that are semantically meaningful.

These methods do, however, also share some of the difficulties of the unsupervised methods, when it comes to solving specific problems. For example, the document embeddings for Doc2vec are made by maximizing performance only, and thus, it is virtually impossible to know what the resulting feature vectors really mean. Thus, we can obtain document embeddings, and measure semantic similarity, but we are no closer to actually solving the summary quality problem, which involves mapping the document embeddings from the semantic space to the summary content space.

3.4.3 Supervised Learning

Supervised learning methods are made for solving specific problems. These methods involve a training procedure where the goal is to maximize the performance on some specific training objective. This training objective makes the methods experts at solving the specific task they are made for, whilst being unsuitable for anything else. Thus, some insight about the problem we are trying to solve is necessary, typically in form of data samples with reliable labels.

FFNs, LSTM and CNNs, presented in Section 3.2, are examples of this. These models will train their weights by minimizing some loss function. By doing this, the models become experts at minimizing this very loss, but this is the only task they will be able to solve.

Ideally, we would like to use supervised learning methods in order to map reports and summaries to the summary content space, since supervised methods can become very good at such specific tasks. However, to use such methods, information about the true summary quality is necessary, in the form of labels. Traditionally, this can only be obtained by spending endless hours on hand-labelling a training set for the models. However, as this is not an option in this work, we will instead attempt to get such labels by using weak supervision.

3.4.4 Weak Supervision

Weak supervision is a branch of machine learning that attempts to avoid the exhausting job of hand-labelling data. This is done by making labels from weaker forms of supervision instead. By using weak supervision, noisy and imprecise labels can be made on large amounts of data, with a much smaller effort. And as various results have shown (Ratner et al. 2017), weak supervision can be sufficient to train complex machine learning models.

In this work, the weak supervision system *Snorkel* will be used. Snorkel was first introduced by Ratner et al. (2016), and was later expanded by Bach et al. (2017). Then, in 2019, Snorkel went through a big update, in which the main model behind the system was changed, as described by Ratner et al. (2019).

In the following subsections, the theory and intuition behind the current Snorkel system will be explained.

Labelling Functions

Snorkel is a weak supervision system that allows users to create labels by making labelling rules, instead of providing the labels directly. These rules are formulated as so-called *la-*

belling functions, which capture some characteristic of the data samples, and then classify them based on this.

Labelling functions can be based on rules or patterns, they can be weak classifiers; that is, classifiers that have noise, bias or insufficient coverage and they can be based on distant supervision; that is, making classifiers based on external knowledge bases.

Consider a two-way classification problem, where $x_1, x_2, \ldots, x_M$ are the samples in the dataset, and $y_1, y_2, \ldots, y_M$ are the true labels, such that $y_m \in \{-1, 1\}$ for $m = 1, 2, \ldots, M$. Note that in a weak supervision setting, the true labels $y_1, \ldots, y_M$ are latent, that is, hidden.

A labelling function, denoted $\lambda(x)$, is a function that predicts the class of a sample, i.e., $\lambda(x) \in \{-1, 0, 1\}$. Note that a labelling function might abstain from predicting the class for a given sample, which is denoted by a '0'.

In a weak supervision system, there will generally be many labelling functions. Let $\boldsymbol{\lambda}(x) = \big(\lambda_1(x), \ldots, \lambda_L(x)\big)$ denote all labelling functions. Applying the labelling functions $\boldsymbol{\lambda}(x)$ on all samples $x_1, \ldots, x_M$ will result in a $M \times L$ label matrix $\boldsymbol{\Lambda} \in \{-1, 0, 1\}^{M \times L}$.

The label matrix $\boldsymbol{\Lambda} = (\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_M)^{\mathrm{T}}$ forms a basis for creating labels. However, the various labelling functions might disagree on some samples, and abstain from labelling on others. Therefore, we must decide how to combine the labels from the various labelling functions $\boldsymbol{\lambda}_m$ to create a single label y_m. A naive approach would be to use a majority-vote system, which outputs the label that the majority of labelling functions output. This is, however, not the best approach, since some labelling functions might be more accurate than others, and the labelling functions might be correlated. Ideally, we would like to model both the accuracy and correlation of the labelling functions.

Snorkel does this by defining a *label model* $P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda})$, which takes the predicted labels $\boldsymbol{\lambda}$ from the labelling functions as input, and then outputs a conditional probability for y. The label model $P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda})$ is constructed to consider both the accuracy of the labelling functions, and the correlations between them. It is parameterized by a vector of probabilities $\boldsymbol{\mu}$. Before we can specify, we need to introduce some background theory. In particular, the label model will be given by

$$P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda}) = \frac{P_{\boldsymbol{\mu}}(y, \boldsymbol{\lambda})}{\sum_{y \in \{-1,1\}} P_{\boldsymbol{\mu}}(y, \boldsymbol{\lambda})}, \tag{3.23}$$

where $P_{\boldsymbol{\mu}}(y, \boldsymbol{\lambda})$ is formulated as a *probabilistic graphical model*, of a specific type called a *junction tree*. These are core concepts, and will therefore be explained.

Graphical Models

A graph is a mathematical structure that models the pairwise relations between a set of objects. A graph is formulated as a set of vertices $\mathcal{V} = \{V_1, \ldots, V_{N_\mathcal{V}}\}$, which represents the objects in the graph, and a set of edges $\mathcal{E} \subseteq \{(V_i, V_j) : V_i, V_j \in \mathcal{V} \text{ and } V_i \neq V_j\}$, where an edge (V_i, V_j) represents a relation between the objects V_i and V_j. Edges can be directed or undirected, and in this work we deal with undirected graphs.

In Snorkel, $P_{\boldsymbol{\mu}}(y, \boldsymbol{\lambda})$ will be represented by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where the vertices are given by $\mathcal{V} = (V_0, V_1, \ldots, V_L) = (y, \lambda_1, \ldots, \lambda_L)$. In words, all the labelling functions, as well as the latent, true label, are vertices in the graph. Thus, the graph has $L + 1$ vertices.

Furthermore, there will be an edge (V_0, V_l) for $l = 1, \ldots, L$, that is, there will be an edge between all labelling functions and the true, latent label y. These edges encode the idea that each labelling function is related to the true label by an accuracy. Finally, there will also be edges between some labelling functions, (V_i, V_j) for $(i, j) \subseteq \{(l, k) : l, k = 1, \ldots, L \text{ and } l \neq k\}$, that is, there will be edges between some, but not necessarily all, of the labelling functions. These edges encode the idea that some of the labelling functions are related to each other by a correlation. An example is illustrated to the left in Figure 3.5.

In Snorkel, the graph structure in $\mathcal{G}$ will be based on user input. The vertices and accuracy edges will be defined implicitly by the labelling functions, however, the user will have to define the correlation edges, that is, which labelling functions should be modelled as correlated. Based on the input correlations, the graph $\mathcal{G}$ can be defined. The next step in Snorkel is then to create a junction tree representation of the graph $\mathcal{G}$. Before proceeding, an introduction to junction trees is necessary.

Junction Trees

A junction tree is a decomposition of a graph. The junction tree is formulated by a set of vertex subsets, $\mathcal{X} = \{X_1, \ldots, X_{N_\mathcal{X}}\}$ where $X_i \subset \mathcal{V}$, and a tree structure $\mathcal{T}$. The junction tree can then be described by a new graph where the vertices are given by the subsets $X_1, \ldots, X_{N_\mathcal{X}}$, and the edges are given in the tree structure $\mathcal{T}$. Furthermore, to satisfy the conditions of a junction tree, the decomposition $(\mathcal{X}, \mathcal{T})$ must have the following properties:

- The union of the subsets $X_1, \ldots, X_{N_\mathcal{X}}$ must equal $\mathcal{V}$, i.e., $\{X_1 \cup \cdots \cup X_{N_\mathcal{X}}\} = \mathcal{V}$.
- For every edge $(V_l, V_k) \in \mathcal{E}$, there must be a subset $X_i \in \mathcal{X}$ that contains both V_l and V_k.
- If a vertex V_l is both in X_i and X_j, then V_l must also be in all subsets that are on the path between X_i and X_j in the tree $\mathcal{T}$.

Note that a junction tree is not unique, and can be constructed in many ways.

In Snorkel, the graph $\mathcal{G}$ will be transformed to a junction tree with some special properties. In particular, the subsets $X_1, \ldots, X_{N_\mathcal{X}}$ will be defined by a set of *maximal cliques* and *singleton separator sets*. We want a junction tree of this particular form, because such trees have some nice properties, which makes it easier to estimate the model parameters $\boldsymbol{\mu}$ of the model $P_{\boldsymbol{\mu}}(y, \boldsymbol{\lambda})$. Before proceeding with these properties, however, the terms maximal cliques and singleton separator sets must be explained.

A clique is a subset of vertices in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where there is an edge between all vertices in the clique. Thus, if $\mathcal{V} = \{V_0, V_1, \ldots, V_L\}$, then $C \subseteq \{0, 1, \ldots, L\}$ such that $i, j \in C \implies (V_i, V_j) \in \mathcal{E}$. Furthermore, the clique C is referred to as a maximal clique if it cannot be expanded by including another vertex from the graph.

A separator set is an intersection between two adjacent cliques in a graph, i.e., $S = C_i \cap C_j$. The separator set is further referred to as a singleton separator set if it only contains a single element.

Now, as previously stated, we want a junction tree where the subsets $X_1, \ldots, X_{N_\mathcal{X}}$ are formed by maximal cliques and singleton separator sets. However, such a representation

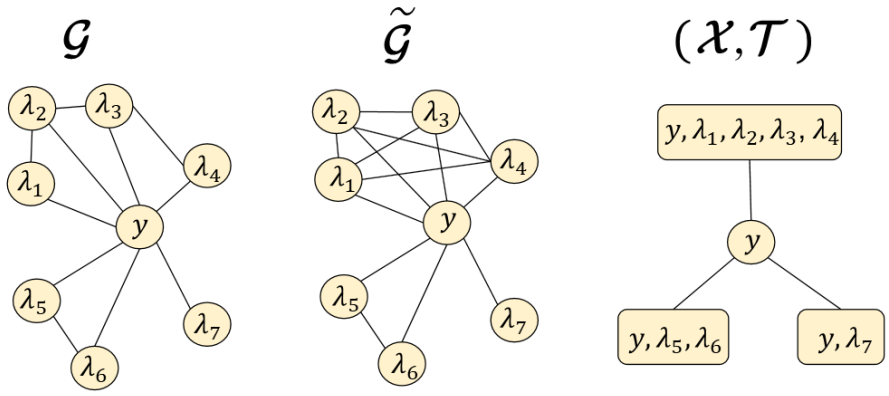

Figure 3.5: An illustration of an input graph $\mathcal{G}$ to the left, the augmented graph $\tilde{\mathcal{G}}$ in the middle, and the junction tree $(\mathcal{X}, \mathcal{T})$ with maximal cliques and singleton separator sets to the right. In this example, the labelling functions $\lambda_1, \lambda_2, \lambda_3$ and λ_4, as well as λ_5 and λ_6, will be modelled as correlated. This results in three maximal cliques $C_1 = \{0, 1, 2, 3, 4\}$, $C_2 = \{0, 5, 6\}$ and $C_3 = \{0, 7\}$.

might not exist for the user-defined input graph $\mathcal{G}$. Therefore, in Snorkel, the input graph will be augmented such that maximal cliques and singleton separator sets are obtained. In particular, cliques will be obtained by first splitting the labelling functions $\lambda_1, \ldots, \lambda_L$ into non-intersecting groups, based on the correlation edges in $\mathcal{E}$. If $(V_l, V_k) \in \mathcal{E}$, then λ_l and λ_k must belong to the same group. However, the groups will be kept as small as possible under these constraints. This means that if λ_l has no correlation edge in the input graph $\mathcal{G}$, it will be the only element in its group. Finally, edges will be added to the non-intersecting groups, such that they form cliques. This augmentation results in an new graph $\tilde{\mathcal{G}}$. An example is illustrated in the middle in Figure 3.5.

Note then that the labelling function cliques in $\tilde{\mathcal{G}}$ are not maximal, since all labelling functions have an edge to the true, latent label y. However, by adding y to all labelling function cliques, we obtain a set of maximal cliques. By doing this, all cliques become adjacent, and y becomes the intersection, that is, singleton separator set, between them. Thus, the augmented graph $\tilde{\mathcal{G}}$ can be represented by a set of maximal cliques $\mathcal{C} = \{C_1, \ldots, C_{N_\mathcal{C}}\}$, and a set of singleton separator sets $\mathcal{S} = \{S\}$, where $S = \{0\}$ since $y = V_0$. A junction tree can then be defined such that $\mathcal{X} = \mathcal{C} \cup \mathcal{S}$, and $\mathcal{T}$ becomes an arbitrary tree structure that connects the maximal cliques. An example is illustrated to the right in Figure 3.5.

The reason why Snorkel wants to represent the input graph $\mathcal{G}$ in such a junction tree, is because it has some nice properties. These will now be stated, while more details can be found in the work of Loh and Wainwright (2013).

- The covariance matrix of $\tilde{\mathcal{G}}$, denoted $\mathbf{\Sigma}$, is invertible, and the inverse covariance matrix is block-structured. In particular, if $(V_i, V_j) \notin \tilde{\mathcal{E}}$, then $\mathbf{\Sigma}^{-1}_{i,j} = 0$.

- The probability distribution of the graph $\tilde{\mathcal{G}}$ can be factorized to the form

$$P(V_0, V_1, \ldots, V_L) = \frac{\prod_{C \in \mathcal{C}} P(\mathbf{V}_C)}{\prod_{S \in \mathcal{S}} P(\mathbf{V}_S)}, \tag{3.24}$$

  where $P(\mathbf{V}_C)$ is the marginal probability of observing the values V_i for $i \in C$, and $P(\mathbf{V}_S)$ is the marginal probability of observing the values V_j for $j \in S$.

Now, the graphical model that Snorkel is based on, $\tilde{\mathcal{G}}$, has been introduced. We can finally proceed with specifying the model $P_{\boldsymbol{\mu}}(y, \boldsymbol{\lambda})$, and the corresponding parameter vector $\boldsymbol{\mu}$.

Defining the Parameter Vector

The result in (3.24) shows that the joint probability distribution for y and $\boldsymbol{\lambda}$ is given by

$$P_{\boldsymbol{\mu}}(y, \boldsymbol{\lambda}) = \frac{\prod_{C \in \mathcal{C}} P(\mathbf{V}_C)}{\prod_{S \in \mathcal{S}} P(\mathbf{V}_S)}, \tag{3.25}$$

where $V_0 = y, V_1 = \lambda_1, \ldots, V_L = \lambda_L$. However, we still need to determine $P(\mathbf{V}_C)$ and $P(\mathbf{V}_S)$. In our specific case with singleton separator sets, the latter simply becomes $P(\mathbf{V}_S) = P(y)$, that is, the class balance. In Snorkel, this class balance can either be given as input or be estimated. The estimation process will not be given here but is described by Ratner et al. (2019).

Determining $P(\mathbf{V}_C)$, on the other hand, is a little more tricky. This is where the parameter vector $\boldsymbol{\mu}$ comes into play. We stated earlier that this is a vector of probabilities. Specifically, we want to construct $\boldsymbol{\mu}$ such that it contains $P(\mathbf{V}_C)$ for each $C \in \mathcal{C}$, and for each possible input combination of $(y, \boldsymbol{\lambda})$. This can be done the following way.

We first define an indicator random variable for the event that the vertices in a clique take a specific set of values $\mathbf{y}_C$, i.e.,

$$\psi(C, \mathbf{y}_C) = I\big(\cap_{i \in C} V_i = (\mathbf{y}_C)_i\big). \tag{3.26}$$

Note that since this is an indicator random variable, its expected value $E\big(\psi(C, \mathbf{y}_C)\big)$ can be interpreted as the marginal probability of observing the values $\mathbf{y}_C$ in the clique C, i.e., $E\big(\psi(C, \mathbf{y}_C)\big) = P(\mathbf{V}_C = \mathbf{y}_C)$.

Now, to be able to determine the probability $P(\mathbf{V}_C)$ for any input values of $(y, \boldsymbol{\lambda})$, we need to know $E\big(\psi(C, \mathbf{y}_C)\big)$ for all combinations of $\mathbf{y}_C$ except one. The last combination is not needed since its probability can be determined as one minus the rest. Thus, we define $\boldsymbol{\psi}(C)$ as the vector of indicator random variables $[\psi(C, \mathbf{y}_C)]$ for all combinations of $\mathbf{y}_C$ except one, i.e.,

$$\boldsymbol{\psi}(C) = [\psi(C, \mathbf{y}_C)]_{\mathbf{y}_C \in \mathcal{Y}_C \setminus (\mathcal{Y}_C)_0}. \tag{3.27}$$

Here, $\mathcal{Y}_C$ is the set of all combinations of $\mathbf{y}_C$, and $(\mathcal{Y}_C)_0$ is an arbitrary combination which we choose to exclude. We then note that the expected value $E\big(\boldsymbol{\psi}(C)\big)$ contains the necessary information to determine $P(V_C)$ for any input value of $(y, \boldsymbol{\lambda})$.

Finally, we need the expected value $E\big(\boldsymbol{\psi}(C_m)\big)$ for all maximal cliques $C_m \in \mathcal{C}$. We gather the indicator random variable vectors for all cliques in a single, final vector

$$\boldsymbol{\psi}(\mathcal{C}) = [\boldsymbol{\psi}(C_1), \ldots, \boldsymbol{\psi}(C_{N_\mathcal{C}})]. \tag{3.28}$$

Then, the expected value $E\big(\boldsymbol{\psi}(\mathcal{C})\big)$ contains all the necessary information to determine $P(V_{C_m})$ for all maximal cliques $C_m \in \mathcal{C}$ and all input values of $(y, \boldsymbol{\lambda})$. Thus, we can finally define

$$\boldsymbol{\mu} = E\big(\boldsymbol{\psi}(\mathcal{C})\big). \tag{3.29}$$

Note that this is a minimal sufficient statistic for the graphical model $P_{\boldsymbol{\mu}}(y, \boldsymbol{\lambda})$.

Inference

The final step in completing the model $P_{\boldsymbol{\mu}}(y, \boldsymbol{\lambda})$ is to determine the parameter vector $\boldsymbol{\mu}$. The challenge here is that we do not observe the latent, true label y. The objective is therefore to determine $E\big(\boldsymbol{\psi}(\mathcal{C})\big)$ without observing y.

To do this, we first split the maximal cliques $\mathcal{C}$ into the observable part $\mathcal{O}$ and unobservable part $\mathcal{S}$, as

$$\mathcal{O} = \{C \backslash \mathcal{S} : C \in \mathcal{C}\} \qquad \mathcal{S} = \{0\}, \tag{3.30}$$

where $C \backslash \mathcal{S}$ denotes all the elements in a maximal clique C except V_0, since this corresponds to the unobserved y. Note then that the columns of $\boldsymbol{\psi}(\mathcal{O})\boldsymbol{\psi}(\mathcal{S})^{\mathrm{T}}$ correspond to $\boldsymbol{\psi}(\mathcal{C})$, and thus, if we can determine $E\big(\boldsymbol{\psi}(\mathcal{O})\boldsymbol{\psi}(\mathcal{S})^{\mathrm{T}}\big)$, we can estimate $\boldsymbol{\mu}$. Thus, we have

$$\hat{\boldsymbol{\mu}} = E\big(\boldsymbol{\psi}(\mathcal{O})\boldsymbol{\psi}(\mathcal{S})^{\mathrm{T}}\big) = E\big(\boldsymbol{\psi}(\mathcal{O})\big)E\big(\boldsymbol{\psi}(\mathcal{S})\big)^{\mathrm{T}} + \mathrm{Cov}\big(\boldsymbol{\psi}(\mathcal{O}), \boldsymbol{\psi}(\mathcal{S})\big). \tag{3.31}$$

Now, $E\big(\boldsymbol{\psi}(\mathcal{O})\big)$ is easy to estimate from the observed label matrix $\boldsymbol{\Lambda} = [\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_M]^{\mathrm{T}}$, since $\boldsymbol{\psi}(\mathcal{O})$ corresponds to the observable part of the augmented graph $\tilde{\mathcal{G}}$, i.e., $(V_1, \ldots, V_L) = (\lambda_{m1}, \ldots, \lambda_{mL})$ for $m = 1, \ldots, M$. Furthermore, $E\big(\boldsymbol{\psi}(\mathcal{S})\big)$ is simply given by the class balance $P(y)$, which, as previously mentioned, can be estimated as described by Ratner et al. (2019). Thus, the last element we need to determine is $\mathrm{Cov}\big(\boldsymbol{\psi}(\mathcal{O}), \boldsymbol{\psi}(\mathcal{S})\big)$. To do this, we investigate $\mathrm{Cov}\big(\boldsymbol{\psi}(\mathcal{O} \cup \mathcal{S})\big)$.

We noted in the section about junction trees that when the graph $\tilde{\mathcal{G}}$ can be represented as a junction tree with maximal cliques and singleton separator sets, then its inverse covariance matrix will be block structured. Ratner et al. (2019) extends this result and shows that also $\mathrm{Cov}\big(\boldsymbol{\psi}(\mathcal{O} \cup \mathcal{S})\big)^{-1}$ is sparse and block-structured.

Thus, by defining

$$\mathrm{Cov}\big(\boldsymbol{\psi}(\mathcal{O} \cup \mathcal{S})\big) \equiv \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{\mathcal{O}} & \boldsymbol{\Sigma}_{\mathcal{OS}} \\ \boldsymbol{\Sigma}_{\mathcal{OS}} & \Sigma_{\mathcal{S}} \end{bmatrix} \qquad \boldsymbol{\Sigma}^{-1} = \mathbf{K} = \begin{bmatrix} \mathbf{K}_{\mathcal{O}} & \mathbf{K}_{\mathcal{OS}} \\ \mathbf{K}_{\mathcal{OS}} & K_{\mathcal{S}} \end{bmatrix}, \tag{3.32}$$

we can note two tings: First, we have $\mathrm{Cov}\big(\boldsymbol{\psi}(\mathcal{O}), \boldsymbol{\psi}(\mathcal{S})\big) = \boldsymbol{\Sigma}_{\mathcal{OS}}$. Thus, $\boldsymbol{\Sigma}_{\mathcal{OS}}$ is the only missing piece to estimate $\boldsymbol{\mu}$. Our main goal now is therefore to estimate $\boldsymbol{\Sigma}_{\mathcal{OS}}$. Secondly, $\boldsymbol{\Sigma}^{-1} = \mathbf{K}$ is sparse and block-structured, such that $\mathbf{K}_{i,j} = 0$ whenever i, j corresponds to different cliques. This important property can be utilized to estimate $\boldsymbol{\Sigma}_{\mathcal{OS}}$.

It is shown by Ratner et al. (2019) that the following holds:

$$\mathbf{K}_{\mathcal{O}} = \boldsymbol{\Sigma}_{\mathcal{O}}^{-1} + \mathbf{z}\mathbf{z}^{\mathrm{T}}, \tag{3.33}$$

where $\mathbf{z} = \sqrt{c}\boldsymbol{\Sigma}_{\mathcal{O}}^{-1}\boldsymbol{\Sigma}_{\mathcal{OS}}$ and $c = (\Sigma_{\mathcal{S}} - \boldsymbol{\Sigma}_{\mathcal{OS}}^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathcal{O}}^{-1}\boldsymbol{\Sigma}_{\mathcal{OS}})^{-1}$. In (3.33), $\boldsymbol{\Sigma}_{\mathcal{O}}$ is known while $\mathbf{K}_{\mathcal{O}}$ and $\mathbf{z}$ are unknown. However, we know the sparsity structure of $\mathbf{K}_{\mathcal{O}}$, and we denote

the set of entries (i, j), when i and j correspond to different cliques in $\mathbf{K}_{\mathcal{O}}$, as Ω. Then, $\mathbf{z}$ can be estimated by solving the matrix completion problem

$$\hat{\mathbf{z}} = \arg\min_{\mathbf{z}} \ ||\boldsymbol{\Sigma}_{\mathcal{O}}^{-1} + \mathbf{z}\mathbf{z}^{\mathrm{T}}||_{\Omega}, \tag{3.34}$$

where $||\cdot||_{\Omega}$ denotes the Frobenius norm of only the entries $(i, j) \in \Omega$. From the estimated $\hat{\mathbf{z}}$, the estimate $\hat{\boldsymbol{\Sigma}}_{\mathcal{OS}}$ can be calculated algebraically.

The complete inference procedure for estimating $\boldsymbol{\mu}$ is summarized in Algorithm 1.

Algorithm 1: Procedure for estimating the label model parameter vector $\boldsymbol{\mu}$.

Input: Observed labelling rates $\hat{E}\big(\boldsymbol{\psi}(\mathcal{O})\big)$ and covariance matrix $\hat{\boldsymbol{\Sigma}}_{\mathcal{O}}$, class balance $\hat{E}\big(\boldsymbol{\psi}(\mathcal{S})\big)$ and variance $\Sigma_{\mathcal{S}}$, correlation sparsity structure Ω.

$\hat{\mathbf{z}} \leftarrow \arg\min_{\mathbf{z}} \ ||\hat{\boldsymbol{\Sigma}}_{\mathcal{O}}^{-1} + \mathbf{z}\mathbf{z}^{\mathrm{T}}||_{\Omega}$

$\hat{c} \leftarrow \Sigma_{\mathcal{S}}^{-1}(1 + \hat{\mathbf{z}}^{\mathrm{T}}\hat{\boldsymbol{\Sigma}}_{\mathcal{O}}\hat{\mathbf{z}})$

$\hat{\boldsymbol{\Sigma}}_{\mathcal{OS}} \leftarrow \hat{\boldsymbol{\Sigma}}_{\mathcal{O}}\hat{\mathbf{z}}/\sqrt{\hat{c}}$

$\hat{\boldsymbol{\mu}} \leftarrow \hat{E}\big(\boldsymbol{\psi}(\mathcal{O})\big)\hat{E}\big(\boldsymbol{\psi}(\mathcal{S})\big)^{\mathrm{T}} + \hat{\boldsymbol{\Sigma}}_{\mathcal{OS}}$

Return: $\hat{\boldsymbol{\mu}}$

Training Models

Once the parameters of the model $P_{\boldsymbol{\mu}}(y, \boldsymbol{\lambda})$ have been determined, the latent labels $\mathbf{y} = (y_1, \ldots, y_M)$ can finally be predicted by the label model $P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda})$, as described in (3.23). Note that the output from the label model is not a label $y \in \{-1, 1\}$, but instead a *probabilistic label* $y^+ = P_{\boldsymbol{\mu}}(y = 1|\boldsymbol{\lambda}) \in [0, 1]$. A probabilistic label y^+ can then be transformed to a standard predicted label by calculating $\hat{y} = I(y^+ \geq 0.5) - I(y^+ < 0.5) \in \{-1, 1\}$.

However, the goal in weak supervision is not to make a classifier based on the labelling functions. Instead, the goal is to train supervised models, and since we do not have hand-made labels, we use the weak supervision label model to obtain labels. These are expected to be noisy, and we therefore train supervised models that we believe can generalize better, and thereby become superior to the label model, even though it is trained on the labels from the label model. With this goal in mind, we want to make the training process as informed as possible. The weak supervision labels are noisy, and the probabilistic label y^+ contains some of this noise. We therefore want to train our models on the probabilistic labels, instead of transforming them to standard ones.

When training the parameters or weights $\mathbf{w}$ of a machine learning model $h_{\mathbf{w}}(x)$, the objective is generally to minimize some loss function $l(h_{\mathbf{w}}(x), y)$ over a set of data samples, that is, to solve

$$\arg\min_{\mathbf{w}} \sum_{m=1}^{M} l(h_{\mathbf{w}}(x_m), y_m). \tag{3.35}$$

This training objective can be altered to take into account the information in $y_m^+ = P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda}_m)$, by defining a *noise-aware loss function*. A more informed learning objec-

tive becomes solving

$$\arg\min_{\mathbf{w}} \sum_{m=1}^{M} E_{y\sim P_{\boldsymbol{\mu}}(y|\boldsymbol{\lambda}_m)}\big[l\big(h_{\mathbf{w}}(x_m), y\big)\big], \tag{3.36}$$

where $E_{y\sim P_{\boldsymbol{\mu}}(y|\boldsymbol{\lambda}_m)}[\cdot]$ is the expected value under the assumption that y is a random variable with probability distribution given by $P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda}_m)$. We then have

$$\begin{aligned} E_{y\sim P_{\boldsymbol{\mu}}(y|\boldsymbol{\lambda})}\Big[l\big(h_{\mathbf{w}}(x_m), y\big)\Big] &= \sum_{y} \Big(P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda}_m) \cdot l\big(h_{\mathbf{w}}(x_m), y\big)\Big) \\ &= y_m^+ \cdot l\big(h_{\mathbf{w}}(x_m), 1\big) \\ &\quad + (1 - y_m^+) \cdot l\big(h_{\mathbf{w}}(x_m), -1\big). \end{aligned} \tag{3.37}$$

In this work, we will train models by using the noise-aware learning objective for a given loss function $l\big(h_{\mathbf{w}}(x_m), y\big)$, as defined in (3.36) and (3.37).

The necessary background theory has now been introduced. In the next chapter, the experimental setup will be given.

Chapter 4

Experimental Setup

The objective of this work is to measure the quality of summaries for real estate condition reports. The end goal is therefore to make one or more models that are able to capture the quality of the summaries. Since the real estate condition reports are unlabelled, we first construct a weak supervision model, such that noisy labels can be obtained. Once we have labels, we can use supervised methods. We will then propose various supervised architectures for measuring summary quality. In particular, the model architectures of this work will be attempts to map reports and summaries to the conceptual summary content space, as described in Section 1.3.1.

In the following chapter, the complete experimental setup will be presented. First, the dataset of real estate condition reports will be formally introduced. Then, the weak supervision model, with corresponding labelling functions, will be defined. Finally, model architectures for mapping reports and summaries to the summary content space will be proposed.

4.1 The Dataset

The company Vendu was briefly introduced in Section 1.1. In cooperation with Norsk Takst, they have prepared a large amount of real estate condition reports for analysis. These reports have corresponding summaries, and the objective of this work is to examine methods for automatically measuring the quality of these summaries.

More specifically, the dataset consists of $96\,534$ real estate condition reports. A real estate condition report consists of the following parts:

- Textual descriptions of various parts of the real estate.
- Textual condition assessments for various parts of the real estate.
- Condition degrees (TG) for various parts of the real estate. Can be in the range 0–3, where 0 indicates perfect condition and 3 indicates a very bad condition.
- The summary.

- Descriptions of the environment around the real estate.
- Metadata for the real estate and the condition report, for instance, size, building year, the author of the report, date of assessment, etc.

In this work, the collection of textual descriptions and condition assessments for a report will be regarded as the complete report text, denoted $\mathbf{r}$, while the summary text will be denoted $\mathbf{s}$. These texts will be regarded as separate documents. Thus, the dataset contains a total of $M = 193\ 068$ documents, with $M_{\mathbf{r}} = 96\ 534$ report documents, $\mathcal{D}_{\mathbf{r}} = \{\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_{M_{\mathbf{r}}}\}$, and $M_{\mathbf{s}} = 96\ 534$ corresponding summary documents, $\mathcal{D}_{\mathbf{s}} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_{M_{\mathbf{s}}}\}$. For these documents, the average report length is 1302 words, while the average summary length is 187 words.

The information concerning which parts of the report $\mathbf{r}$ belongs to which part of the real estate, and the condition degree of that part, could also be useful information for the models. This information is, however, very specific for the real estate domain, and model architectures based on this information could not have been used for the general task of measuring summary quality for other domains. The main focus of this work will therefore be on models that are purely based on the report document $\mathbf{r}$ and the summary document $\mathbf{s}$.

4.1.1 Defining a Good Summary

The goal of this work is to measure the quality of summaries. For that occasion, it is appropriate to define what a good summary really is. This is, in itself, a challenge. There are possibly an infinite number of ways to write a good summary, and if humans were to evaluate summaries, the resulting assessments would probably be relatively subjective.

However, we will do our best to describe what a good summary is. And to do that, we look to the Norwegian standard "NS3600:2018 – Teknisk tilstandsanalyse ved omsetning av bolig"[9]. This document describes how a real estate condition report should be written, and also includes a brief description of what a summary should include. The following is listed:

1. An overall professional assessment.
2. An overview of where TG2 and TG3 have been registered, with a reference to where the corresponding textual condition assessment is found in the report.
3. An overview of places where an assessment has not been performed
4. Any further recommendations for investigation.
5. If rooms for habitation are approved.
6. If there are deviations from regulations concerning escape routes, daylight surface and roof height.

[9] https://www.standard.no/fagomrader/bygg-anlegg-og-eiendom/teknisk-tilstandsanalyse-av-bolig---ns-3600/

7. The technical value of the real estate. This is estimated based on how much it would cost to build a new real estate equivalent to the one being assessed. Eventual costs due to wear and tear, old age or faults are subtracted from the technical value.

These points imply what a good summary should look like. When defining a weak supervision model in the next section, the goal will be to find rules that can indicate whether or not summaries are up to this standard.

4.2 Weak Supervision Model

Since the data is unlabelled, a weak supervision label model, as defined in Section 3.4.4 will be constructed. This will be used to create probabilistic labels for the quality of the summaries, which in turn can be used to train supervised models. In this section, this label model will be defined. In particular, the labelling functions that will be applied will be presented

4.2.1 Labelling Functions for Summary Quality

The choice of labelling functions is likely to have a big impact on the end result of this work. After all, the models of this work will be based entirely on the output from the labelling functions. It is therefore very important that these are as accurate as possible. This is a challenge because it is difficult to find accurate rules that are good implications of quality. To achieve this, the labelling functions have been developed in cooperation with Vendu. The team at Vendu has done a lot of analysis on real estate condition reports and has thereby gained a lot of insight into what makes for a good summary. For simplicity, we choose to define only two possible outcomes for our summaries: They are either good or bad, denoted by $y = 1$ and $y = -1$, respectively. The labelling rules should therefore be indications of either a good or bad summary. The resulting labelling functions from the cooperation with Vendu are the following:

1. Summary shorter than 50 words. Implication: **Bad**.
2. Summary longer than 400 words. Implication: **Bad**.
3. TG3 for the bathroom, but no mention of the bathroom in summary. Implication: **Bad**.
4. TG3 for the kitchen, but no mention of the kitchen in summary. Implication: **Bad**.
5. TG3 for the roof, but no mention of the roof in summary. Implication: **Bad**.
6. TG2 or TG3 for the bathroom, with mention of the bathroom in summary. Implication: **Good**.
7. TG2 or TG3 for the kitchen, with mention of the kitchen in summary. Implication: **Good**.
8. TG2 or TG3 for the roof, with mention of the roof in summary. Implication: **Good**.

9. Correction of TG in the bathroom, but no mention of the bathroom in summary. Implication: **Bad**.

   (The correction of TGs is a part of Vendu's previous work with the condition reports and targets the idea that a correction might imply an error.)

10. Correction of TG in the kitchen, but no mention of the kitchen in summary. Implication: **Bad**.

11. Correction of TG on the roof, but no mention of the roof in summary. Implication: **Bad**.

12. Summary with LIKS-score over 55. Implication: **Bad**.

    (LIKS is a readability score used by Vendu. It is defined by

    $$\text{LIKS} = 100 \cdot \frac{\text{Number of long words}}{\text{Number of words}} + \frac{\text{Number of words}}{\text{Number of sentences}},$$

    where long words are words that have more than 6 letters. Hence, LIKS gives a higher score for texts with long words and sentences.)

13. Summary with OVR-score over 96. Implication: **Bad**.

    (OVR is another readability score used by Vendu, and is given by

    $$\text{OVR} = 100 \cdot \frac{\log(\text{Number of unique words})}{\log(\text{Number of words})}.$$

    Hence, OVR gives a higher score for texts with many unique words.)

14. An insurance claim has been raised on the real estate after the transaction. Implication: **Bad**.

15. Written by an agent with insurance claims on more than 7.5% of her reports. Implication: **Bad**.

16. Written by an agent with LIKS-score higher than 55 on more than 40% of her reports. Implication: **Bad**.

17. Written by an agent with OVR-score higher than 96 on more than 40% of her reports. Implication: **Bad**.

18. Written by an agent with fewer than 10 reports that year. Implication: **Bad**.

19. Fewer than 20% of the words in the summary are found in the report. Implication: **Bad**.

20. Fewer than 3% of the words in the report are found in the summary. Implication: **Bad**.

21. More than 70% of the words in the summary are also found in the report. Implication: **Good**.

22. More than 20% of the words in the report are also found in the summary. Implication: **Good**.

A summary should neither be too long nor too short, which is handled by rules 1–2. Rules 3–8 deal with the condition degrees, and in particular, the fact that a good summary should mention real estate parts that are in a bad condition. Rules 9–11 deal with corrections of condition degrees, and targets the idea that a correction might imply an error. Rules 12–13 deal with language scores, which imply a difficult language. Furthermore, rules 15–18 deal with patterns that might imply that an agent writes bad summaries, while rules 19–22 capture general semantic similarity.

These rules mainly target point 1 and 2 in the definition of a good summary from the last section. This is because point 3–7 are difficult to target with such rules. Furthermore, after reading quite a few summaries, we find that most summaries in practice mainly target the first two points. We therefore believe that point 1 and 2 are more important to cover with the labelling functions, and thus, point 3–7 in Section 4.1.1 might not be very well covered by the labelling functions above. However, if a summary is good according to point 1 and 2, it is more likely that it is written by a well-informed real estate agent that does a thorough job. We can therefore hope that many of the summaries that are good according to point 1 and 2 also fulfil point 3–7. If that is the case, then the various supervised methods might actually learn to pick up patterns related to these points, even if the weak supervision labelling functions do not cover them explicitly.

The above rules result in a set of $L = 22$ labelling functions $\boldsymbol{\lambda}(x) = \big(\lambda_1(x), \ldots, \lambda_{22}(x)\big)$. These can then be applied to the real estate condition reports, to obtain a label matrix $\boldsymbol{\Lambda}$. From this matrix, a label model can be made as described in Section 3.4.4.

Note that in the current version of Snorkel, it is not possible to model labelling functions as correlated. This will be implemented in a future version. This means that in the current version, the graphical model $\tilde{\mathcal{G}}$ will have $L = 22$ maximal cliques, and the label model $P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda})$ will only model the labelling function accuracies. If the models in this work are to be used in the future, it is recommended they be re-trained on new weak-supervision labels when Snorkel implements the correlation functionality.

4.2.2 Weak Supervision Objective

It is clear that the labelling functions in Section 4.2.1 include a lot of meta-information about the condition reports. To clarify any confusion, it should therefore be noted that the objective of this work is not to make models based on this meta-information. As discussed in Section 4.1, the input to a quality-measuring model should ideally only be the report document $\mathbf{r}$ and the summary document $\mathbf{s}$, since this would result in a more general, domain-independent model architecture for summary quality. This general input will be used in the proposed model architectures in this work.

However, the labelling functions in Section 4.2.1 are based on much more metadata. In particular, data about insurance claims and historical agent behaviours will generally not be available for new condition reports. This metadata, which is available only for a historical subset of real estate condition reports, will only be used in the weak supervision

label model, to create labels. The main models of this work will be trained on these labels, but they will only be given the inputs $\mathbf{r}$ and $\mathbf{s}$.

This is, in fact, a very important point. If the main models of this work were given the same input as the weak supervision label model, the main models would probably only learn to mimic the behaviour of the label model. This is not desirable, since the label model is expected to be imprecise, and can only be used on a subset of the condition reports. By giving the main models a more general input, they will not be able to mimic the labelling functions, thus, they will have to find other, underlying patterns that can describe the weak supervision labels. These are the real patterns of summary quality that we are trying to capture in the main models of this work.

4.3 Model Architectures

The end goal of this work is to create models that can measure the quality of summaries for real estate condition reports. As discussed in the introduction, this will be done by mapping reports and summaries to the conceptual summary content space, as described in Section 1.3.1. The summary quality will then be measured by the cosine similarity between the embedded report and summary.

In particular, three supervised architectures will be proposed in this work, in addition to baseline models, namely an FFN, LSTM and CNN, as described in Section 3.2. Before going into specifics of these architectures, however, we will give a general definition of a quality-measuring model, based on the idea of mapping reports and summaries to the summary content space.

4.3.1 Defining a General Quality-Measuring Model

The output from a model should be a measure of quality, with the most natural representation being a continuous number on a specific domain. Thus, any quality model, which we will denote $q(\cdot)$, should be a mapping from a condition report to a single continuous number. A condition report in this work will be represented by the report and summary documents, i.e., $(\mathbf{r}, \mathbf{s})$.

Let $\mathcal{R}$ denote the complete report space, i.e., $(\mathbf{r}, \mathbf{s}) \in \mathcal{R}$. A quality measuring model should then be a function q defined by

$$q : \mathcal{R} \to \mathcal{Q} \quad \text{where} \quad \mathcal{Q} = \{x \in \mathbb{R} : a \leq x \leq b\}, \tag{4.1}$$

where a and b are the lowest and highest possible quality measures, respectively.

Now, there are countless ways to make such a mapping, and it might be difficult to know where to start. Instead of doing this arbitrarily, a lot of inspiration can be found by looking to similar problems in NLP. In particular, the task of finding document similarity is one of the most similar tasks to the problem at hand. And in document similarity, the currently most common and efficient technique is to map documents to a semantic feature space that reflects the semantics of the documents, and then to measure similarity by using a mathematical similarity measure in that space. In particular, cosine similarity, which was introduced in (3.1) in Section 3.1.1, is the most commonly used similarity measure in NLP.

With this insight, there is reason to believe that a similar strategy is likely to be effective for the task of measuring summary quality. Instead of mapping documents to a semantic feature space, we want to map reports and summaries to a feature space that somehow represents the content that a good summary should have, that is, the summary content space. With this mapping, the quality of the summaries could then be measured by applying cosine similarity on the embedded reports and summaries.

In this work, this approach will be investigated. Thus, the proposed model architectures of this work will be functions h defined by

$$h : \mathcal{R} \to \mathcal{Z}, \quad \text{where} \quad \mathcal{Z} = \left\{ \mathbf{z_r}, \mathbf{z_s} \in \mathbb{R}^K \right\}. \tag{4.2}$$

Here $\mathbf{z_r}$ and $\mathbf{z_s}$ denotes K-dimensional feature vectors, and $\mathcal{Z}$ denotes the conceptual summary content space. The general form of a quality-measuring model in this work will then be given by

$$q(\mathbf{r}, \mathbf{s}) = \cos\,\mathrm{sim}\big(h(\mathbf{r}, \mathbf{s})\big) = \cos\,\mathrm{sim}(\mathbf{z_r}, \mathbf{z_s}), \tag{4.3}$$

where $\cos\,\mathrm{sim}(\cdot, \cdot)$ is defined in (3.1). Note that since we use cosine similarity, the resulting quality measures will be in the domain $q \in [-1, 1]$. In the following sections, various proposals for $h(\mathbf{r}, \mathbf{s})$ will be given. However, we will first define the training objective that we will use for the supervised model architectures.

Noise-Aware Cosine Embedding Loss

The general form of a quality-measuring model has now been defined in (4.3). The challenge of making a good model then lies in finding a good feature mapping $h(\mathbf{r}, \mathbf{s})$. This will be done by using supervised architectures, and thus, we need an appropriate training objective, with a corresponding loss function.

Now, when mapping the reports and summaries to the summary content space, a good mapping should yield a high cosine similarity for good summaries, and low cosine similarity for bad summaries. A natural measure of performance is then given by the loss function

$$l\big(h(\mathbf{r}, \mathbf{s}), y\big) = l(\mathbf{z_r}, \mathbf{z_s}, y) = \begin{cases} \max\big(0, \tau_{\text{good}} - \cos\,\mathrm{sim}(\mathbf{z_r}, \mathbf{z_s})\big), & y = 1 \\ \max\big(0, \cos\,\mathrm{sim}(\mathbf{z_r}, \mathbf{z_s}) - \tau_{\text{bad}}\big), & y = -1. \end{cases} \tag{4.4}$$

Here, $y = 1$ encodes the event that the summary $\mathbf{s}$ is good, while $y = -1$ encodes the event that the summary is bad.

Note that the above loss function also includes a threshold for good summaries τ_{good}, and a threshold for bad summmaries τ_{bad}. If a good summary gets a quality score $q = \cos\,\mathrm{sim}(\mathbf{z_r}, \mathbf{z_s}) \geq \tau_{\text{good}}$, a loss of 0 is obtained, which is the lowest possible value. Likewise, a loss of 0 is obtained if a bad summary gets a score $q \leq \tau_{\text{bad}}$. Since the objective of a supervised method is to minimize its loss function, it will attempt to give good summaries a score of $q \geq \tau_{\text{good}}$ whilst giving bad summaries a score of $q \leq \tau_{\text{bad}}$. Thus, τ_{good} defines how high quality a good summary at least should have according to the model, while τ_{bad} defines how low quality a bad summary at least should have. The loss function, for two different values of τ_{good} and τ_{bad} is illustrated in Figure 4.1.

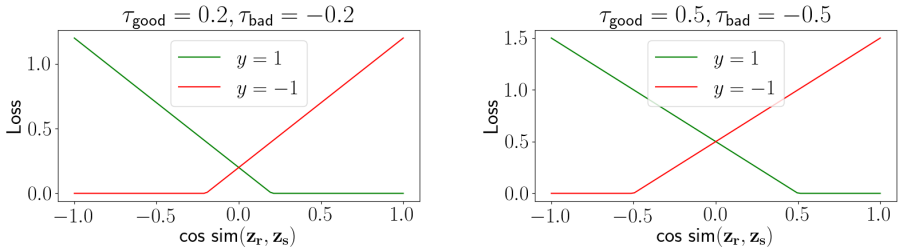

Figure 4.1: Summary quality loss function, as given in (4.4) for two different values of τ_{good} and τ_{bad}.

By setting $\tau_{\text{good}} = 1$ and $\tau_{\text{bad}} = 0$, we get a standard loss function, often referred to as *cosine embedding loss*. These values of τ_{good} and τ_{bad} are, however, not the best in this particular context. By using these threshold values, the resulting quality measures would typically be squeezed towards the edges of the domain, instead of giving a continuous quality measure. This would, effectively, make the models classifiers.

By instead using lower values for the thresholds, for example, $\tau_{\text{good}} = 0.2$ and $\tau_{\text{bad}} = -0.2$, the loss would return zero for good summaries if they have a quality higher than 0.2, and return zero for bad summaries if they have a quality lower than -0.2. This encourages the models to return qualities on a larger part of the domain, which is desirable in this context.

Now, the performance measure must be based on the weak supervision labels, since these are the only quality signal that we have. And the weak supervision labels are in fact probabilistic labels $y^+ = P_{\boldsymbol{\mu}}(y = 1 \mid \boldsymbol{\lambda})$. The probabilistic labels y^+ can easily be transformed to standard predicted labels, which will be denoted $\hat{y} \in \{-1, 1\}$. However, as discussed in Section 3.4.4, useful information would be lost in the process. Instead, it would be better to train the models by using a *noise-aware* version of the loss function in (4.4). The general notion of a noise-aware loss function was defined in (3.36).

Thus, a better training objective in the weak supervision setting can be defined by inserting the loss function in (4.4) into the general noise-aware version of a loss function, as defined in (3.37). We then get

$$\begin{aligned} l\big(h(\mathbf{r}, \mathbf{s}), y^+\big) &= E_{y \sim P_{\boldsymbol{\mu}}(y|\boldsymbol{\lambda})}\Big[l\big(h(\mathbf{r}, \mathbf{s}), y\big)\Big] = \sum_y \Big(P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda}) \cdot l\big(h(\mathbf{r}, \mathbf{s}), y\big)\Big) \\ &= y^+ \cdot \max\big(0, \tau_{\text{good}} - \cos\text{ sim}(\mathbf{z_r}, \mathbf{z_s})\big) \\ &+ (1 - y^+) \cdot \max\big(0, \cos\text{ sim}(\mathbf{z_r}, \mathbf{z_s}) - \tau_{\text{bad}}\big). \end{aligned} \tag{4.5}$$

This expression will be referred to as the *noise-aware cosine embedding loss*, and will be used in this work to both train and evaluate models. In particular, the training objective will be to minimize the average loss across the data, given by

$$\frac{1}{M} \sum_{m=1}^{M} l\big(h(\mathbf{r}_m, \mathbf{s}_m), y_m^+\big). \tag{4.6}$$

The choice of τ_{good} and τ_{bad} has a big impact on the resulting models. When we train models in this work, we will set $\tau_{\text{good}} = 0.2$ and $\tau_{\text{bad}} = -0.2$. In doing so, we define that the models should evaluate good summaries with a quality higher than $q = 0.2$, and bad summaries with a quality lower than $q = -0.2$. Furthermore, we would ideally like to obtain models that can identify good patterns that increase the quality measure and bad patterns that decrease the quality measure. Then, a summary with many good patterns would have a higher quality q than summaries with only a few good patterns, even if they both were good. Then, the resulting quality model would yield a continuous measure of quality on a large part of the cosine domain $[-1, 1]$. This is the behaviour we hope to obtain by training models on the loss function in (4.5).

Classification Scores

The noise-aware cosine embedding loss, as defined in (4.5), is a precise and informed performance measure. However, it does not really give a good intuition of how well the models are performing. To get a better understanding of how the various models are performing, classification scores will also be given.

To do this, the probabilistic labels y^+ must first be transformed to standard predicted labels $\hat{y}$. This can easily be done by calculating

$$\hat{y}(y^+) = I(y^+ \geq 0.5) - I(y^+ < 0.5), \tag{4.7}$$

that is, we predict the summary to its most probable class according to the label model $P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda})$. Furthermore, the models must be transformed to classification models. This will be done by calculating

$$y^*\big(q(\mathbf{r}, \mathbf{s}), \tau\big) = I(q(\mathbf{r}, \mathbf{s}) \geq \tau) - I(q(\mathbf{r}, \mathbf{s}) < \tau), \tag{4.8}$$

where τ is a quality threshold that determines where the line between a good and bad summary should go. Note that τ differs from τ_{good} and τ_{bad}, even though they all define what the quality of a good and/or bad summary should be. The difference is that τ_{good} and τ_{bad} are used in the training process to teach our models the desired behaviour, while τ is used to evaluate how good the models are at classification. Therefore, τ_{good} and τ_{bad} still are the thresholds that define how high/low quality score a good/bad summary should have, while τ is a threshold that only applies in the context of classification. For each model, we choose τ by the value that maximizes the accuracy, which is defined below.

The best threshold will first be determined on a validation set. Then, classification scores will be reported on a held-out test set. The following metrics will be reported in the results:

$$\begin{aligned}
\text{accuracy} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}, \\
\text{precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}}, \\
\text{recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \\
F_1 \text{ score} &= 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},
\end{aligned} \tag{4.9}$$

Table 4.1: Confusion matrix, with illustrations of TP, TN, FP and FN.

		Weak supervision labels	
		Good ($\hat{y} = 1$)	Bad ($\hat{y} = -1$)
Pred. labels	Good ($y^* = 1$)	TP	FP
	Bad ($y^* = -1$)	FN	TN

where TP is the number of *true positives*, given by TP $= \sum_{m=1}^{M} I(y_m^* = \hat{y}_m = 1)$, TN is the number of *true negatives*, given by TN $= \sum_{m=1}^{M} I(y_m^* = \hat{y}_m = -1)$, FP is the number of *false positives*, given by FP $= \sum_{m=1}^{M} I(y_m^* \neq \hat{y}_m = -1)$ and FN is the number of *false negatives*, given by FN $= \sum_{m=1}^{M} I(y_m^* \neq \hat{y}_m = 1)$. An illustration of TP, TN, FP and FN is also shown in Table 4.1. Such a table is often referred to as a confusion matrix.

Now, the general form of a quality-measuring model, the model training objective and an additional model performance measure have been defined in (4.3), (4.5) and (4.9), respectively. In the following subsections, various model proposals for $h(\mathbf{r}, \mathbf{s})$ will be proposed. These will then be implemented, and the results will be given in Chapter 5.

4.3.2 Baselines

We wish to compare the models proposed in this work to known baseline models. Since there is a lack of previous work on summary quality in general, especially for the real estate domain, we will have to look for baseline models that can solve similar tasks.

It is clear that a good summary should convey a lot of the same semantics as the report it is meant to summarize. Therefore, a general measure of document similarity between the report $\mathbf{r}$ and the summary $\mathbf{s}$ is expected to be a useful measure of summary quality. However, from the definition of a good summary in Section 4.1.1, it is also clear that not all parts of a report text are equally important for the summary. In particular, the condition degree for a real estate part gives a clear implication of its importance in the summary. Thus, general measures of document similarity are not expected to be very good. They are, however, the most meaningful currently available baseline.

Thus, for baseline models, $h(\mathbf{r}, \mathbf{s})$ in (4.3) will be chosen as general document embedding techniques that can map the reports $\mathbf{r}$ and summaries $\mathbf{s}$ to the semantic feature space. There are a wide range of available techniques that can do this. However, the length of the reports imposes a challenge for many of them. In particular, many methods are based on word vectors, which often are averaged when dealing with documents. However, for documents as long as ours, this is not expected to be a good strategy. Furthermore, the length of the reports exceeds the maximum limit of popular embedding techniques, like BERT (Devlin et al. 2018).

Still, there are some methods that can easily be applied to arbitrarily long documents. In this work, two of these will be included as baseline models: LSA, which maps documents to a latent topic space, and Doc2vec, which maps documents to a semantic feature space.

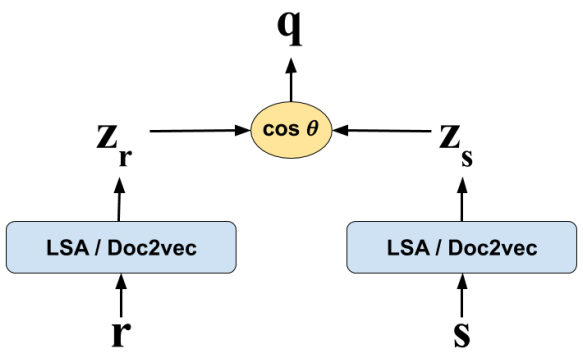

Figure 4.2: Illustration of baseline quality models $q(\mathbf{r}, \mathbf{s})$.

LSA

The theory and intuition behind LSA was introduced in Section 3.3.2. This model will be used as a baseline summary quality model. Thus, the quality measuring model in this case becomes as in (4.3), with

$$h_{\text{LSA}}(\mathbf{r}, \mathbf{s}) = \text{LSA}(\mathbf{r}), \text{LSA}(\mathbf{s}) = \mathbf{z}_\mathbf{r}, \mathbf{z}_\mathbf{s}, \tag{4.10}$$

where $\text{LSA}(\mathbf{r})$ and $\text{LSA}(\mathbf{s})$ denotes the semantic feature vectors of $\mathbf{r}$ and $\mathbf{s}$ when embedded by LSA. The resulting complete model $q(\mathbf{r}, \mathbf{s})$ is illustrated in Figure 4.2.

The model estimation procedure for LSA is given in Section 3.3.2. In this particular case, the LSA model will be made on a training set consisting of T real estate condition reports. Since the model is meant to create embeddings both for report documents $\mathbf{r}$ and summary documents $\mathbf{s}$, the model is trained on both sets of documents. This results in a training corpus of $M = 2T$ documents, given by $\mathcal{D} = \{\mathbf{r}_1, \mathbf{s}_1, \ldots, \mathbf{r}_T, \mathbf{s}_T\}$.

Doc2vec

The theory and intuition behind Doc2vec has also been introduced, in Section 3.3.4. Similarly to LSA, the quality measuring model now becomes as in (4.3), with

$$h_{\text{Doc2vec}}(\mathbf{r}, \mathbf{s}) = \text{Doc2vec}(\mathbf{r}), \text{Doc2vec}(\mathbf{s}) = \mathbf{z}_\mathbf{r}, \mathbf{z}_\mathbf{s}. \tag{4.11}$$

The resulting complete model $q(\mathbf{r}, \mathbf{s})$ is also illustrated in Figure 4.2.

The training procedure for Doc2vec has also been given, in Section 3.3.4. This model will be trained on the same set of documents that LSA was. Thus, the resulting training corpus has $M = 2T$ documents, given by $\mathcal{D} = \{\mathbf{r}_1, \mathbf{s}_1, \ldots, \mathbf{r}_T, \mathbf{s}_T\}$.

4.3.3 Embedder + FFN

The first supervised model proposal $h_{\text{FFN}}(\mathbf{r}, \mathbf{s})$, for mapping reports and summaries to the summary content space, is a fully connected feed-forward neural network, as described in

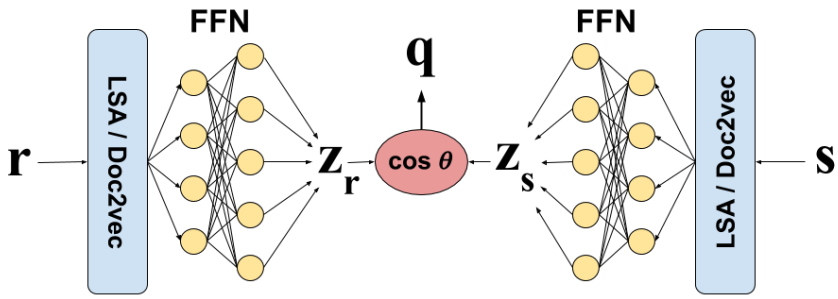

Figure 4.3: Illustration of embedder+FFN quality model architecture for $q(\mathbf{r}, \mathbf{s})$. The name "Embedder + FFN" simply means that the FFN is placed on top of the document embeddings from the embedder model.

Section 3.2.1. Such networks require numerical feature vectors as input, and in this work, we will use semantic feature vectors from LSA or Doc2vec as input to the FFN. This model architecture has therefore been named "Embedder + FFN", which simply means that the FFN is placed on top of the embedder models. The architecture is illustrated in Figure 4.3.

The embedded report and summary will be sent through the same neural network, with the same weights. This makes sense since the input report and summary are embedded by the same embedder. In the neural network, various numbers of layers will be tested, and the ReLu activation function will be employed in all layers except the last. The last layer will always be linear when the output embeddings $\mathbf{z_r}$ and $\mathbf{z_s}$ are created. Note therefore that by using only a single layer in the FFN, this model architecture becomes equivalent to applying a linear transformation to the full semantic feature vectors.

The intuition behind this model is the following: The input will be full semantic feature vectors. As previously discussed, some parts of this semantics are important to include in a summary, while other parts are irrelevant. By connecting an FFN, an arbitrary transformation of the full semantic feature space will be done. This transformation will be performance-driven, and thus, the resulting transformed space will contain the parts of the semantic vectors that best explain the weak supervision labels. If the labels are good, the resulting space should be a good representation of a summary content space.

In this model, training is done by first training LSA/Doc2vec the same way it was done in the baseline models, that is, on all individual documents $\mathcal{D} = \{\mathbf{r}_1, \mathbf{s}_1, \ldots, \mathbf{r}_M, \mathbf{s}_M\}$. The embedder is then applied to the same documents to obtain M training samples $(\mathbf{r}_1, \mathbf{s}_1), \ldots, (\mathbf{r}_M, \mathbf{s}_M)$, which are given as input to the FFN. Each sample $\mathbf{r}_m$ and $\mathbf{s}_m$ is then sent through the same FFN to obtain $\mathbf{z}_{\mathbf{r}_m}$ and $\mathbf{z}_{\mathbf{s}_m}$. The model weights are finally learned by minimizing the loss function $l(\mathbf{z}_{\mathbf{r}_m}, \mathbf{z}_{\mathbf{s}_m}, y_m^+)$ in (4.5) with $\tau_{\text{good}} = 0.2$ and $\tau_{\text{bad}} = -0.2$.

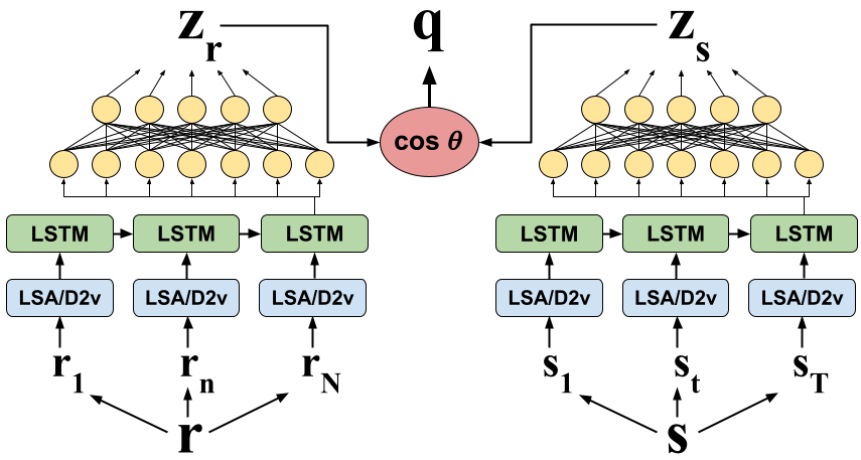

Figure 4.4: Illustration of embedder+LSTM quality model architecture for $q(\mathbf{r}, \mathbf{s})$. Here, D2v is short for Doc2vec, $\mathbf{r}_1, \ldots, \mathbf{r}_N$ denotes the N sections of the real estate condition report $\mathbf{r}$, and $\mathbf{s}_1, \ldots, \mathbf{s}_T$ denotes the T sentences of the summary $\mathbf{s}$. The name "Embedder + LSTM" simply means that the LSTM network is placed on top of the embedded sections/sentences from the embedder model LSA/Doc2vec. Note that the fully connected layer on top of the LSTM layer is always a single, linear layer.

4.3.4 Embedder + LSTM

The second supervised model proposal $h_{\text{LSTM}}(\mathbf{r}, \mathbf{s})$, for mapping reports and summaries to the summary content space, is an LSTM network. This type of network requires a sequence of numerical vectors as input, instead of a single vector. This will be obtained by dividing the reports into sections, and summaries into sentences, and then create semantic embeddings for the sections/sentences. Again, LSA and Doc2vec will be used as embedding techniques, and will be applied to the sections of the report and sentences of the summary, such that the report and summary can be represented as sequences of semantic feature vectors. The name "Embedder + LSTM" is therefore given to the model, which simply means that the LSTM is placed on top of the embedded sections/sentences. Finally, the output of the last sequence element from top LSTM layer will be sent through a fully connected linear layer to yield final embeddings $\mathbf{z}_\mathbf{r}$ and $\mathbf{z}_\mathbf{s}$. This model architecture is illustrated in Figure 4.4

Again, the same LSTM network will be used on the report and summary sequences, since the sequences consist of semantic feature vectors from the same embedder. Various numbers of LSTM layers, both one-directional and bi-directional, will be tested.

The intuition behind this model comes from a human perspective. If a person were to evaluate the quality of a summary, he would perhaps proceed with going through all the sections of the report, to decide which sections were important. Then, he would possibly go through the sentences of the summary to see if they covered the important sections of the report. This conceptual approach explains the idea behind splitting the reports into sections, and summaries into sentences.

Furthermore, the LSTM cell has some good qualities for this approach. In particular, the update gate, as described in Section 3.2.4, can specialize in determining which kinds of semantic content from the report is important, and let only this through to the internal cell state. Then, the output gate can specialize in returning a good representation of the content that a good summary should have. Thus, this scheme can possibly become a good mapping into the summary content space.

In the LSTM model, training will be done by first training the embedder LSA/Doc2vec, however, this time we want to train the embedder differently. Since they are to be applied to sections and sentences, they should also be trained on sections and sentences. Thus, to train the embedder model, a section/sentence corpus $\mathcal{D} = \{\mathbf{r}_{m,n}, \mathbf{s}_{m,t}\}$ for $n = 1, \ldots, N_{\mathbf{r}_m}$, $t = 1, \ldots, T_{\mathbf{s}_m}$ and $m = 1, \ldots, M$ will first be constructed. Note that $N_{\mathbf{r}_m}$ is the number of sections in the report $\mathbf{r}_m$, and $T_{\mathbf{s}_m}$ is the number of sentences in the summary $\mathbf{s}_m$. The embedder will first be trained on the section/sentence corpus, and then be applied to the same documents to obtain M training samples $(\mathbf{r}_1, \mathbf{s}_1), \ldots, (\mathbf{r}_M, \mathbf{s}_M)$, where $\mathbf{r}_m = (\mathbf{r}_{m,1}, \ldots, \mathbf{r}_{m,N_{\mathbf{r}_m}})$ and $\mathbf{s}_m = (\mathbf{s}_{m,1}, \ldots, \mathbf{s}_{m,T_{\mathbf{s}_m}})$. Each sample $\mathbf{r}_m$ and $\mathbf{s}_m$ will then be sent through the same LSTM network to obtain $\mathbf{z}_{\mathbf{r}_m}$ and $\mathbf{z}_{\mathbf{s}_m}$. The model weights are finally learned by minimizing the loss function $l(\mathbf{z}_{\mathbf{r}_m}, \mathbf{z}_{\mathbf{s}_m}, y_m^+)$ in (4.5) with $\tau_{\text{good}} = 0.2$ and $\tau_{\text{bad}} = -0.2$.

4.3.5 Embedder + CNN

The final supervised model architecture $h_{\text{CNN}}(\mathbf{r}, \mathbf{s})$, for mapping reports and summaries to the summary content space, is a CNN network, which was described in Section 3.2.5. On text data, this type of network generally has semantic word embeddings as input. To obtain word embeddings, two strategies will be tested: Applying an embedding layer, which was described in Section 3.2.3, and thus, training our own word embeddings; and using the word embedding technique Word2vec, which was presented in Section 3.3.3. The name "Embedder + CNN" is given to the model since the CNN network is placed on top of word embeddings, given either by an embedding layer or by Word2vec. Again, the same CNN network will be used on both the report $\mathbf{r}$ and the summary $\mathbf{s}$, since the same word embedder is used on both $\mathbf{r}$ and $\mathbf{s}$.

Let K be the dimensionality of the word embeddings. After embedding the words, the documents will be represented by two-dimensional matrices, where the K columns are the dimensions in the word embeddings, and the rows are the words in the documents. Then, two-dimensional convolutions will be applied. The convolution filters will have the same width as the word embeddings, i.e, K, while the filter height will be a hyperparameter for the model. In particular, we will use filters of different heights, which should be able to learn different patterns.

Since the width of the filters and the matrix representation of the documents are the same, given by K, the convolution output for each filter will be a one-dimensional vector. Padding will be employed, such that the convolution outputs have the same length as the input document. F different filter heights will be applied, with N_{F} filters for each filter height. Thus, there will be a total of $F \cdot N_{\text{F}}$ convolution output vectors. Then, the maximum of each vector will be taken, to yield a single $F \cdot N_{\text{F}}$ feature vector. Finally, this $F \cdot N_{\text{F}}$ feature vector will be sent through a fully connected linear network layer, and $\mathbf{z}_{\mathbf{r}}$ and $\mathbf{z}_{\mathbf{s}}$ will be obtained. F and K are hyperparameters in this model, and thus, various values F

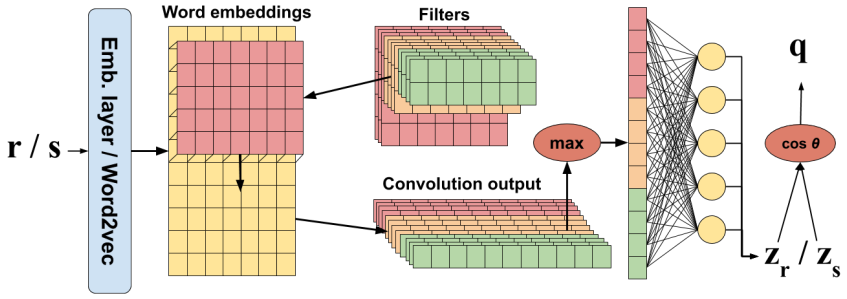

Figure 4.5: Illustration of the CNN quality model architecture for $q(\mathbf{r}, \mathbf{s})$. Note that the report $\mathbf{r}$ and summary $\mathbf{s}$ are not processed at the same time, but independently of each other.

and N_F will be tested. The final choice of hyperparameters will be discussed in the next chapter. The CNN architecture is illustrated in Figure 4.5.

The intuition behind this model is the following: The filters in this model are looking at a window of a certain number of words at a time. The filters have trainable weights, and will therefore learn to pick up patterns. By having a rather high amount of filters, with different filter heights, these filters should learn to pick up many different kinds of patterns. Since they are based on semantic word embeddings, these patterns do not even have to consist of the same words, as long as they have more or less the same meaning.

The fact that the filters move over the documents, and return a single value for each position, means that it does not matter how long the documents are. If a given pattern is present, it will be reflected somewhere in the convolution output vector. Then, by taking the maximum of each convolution output vector, the presence of the various patterns will be contained in the resulting $F \cdot K$ vector. In the context of summary quality, the filters can learn to pick up meanings that a good summary should have. Thus, the resulting mapping of $\mathbf{z_r}$ and $\mathbf{z_s}$ should be a good representation of the summary content space.

For the training process, when Word2vec is used as the embedder model, it has to be trained first. This will be done by training Word2vec on the individual documents $\mathcal{D} = \{\mathbf{r}_1, \mathbf{s}_1, \ldots, \mathbf{r}_M, \mathbf{s}_M\}$. Then, the embedding layer, or Word2vec, can be applied to the documents to obtain M samples $(\mathbf{r}_1, \mathbf{s}_1), \ldots, (\mathbf{r}_M, \mathbf{s}_M)$, where $\mathbf{r}_m$ and $\mathbf{s}_m$ are represented as sequences of words, i.e., $\mathbf{r}_m = (\mathbf{r}_{m,1}, \ldots, \mathbf{r}_{m,N_{\mathbf{r}_m}})$ and $\mathbf{s}_m = (\mathbf{s}_{m,1}, \ldots, \mathbf{s}_{m,T_{\mathbf{s}_m}})$. In this case, $N_{\mathbf{r}_m}$ denotes the number of words in the report $\mathbf{r}_m$, and $T_{\mathbf{s}_m}$ denotes the number of words in the summary $\mathbf{s}_m$. Each sample $\mathbf{r}_m$ and $\mathbf{s}_m$ can then be sent through the same CNN network to obtain $\mathbf{z}_{\mathbf{r}_m}$ and $\mathbf{z}_{\mathbf{s}_m}$. The model weights are learned by minimizing the loss function $l(\mathbf{z}_{\mathbf{r}_m}, \mathbf{z}_{\mathbf{s}_m}, y_m^+)$ in (4.5) with $\tau_\text{good} = 0.2$ and $\tau_\text{bad} = -0.2$.

4.4 Implementation

In this work, the analysis has been performed using Python. The implemented code can be found in the GitHub repository at `https://github.com/joakiol/RealEstateSummaryQuality`. The implementation is a major part of this work.

In particular, a lot of work related to data cleaning, structuring and processing has been performed, in addition to implementing and training the models of this work.

The models and methods of this work have been trained and tested using the resources provided by the NTNU IDUN/EPIC computing cluster (Själander et al. 2020). In particular, this computing cluster has enabled us to use GPUs in the training process, which has drastically reduced computation time.

Throughout the implementations, the natural language toolkit `nltk` (Loper and Bird 2002) has been used for pre-processing data. This mainly involves tokenizing the raw texts into words and sentences. Furthermore, the topic modelling library `gensim` (Rehurek and Sojka 2010) has been used for the embedding techniques LSA, Word2vec and Doc2vec. Finally, the deep learning architectures of this work, that is, FFN, LSTM and CNN, are implemented using PyTorch (Paszke et al. 2019). This is a deep learning Python library that has enabled us to efficiently implement a wide range of deep learning architectures.

Chapter 5

Results and Discussion

A method for obtaining noisy labels for the real estate condition report dataset has now been proposed, by way of the weak supervision model described in Section 4.2. Furthermore, various models for measuring summary quality have been proposed and implemented, as described in Section 4.3.

In this chapter, the labels from the weak supervision label model will first be analysed and discussed. Then, the results of the various summary quality models, when trained and evaluated on the weak supervision labels, will be presented and discussed. Finally, the models will be applied to the complete dataset of real estate condition reports, with the objective of analysing the general summary quality. This analysis can then shed some light on the high conflict rate for real estate transactions mentioned in Section 1.1.

5.1 Weak Supervision Labels

In this section, we first investigate the effect of the various labelling functions on the condition report dataset. Then, the resulting weak supervision labels will be investigated and visualized. Finally, a discussion will follow.

5.1.1 Labelling Function Analysis

The coverage, overlap, conflict and estimated accuracy percentages of the 22 labelling functions in Section 4.2.1, when applied to the real estate condition report dataset, are presented in Table 5.1. In this table, "coverage" is the percentage of condition reports that a labelling function actually labels, that is, does not abstain from labelling, "overlap" is the percentage of labelled samples where at least one other labelling function predicts a label, "conflict" is the percentage of labelled samples where at least one other labelling function gives a different label, and "estimated accuracy" is the labelling function accuracy estimated by the weak supervision label model $P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda})$. Note that the "overlap" and "conflict" percentages are relative to the "coverage" percentage. Thus, a labelling function generally predicts many labels if the coverage is high, it agrees with other labelling functions if the overlap is high while the conflict is low, and it disagrees with other functions if

the conflict is high.

First of all, we see from Table 5.1 that the overlap percentages in general are high. This means that the labelling functions often overlap, and there are few samples where only a single labelling function contributes to the probabilistic label. It is reassuring to see that most weak supervision labels are not the output of a single, imprecise labelling function, but instead a combination of many, which we expect will make the labels more robust.

Furthermore, we see that the rules with relatively high conflict numbers ($> 35\%$) are rule numbers 2 (long reports), 4 and 6-8 (TG 2/3 with/without mention of corresponding rooms in summary), 10 (correction of TG in the kitchen), 12 (OVR language difficulty score), 14 (insurance claim), 15-18 (written by an agent with certain patterns), 20 and 22 (many common words in report and summary). This does not necessarily mean that these rules are bad, but might instead show that a summary can be good on some points, whilst being bad at others. In the case of conflicting labelling functions, the result will be probabilistic labels y^+ that are not completely one-sided, that is, $y^+ \not\approx \{0, 1\}$. Instead, conflicting labelling functions will result in probabilistic labels that are distributed on the complete probability range $y^+ \in \langle 0, 1 \rangle$.

For the other rules, the conflict ratio is relatively low, which shows that many rules also agree on which summaries are good and bad. Therefore, there will also be many labels that are more one-sided with a high probability of the summary being either good or bad, that is, $y^+ \approx \{0, 1\}$. In conclusion, the overlap and conflict percentages indicate that the labels will contain both strong quality signals with $P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda}) \approx \{0, 1\}$, as well as weaker quality signals where $P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda}) \in \langle 0, 1 \rangle$.

Finally, we see from Table 5.1 that rule numbers 1, 7, 13, 17, 21 and 22 have particularly high estimated accuracies ($> 95\%$), while 2 and 14 have low accuracies ($< 50\%$). The rest are distributed on the range $[57\%, 84\%]$, where $\approx 70\%$ is typical. Rule number 2 (long summaries) is especially low, which is natural since we expect long summaries to be good on very many other points, and thus, rule number 2 has very many conflicts. The fact that this rule gets such a low accuracy indicates that the resulting label model does not properly penalize long summaries. As a consequence, we do not expect our final models to take long summaries into consideration either.

5.1.2 Label Analysis

After training and applying the label model in Section 4.2 on the real estate condition report data, a set of weak supervision labels is finally obtained. Out of $96\,534$ possible reports, the weak supervision model actually predicts a label for $81\,195$ of them. From these, a labelled dataset of $M_{\text{lab}} = 81\,195$ real estate condition reports can be constructed.

Since the labels are in fact probabilistic labels $y_m^+ = P_{\boldsymbol{\mu}}(y_m = 1 \mid \boldsymbol{\lambda}_m)$ for $m = 1, \ldots, M_{\text{lab}}$, the labels can be visualized by showing a histogram of $y_1^+, y_2^+, \ldots y_{M_{\text{lab}}}^+$. This histogram is shown in Figure 5.1. The figure shows that there are especially many labels where $P_{\boldsymbol{\mu}}(y_m = 1 | \boldsymbol{\lambda}_m) \approx 0$ and $P_{\boldsymbol{\mu}}(y_m = 1 \mid \boldsymbol{\lambda}_m) > 0.7$. Apart from this, the labels seem to be quite evenly distributed on the probability range $[0, 1]$.

Furthermore, we find that the average probability of a good label is given by

$$\frac{1}{M_{\text{lab}}} \sum_{m=1}^{M_{\text{lab}}} P_{\boldsymbol{\mu}}(y_m = 1 \mid \boldsymbol{\lambda}_m) = 0.493. \tag{5.1}$$

Table 5.1: Analysis of the effect of the labelling functions listed on pp. 45-47, when applied to the real estate condition report dataset.

LF No.	Implication	Coverage	Overlap	Conflict	Estimated Accuracy
1	Bad	10.4 %	96.2 %	22.1 %	100 %
2	Bad	7.9 %	91.1 %	82.3 %	10.9 %
3	Bad	5.1 %	90.2 %	27.5 %	71.5 %
4	Bad	2.4 %	95.8 %	50.0 %	58.5 %
5	Bad	2.6 %	92.3 %	30.8 %	78.0 %
6	Good	36.9 %	76.4 %	46.1 %	74.9 %
7	Good	11.6 %	93.1 %	47.4 %	97.3 %
8	Good	25.1 %	83.7 %	46.2 %	82.0 %
9	Bad	7.6 %	84.2 %	22.4 %	73.5 %
10	Bad	5.1 %	90.2 %	45.1 %	60.8 %
11	Bad	8.1 %	82.7 %	34.6 %	72.9 %
12	Bad	11.8 %	92.4 %	42.4 %	73.4 %
13	Bad	10.7 %	93.5 %	26.2 %	100 %
14	Bad	1.8 %	83.3 %	55.6 %	47.9 %
15	Bad	1.6 %	93.8 %	43.8 %	71.5 %
16	Bad	10.8 %	88.9 %	48.1 %	57.9 %
17	Bad	10.0 %	91.0 %	37.0 %	100 %
18	Bad	5.4 %	85.2 %	48.1 %	58.9 %
19	Bad	3.4 %	76.5 %	14.7 %	83.4 %
20	Good	6.3 %	85.7 %	49.2 %	63.3 %
21	Bad	7.1 %	94.4 %	11.3 %	100 %
22	Good	6.2 %	91.9 %	48.4 %	100 %

This shows that the dataset is very balanced. If the average probability for example was higher, then samples with a good label ($y^+ > 0.5$) would contribute more to the noise aware loss function in (4.5) than bad samples. As a consequence, the supervised learning methods would find it more important to correctly predict good samples, and less important to correctly predict bad samples. This unwanted behaviour could be overcome by oversampling bad samples in the training process. However, since the dataset in our case is very balanced, we do not have to oversample.

The labelled dataset of $M_{\text{lab}} = 81\,195$ labels will now be split into a training set, a validation set and a test set. This will be done in the ratio $0.8 : 0.1 : 0.1$. Thus, we obtain a training set of $M_{\text{train}} = 64\,955$ samples, a validation set of $M_{\text{val}} = 8\,120$ samples, and a test set of $M_{\text{test}} = 8\,120$ samples.

5.1.3 Weak Supervision Discussion

The resulting set of labels is quite large and covers 84% of the dataset. This is good since we obtain a large labelled dataset that we can train and evaluate on. Furthermore, the labelling functions appear to be relatively precise, with a typical accuracy around 70%, and they have high overlaps, such that most labels are a product of several labelling func-

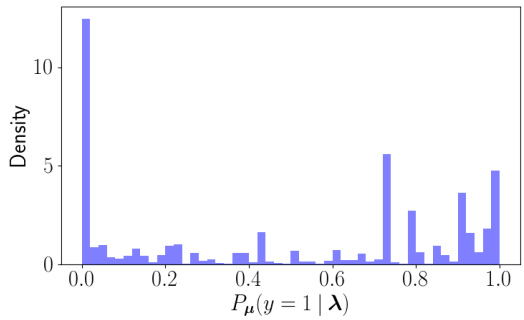

Figure 5.1: Histogram showing distribution of labels from the weak supervision label model, as described in Section 4.2.

tions. These facts indicate that the weak supervision labels should be relatively robust and precise, and we expect them to be a good basis for training and evaluating supervised methods.

The weak supervision label model is, in principle, a human-engineered feature-based system. We note that since the coverage is high (84%), it would, in fact, be possible to use a feature-based system directly as a summary quality model. If the coverage was lower, such a system would only be applicable on a few summaries, which is not that useful. However, since the coverage is so high, a feature-based system is a viable alternative to the weak supervision approach of this work. The main difference is that in weak supervision, the goal is to create models that can generalize beyond feature-based systems, and thus, become superior to them.

We should, however, keep in mind that when using weak supervision, the quality of the trained models will always be limited by the quality of the noisy labels. And in the following section, we will use the noisy labels not only to train, but also to evaluate model performances. These labels are, however, not the ground truth. We will get an impression that the model with the lowest noise-aware cosine embedding loss is the best, but in truth, the models might be more accurate than the labels. Thus, a model with a very high performance might be good at imitating the labels, whilst not that good at measuring the true summary quality. This uncertainty is a great challenge that follows from not knowing the ground truth and can only be overcome by letting experts use a large amount of time to create a gold standard test set.

5.2 Model Performance Evaluation

In this section, the results of the various architectures, as proposed in Section 4.3, will be given. These will be evaluated on the weak supervision labels. It should therefore be noted that the evaluation is not based on the ground truth, and it is therefore uncertain how accurate these results really are. The models are trained on the training set, while the validation set is used to determine hyperparameters and other configurations. Finally, the results of this section are given on the held-out test set, such that the models have never

Table 5.2: Model performances on the test set, measured by the noise-aware cosine embedding loss, as defined in (4.5), and classification scores, as defined in (4.9). In the loss function, $\tau_{\text{good}} = 0.2$ and $\tau_{\text{bad}} = -0.2$ are used. Note that all models yield a continuous quality measure; classification scores are only included to give an intuitive measure of performance.

Model	Loss	Accuracy	Precision	Recall	$\mathbf{F}_1$-score
LSA	-	0.726	0.727	0.786	0.755
Doc2vec	-	0.684	0.737	0.641	0.686
LSA+LinTrans	0.095	0.863	0.849	0.906	0.876
Doc2vec+LinTrans	0.101	0.850	0.845	0.882	0.863
LSA+FFN	0.080	0.882	0.871	0.916	0.893
Doc2vec+FFN	**0.079**	0.885	0.868	0.928	0.897
LSA+LSTM	**0.079**	0.882	0.863	0.929	0.895
Doc2vec+LSTM	0.080	0.880	0.869	0.914	0.891
EmbLayer+CNN	0.088	0.888	**0.881**	0.915	0.898
Word2vec+CNN	0.085	**0.895**	0.878	**0.934**	**0.905**

previously seen the documents that are in the following evaluation.

The performances on the test set are given in Table 5.2. In this table, the unsupervised baseline models are at the top, while the supervised architectures trained on weak supervision labels are below. The performances are measured by the loss function in (4.5), in addition to the classification scores in (4.9). For the loss function, $\tau_{\text{good}} = 0.2$ and $\tau_{\text{bad}} = -0.2$ are used.

The table shows that the FFN and LSTM-based models are best in terms of loss, while the CNN-based models are best with respect to the classification scores. This indicates that the CNN models make fewer, but bigger mistakes than the FFN and LSTM models. The differences in performance are, however, small, and we cannot really determine which model is better based on the performance scores alone. What we can say for certain is that the supervised architectures substantially outperform the unsupervised baseline models LSA and Doc2vec.

To visualize the performance of the models, the distribution of quality measures can also be shown. In particular, by showing one distribution for good summaries, and one distribution for bad summaries, it will be possible to see how the various models evaluate summaries of different quality. For this visualization, the selection of good summaries is chosen as the reports where $y^+ = P_{\boldsymbol{\mu}}(y = 1|\boldsymbol{\Lambda}) \geq 0.9$, while the selection of bad summaries is chosen as the reports where $y^+ = P_{\boldsymbol{\mu}}(y = 1|\boldsymbol{\Lambda}) \leq 0.1$. The values 0.1 and 0.9 are chosen such that we visualize the distributions only for samples where the weak supervision label model is confident about the label.

The results are shown in Figure 5.2. This figure shows that all models are, to some degree, able to distinguish good summaries from bad ones. The LSA and Doc2vec baselines do, however, have much more overlap between the distributions than the other models, which reflects the poorer performance in Table 5.2.

The distributions for the FFN and LSTM-based models are unexpected. These models push the quality measures just below -0.2 or just above 0.2, instead of distributing them on the complete quality range $[-1, 1]$. This behaviour effectively makes the FFN and LSTM

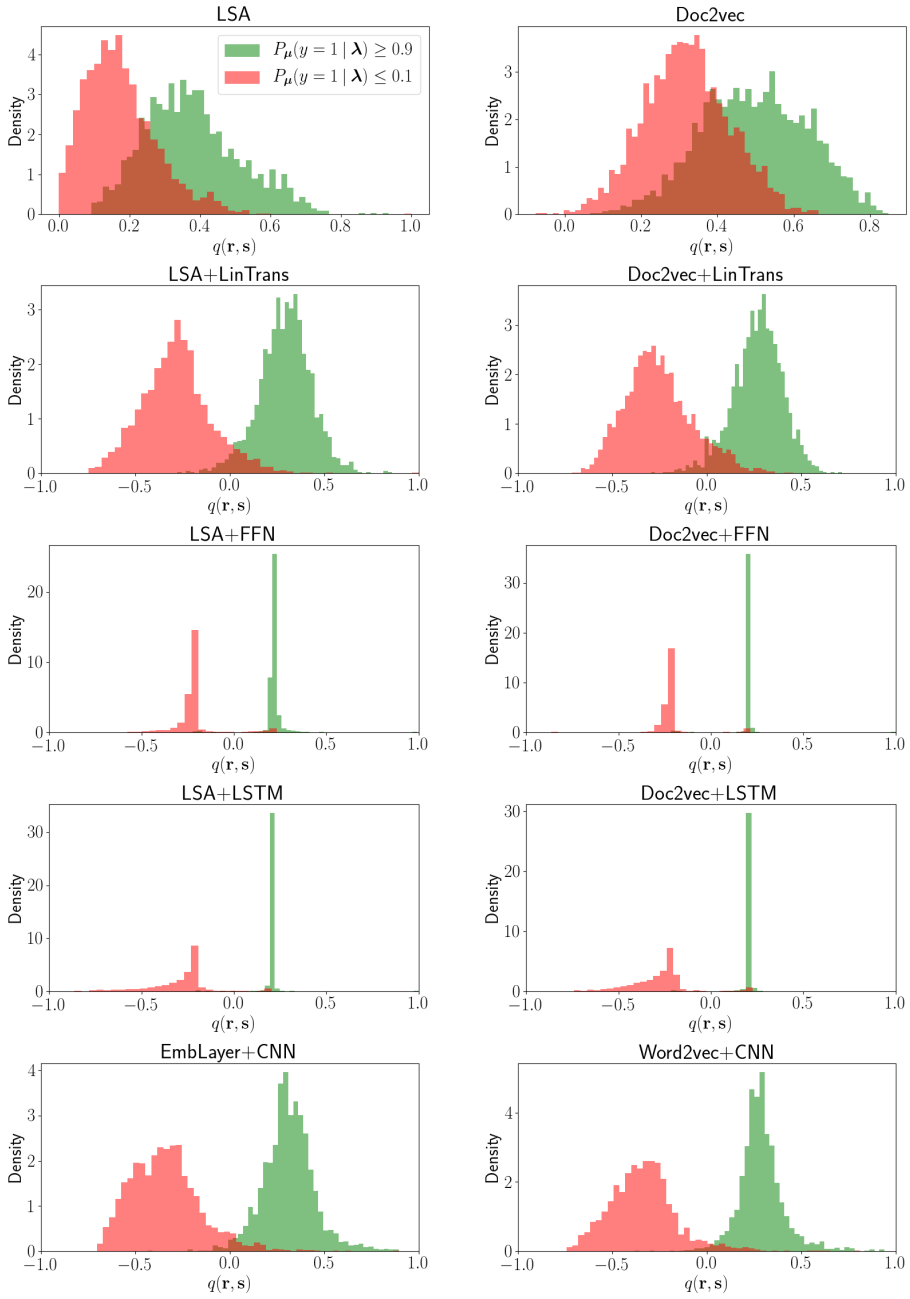

Figure 5.2: Normalized histograms showing the distribution of quality measures $q(\mathbf{r}, \mathbf{s})$ for the models on the test set of the real estate condition report data. Reports where $p(y = 1|\boldsymbol{\Lambda}) \geq 0.9$ are shown as green, while reports where $p(y = -1|\boldsymbol{\Lambda}) \geq 0.9$ are shown as red.

models classifiers, rather than quality measuring models, which is not the behaviour we want. It is interesting to see that the LinTrans models, which are simple forms of FFNs with only a single, linear layer, do not exhibit the same behaviour. Thus, it seems like this behaviour might occur when we add more complexity to the models.

5.3 Discussion

The results of the various model architectures, when evaluated on the weak supervision labels, have now been presented. In light of these results, the model performances will now be discussed. Furthermore, a discussion concerning the choice of loss function will follow, and finally, the choice of hyperparameters for the various models will be discussed.

5.3.1 Model Performance Discussion

The performances of the model architectures of this work were presented in Table 5.2 and Figure 5.2. With these results in mind, the behaviours of the various architectures will now be discussed.

Baselines

From Table 5.2, we see that the baseline models have a higher loss and lower classification scores compared to the other, supervised architectures. This is natural since the baseline models have been constructed without knowing anything about what a good summary is. These poor results indicate that in the real estate context, general semantic similarity is not a very good measure of summary quality.

The weaker performance of the unsupervised architectures is also seen in Figure 5.2. It is clear that the distributions for good and bad summaries are different, which is good. However, there is a lot of overlap between the distributions, which implies that the models are not properly able to distinguish good summaries from bad ones.

The main difference between the baselines and the other models is the fact that the baselines are completely unsupervised or self-supervised, while the other models are supervised using labels attained by weak supervision. The poorer performance of the baselines mainly reflects this very basic difference. The fact that the supervised architectures obtain higher performance scores shows that there is a lot of potential in the weak supervision approach. We cannot measure the true performance of the models, since the ground truth is unknown. However, the results indicate that the use of supervised methods, through weak supervision, can increase performance substantially.

LinTrans and FFN

From Table 5.2, the LinTrans and FFN architectures appear to be very effective, as they substantially increase performance over the baselines LSA and Doc2vec. The LinTrans and FFN models are, however, based on the same amount of information, since they use semantic feature vectors from LSA or Doc2vec as input. This indicates that LSA and Doc2vec actually contain the necessary information to measure summary quality very

well. They simply require a transformation to filter out the important information concerning summary quality. And as the results show, both linear and non-linear transformations, implemented through FFNs, can do this effectively.

However, when looking at the distribution of qualities that the FFN models yield in Figure 5.2, we clearly see that the non-linear FFN models do not behave as we want. In fact, they illustrate very well a problem with the performance-driven approach of supervised learning: The models often learn to solve their training objective very well, but they might not do it the way we intend them to.

We would like the models to identify good patterns that increase summary quality, and bad patterns that decrease summary quality, such that the final quality measure reflects both good and bad patterns. In such a model, a summary with many good patterns would get a higher score than a summary with only a few patterns, even though they both are good. However, the non-linear FFN models instead seem to find patterns that identify *whether* a summary is good or bad, and simply yield a quality measure just above 0.2 for good summaries and just below -0.2 for bad summaries, so that a loss of 0 is obtained. In such a model, a summary with many good patterns would get the same score as a summary with only a few good patterns. This effectively makes the non-linear FFN models classify samples, instead of measuring summary quality properly.

This unwanted behaviour only happens in the more complex FFN models, and we see that the LinTrans models do not suffer from this problem. Instead, they yield distributions with the properties we expect and require for a good summary quality model. The LinTrans architecture is, in fact, rather simple, and it is interesting to see that this model can perform almost as good as the more complex neural networks with respect to the performance scores, whilst exhibiting much better properties with respect to actually measuring summary quality.

LSTM

The LSTM models obtain a very high performance, both in terms of loss and classification scores. However, we see that the LSTM models are not really better than the FFN models. This fact indicates that there is no particular gain in dividing the documents into sections, such that recurrent mechanisms can be used.

Furthermore, when looking at the distribution of summary quality that the LSTM models yield, we see that they have the same classifying behaviour as the FFN models do. This indicates that the LSTM architecture is not particularly well suited for the task of measuring summary quality.

A part of the motivation behind the LSTM approach is the fact that many state-of-the-art embedding techniques are not applicable to very long documents. By making an approach based on shorter documents, which the LSTM architecture is, it would be possible to use better semantic embedding techniques, like for example BERT (Devlin et al. 2018). We do, however, not investigate this idea further, since the LSTM architecture is not particularly well suited for the task at hand.

CNN

The CNN models obtain the best classification scores, in addition to having almost as good loss as FFN and LSTM. Furthermore, the distribution of summary quality looks very good, which implies that the CNN architecture is well suited for measuring summary quality. These results indicate that the CNN approach is a good strategy when working with very long documents. The EmbLayer+CNN model is also the only model in this work that does not make use of existing semantic embedding techniques, and it is interesting to see that we can obtain such good results without relying on existing techniques within the field of document similarity.

We see from Table 5.2 that the Word2vec+CNN model actually performs slightly better than the EmbLayer+CNN model. When training our own word embeddings (EmbLayer+CNN), the model should get higher flexibility, and thus, a higher tendency to overfit is expected. We observe that the training losses for the two models are very similar (see Appendix A.5), while the validation loss and test loss is better for Word2vec+CNN. This indicates that the Word2vec word embeddings generalize better to new, unseen data.

Another advantage of the Word2vec+CNN model is the fact that the Word2vec word embeddings are trained independently of the weak supervision labels. These labels are noisy and possibly imprecise. Therefore, it might be preferable to learn as many model weights as possible without using these labels. We then reduce the models' ability to imitate the weak supervision labels, which we hope can make the models better at generalizing beyond the weak supervision labels.

5.3.2 Loss Function Discussion

When training supervised architectures, the choice of loss function plays an important role for the resulting supervised models. This is because they are performance-driven, and only try to maximize their performance on the given loss. We therefore have to be mindful when choosing which loss function to use.

In this work, we use the noise-aware cosine embedding loss, as defined in (4.5). This is an intuitive loss that reflects the properties we want the summary quality models to have. For this loss function, we have to decide what τ_{good} and τ_{bad} should be. In doing so, we define which score a good or bad summary at least should have. However, when we started to train models, it quickly became apparent to us that by defining these values, we could also influence the shape of the quality distribution for the various models a lot.

This is illustrated in Figure 5.3, which shows the distribution of good and bad summaries on the test set for different values of τ_{good} and τ_{bad}, when using the LSA+LinTrans model. The figure also includes the distribution of all summaries in the test set, which illustrates the shape of the general summary quality distribution. The figure shows that the grey distribution has a shape that depends a lot on the choice of τ_{good} and τ_{bad}. In particular, we see that the model try to push bad summaries below τ_{bad} and good summaries above τ_{good}. This is, of course, the expected and requested behaviour, since it minimizes the loss function. Nevertheless, we observe that before we can determine what τ_{good} and τ_{bad} should be, we have to take into consideration what the general summary quality distribution should look like.

In the real world, many phenomena are normally distributed. Since we do not have

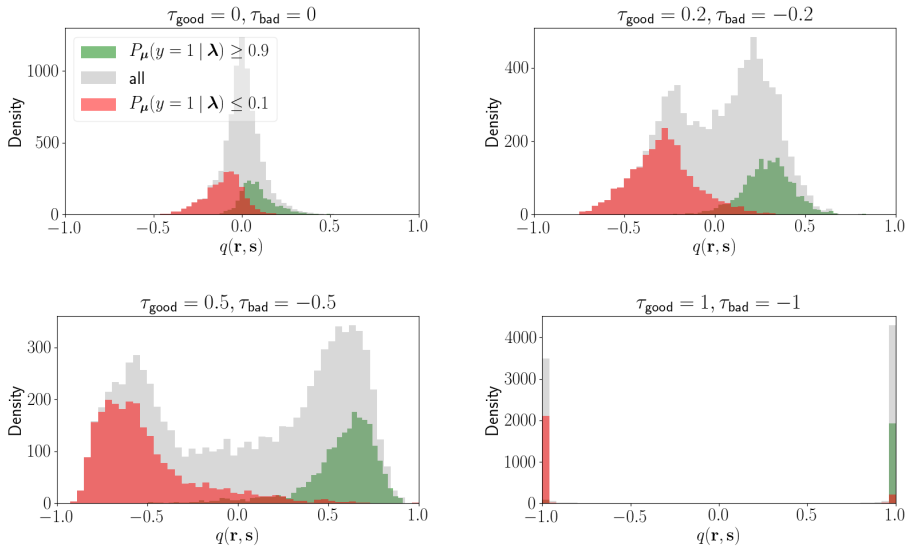

Figure 5.3: Normalized histograms showing the distribution of quality measures $q(\mathbf{r}, \mathbf{s})$ for the the LSA+FFN model on the test set of the real estate condition report data. Reports where $P_{\boldsymbol{\mu}}(y = 1|\boldsymbol{\lambda}) \geq 0.9$ is shown as green, while reports where $P_{\boldsymbol{\mu}}(y = 1|\boldsymbol{\lambda}) \leq 0.1$ is shown as red. The grey distribution shows all reports.

any prior knowledge concerning summary quality, it makes sense to assume that also this should resemble a normal distribution. This motivates us to choose a value of τ_{good} and τ_{bad} that results in a quality distribution similar to that of a normal distribution. From Figure 5.3, we clearly see that this is not achieved by choosing for example $\tau_{\text{good}} = 0.5$ and $\tau_{\text{bad}} = -0.5$.

In this work, we choose to use $\tau_{\text{good}} = 0.2$ and $\tau_{\text{bad}} = -0.2$. This choice is a compromise between two properties: On one hand, we want the quality distribution to resemble a normal distribution, while on the other hand, we want the models to properly distinguish good summaries from bad ones. With $\tau_{\text{good}} = 0.2$ and $\tau_{\text{bad}} = -0.2$, there are still peaks around -0.2 and 0.2, but apart from this, we get a quality measure with many of the expected and requested properties.

5.3.3 Hyperparameter Discussion

The various model architectures of this work all have hyperparameters that influence their performance. In this section, the choice of hyperparameters and the resulting model complexities will be discussed.

In this work, the best hyperparameters are found by thoroughly testing various configurations. The main goal of this testing is to minimize the loss function, while a secondary goal is to maximize the classification scores. This testing is done on a validation set, such that the optimal hyperparameters are found without ever seeing the reports in the test set.

In this section we will only state which configurations we found to be best for each model, and give a short discussion. The complete result of the hyperparameter tuning, with more details, can be found in Appendix A.

Baseline Hyperparameters

A very important element to consider when building semantic embedding techniques is the choice of underlying vocabulary. In particular, it is common to omit words that are very common, or very rare, since very common words probably are structural words that do not contribute to the semantics of the documents, while very rare words probably are typos, names or made-up words that also are unlikely to be helpful to the embedding.

For LSA, the vocabulary is controlled by the vocabulary in the underlying bag-of-words model. For TF-IDF, the implementation in `gensim` has a hyperparameter `nb` (no below), that controls the minimum number of documents a word has to appear in for it to be included in the vocabulary. Furthermore, so-called *stop-words*, which are structural words that generally do not contribute to the semantics, are often removed. We denote by `rs` (remove stopwords) the boolean indicator of whether stop-words are removed or not. In this work, we get the best results for $\texttt{nb} = 15\,000$ and $\texttt{rs} =$ False.

These values are rather surprising. First, $\texttt{nb} = 15\,000$ results in a vocabulary of only 513 words. Thus, it seems like it is more effective to reduce the dimensionality in LSA by reducing the vocabulary, instead of keeping only the K most significant topics. We also try to use TF-IDF alone, and we find that the results are similar. We therefore keep using LSA, but note that TF-IDF seems to be equally good in the real estate domain. We also find, to our surprise, that the results are slightly better when stop-words are not removed. Thus, we use $\texttt{nb} = 15\,000$ and $\texttt{rs} =$ False in the LSA baseline model.

For Word2vec and Doc2vec, the `gensim`-implementation controls the vocabulary differently. There is a hyperparameter `mc` (minimum count) that controls the minimum number of times a word must appear in the corpus for it to be included in the vocabulary. We find that $\texttt{mc} = 20$ gives good results, and is therefore used. Furthermore, stop-words are kept since Word2vec and Doc2vec look at word windows in the training process. The structural stop-words then have a purpose since they contribute to the meaning of the surrounding words.

Word2vec and Doc2vec also have two more hyperparameters, namely the window size R, and the number of *epochs*, that is, the number of passes over the data in the training process, which we denote by `e`. We find that good results are obtained with $R = 6$, and $\texttt{e} = 50$ appears to be sufficient for convergence.

Finally, the dimensionality K of the embedding techniques must be set. Firstly, we find that $K = 500$ is best for LSA. Note that for this value of K, barely any dimensions are removed from the underlying TF-IDF model since we use a vocabulary of only 513 words. Thus, the resulting TF-IDF and LSA models are more or less based on the same information, and it is natural that we observe a very similar performance. For Doc2vec, we obtain the best results with $K = 100$, which is therefore used in the final Doc2vec baseline model.

LinTrans and FFN Hyperparameters

There are several hyperparameters that must be tuned in the FFN model. First of all, there are some hyperparameters that are common for artificial neural networks in general, such as dropout `d`, batch size B, and learning rate `lr`. Dropout is a regularization technique in which some of the network nodes are set equal to zero in the training process. This adds noise into the calculations, which reduces overfitting. A fraction of nodes are chosen at random for each training batch, where the dropout parameter `d` decides how large that fraction is. We get the best results with $\texttt{d} = 0.2$ in the FFN models. Note that dropout is only employed in hidden layers, and thus, this effect is not present in the LinTrans models. Furthermore, `lr` is a hyperparameter that controls the step length in the Adam optimization algorithm. For the LinTrans and FFN models, we get the best results with $\texttt{lr} = 10^{-3}$. Note that the learning rate is reduced by a factor of 0.1 after one third of the epochs, and again after two thirds. The model then takes shorter steps towards the end of the training process, which gives better performance. Thus, the hyperparameter `lr` only determines the initial learning rate. For the batch size, $B = 64$ gives good results and is therefore used.

We also try to vary the dimensionality K of the embedder models LSA and Doc2vec, to see if the FFN model becomes better with a higher input dimensionality. We find that the best results are obtained with $K = 500$ for both LSA and Doc2vec. For LSA, we also try a few different values of `nb`, since the surprisingly high value of $\texttt{nb} = 15\,000$ was found to be best in the baseline LSA model. For the LinTrans and FFN models, we find that the value of `nb` does not influence the performance very much, and we keep using $\texttt{nb} = 15\,000$ for simplicity. For the other hyperparameters in the embedder models, we use the same values as in the baselines.

Finally, the number of layers, as well as the number of nodes in each layer, must be determined. By increasing the number of nodes and layers in the models, we add complexity. We find that the number of output neurons does not influence the results very much, and for the LinTrans models, we use 100 neurons in the output layer for both the LinTrans and FFN models. For LSA, the best results with non-linear FFNs are obtained with two hidden layers of 1000 neurons each, while the best results for Doc2vec are obtained with three hidden layers of 1000 neurons each.

LSTM Hyperparameters

The LSTM network also has the common hyperparameters dropout, batch size and learning rate. We get the best results for LSTM with dropout $\texttt{d} = 0$, batch size $B = 64$ and $\texttt{lr} = 10^{-1}$. Note that for $\texttt{d} = 0$, dropout is not used at all. Thus, it seems like the LSTM model does not benefit from this kind of regularization.

We also try to vary the dimensionality K of the input embedders, as well as `nb` for LSA. Again, we find that the best results are obtained with $K = 500$ and $\texttt{nb} = 15\,000$.

Finally, the dimensionality of the LSTM cells, the number of LSTM layers and whether they are bi-directional or not must be determined. In our hyperparamet testing, we find that the performance on the validation set is not improved by increasing the complexity of the LSTM models. Thus, we use an LSTM cell dimensionality of 100, and we use only a single, uni-directional LSTM layer. Finally, we get good results with 100 nodes in the

output layer, and we therefore use this in the final model.

CNN

The CNN network also has the hyperparameters dropout, batch size and learning rate. We get good results with $\texttt{d} = 0.1$, $B = 64$ and $\texttt{lr} = 10^{-3}$ for EmbLayer+CNN and $\texttt{lr} = 10^{-2}$ for Word2vec+CNN.

We also try to vary both the dimensionality of the word embeddings K, as well as the size of the vocabulary V. In the EmbLayer+CNN model, the vocabulary size is controlled directly by only keeping the V most common words, while in Word2vec, this is done indirectly by tuning the minimum count parameter `mc`, which is the same for Word2vec as for Doc2vec. We find that EmbLayer+CNN is best for $K = 500$ while Word2vec is better for $K = 100$. We also find that the vocabulary size does not influence the performance much, and we choose to use $V = 20\,000$ for EmbLayer+CNN. For Word2vec+CNN, we use `mc`=20, which results in a vocabulary size of $V = 26\,533$. We also tune the window size R in the Word2vec model and find good results with $R = 10$.

Finally, the number of filters must be determined. We do this by first choosing which filter heights we want to include, and then we use N_F filters for each filter height. We get good results for EmbLayer+CNN with only a single filter height of 5 with $N_\text{F} = 500$ filters. For Word2vec+CNN, we get better results with filter heights of 2, 3, 5, 7 and 10, with $N_\text{F} = 200$ filters of each. Note that in the CNN models, we also use N_F neurons in the final, linear output layer.

5.4 General Analysis of Summary Quality

Now that various quality-measuring models have been evaluated and discussed, we finally have the tools necessary to analyse the general summary quality in the real estate condition report dataset. This will be done in this section. In particular, the distribution of summary quality for the various models, on all real estate condition reports (including the reports that the weak supervision model abstained from labelling) will be analysed. Furthermore, a few examples of summaries, with corresponding quality measures, will be investigated.

This section is meant to give an overview of the general summary quality in the real estate condition report dataset. The goal of this analysis is to shed some light on the high conflict rate in real estate transactions, as discussed in Section 1.1. Before proceeding, we emphasize again that the quality measuring models of this work have only been evaluated on weak supervision labels. They have never been tested on reports where the ground truth summary quality is known. Even if the results of this work indicate that the models have a good performance, we cannot know for certain. Hence, the results of this section are not the ground truth, but instead an indication of summary quality.

5.4.1 Distribution of Summary Quality

The distribution of summary quality for all $M = 96\,534$ real estate condition reports, when using the LSA+LinTrans, Doc2vec+LinTrans, EmbLayer+CNN and Word2vec+CNN models, are shown in Figure 5.4. We do not include results from non-linear FFN

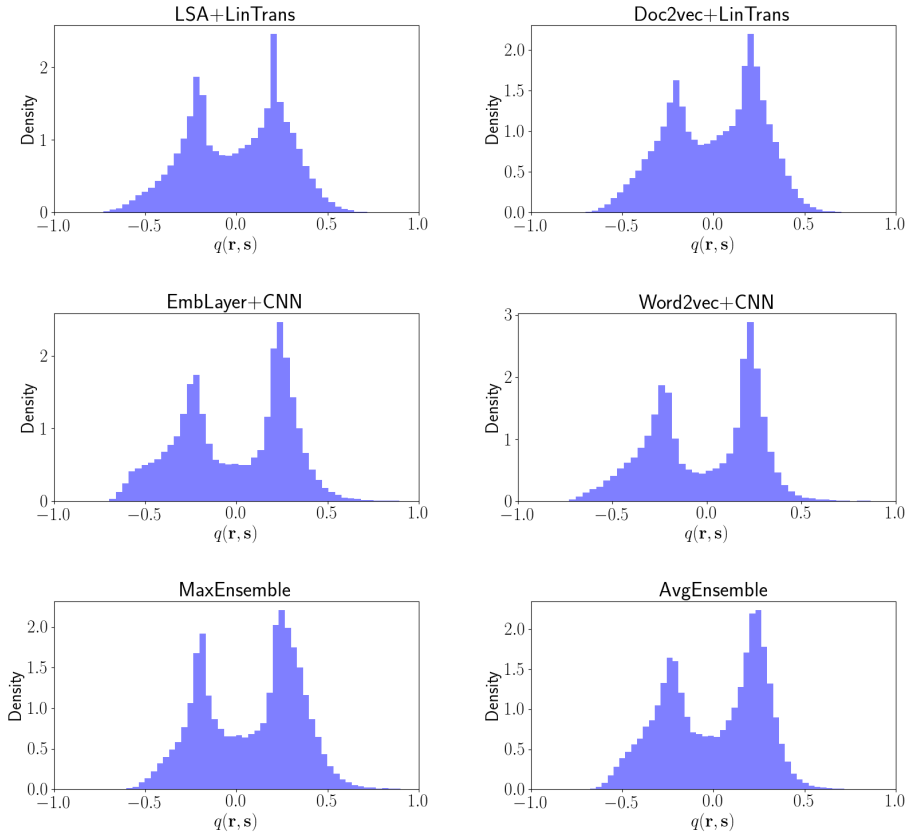

Figure 5.4: Normalized histograms showing the distribution of quality measures $q(\mathbf{r}, \mathbf{s})$ for FFN and CNN on all $M = 96\,534$ real estate condition reports. The distribution when taking the maximum and average of the quality measure from LSA+FFN, Doc2vec+FFN, EmbLayer+CNN and Word2vec+CNN, that is, MaxEnsemble and AvgEnsemble, are also included.

and LSTM models in this section, since we found them not to be particularly well suited for measuring summary quality, as discussed in Section 5.3.1. Note that Figure 5.4 also includes the real estate condition reports that the weak supervision label model abstained from labelling. We also include two ensemble models: MaxEnsemble, in which we take the maximum quality score of the LinTrans and CNN models, as well as AvgEnsemble, where we take the average of them.

We see from Figure 5.4 that the models seem to agree on the overall summary quality. We also observe that the ensemble models have a very similar shape to the other models. All models have peaks around -0.2 and 0.2, which are caused by the choice of $\tau_{\text{good}} = 0.2$ and $\tau_{\text{bad}} = -0.2$, as discussed in Section 5.3.2. The peaks seem to be slightly more distinct in the CNN models, which reflects the slightly better performance of the CNN models in terms of loss, in Table 5.2.

Furthermore, the percentages of reports that the models consider bad, mediocre and good are shown in Table 5.3. Note that by defining τ_{good} and τ_{bad}, we also define at least how good or bad the quality should be for a good or bad summary. We therefore use the thresholds $\tau_{\text{bad}} = -0.2$ and $\tau_{\text{good}} = 0.2$ to determine the limits between bad, mediocre and good summaries.

From Table 5.3, we first observe that all models consider there to be fewer bad summaries than the weak supervision label model estimate. According to the label model, 39.6% of the summaries are bad, which is a very high number. It seems like all of the various quality models think this number is too high, and give fewer summaries a score below $\tau_{\text{bad}} = -0.2$. The CNN models estimate more summaries as bad compared to the LinTrans models. This reflects the higher performance that the CNN models have over the LinTrans models: According to the labels, more summaries are bad, and since CNN fits better to the labels, the CNN models also consider more summaries to be bad. Finally, it is interesting to observe that as many as 20% of the summaries are considered bad by the MaxEnsemble model. This means that in 20% of the cases, all models agree that the summary is bad. The MaxEnsemble model thereby gives an even stronger indication of a bad summary than the other models do.

Table 5.3: Percentages of condition report summaries that the weak supervision label model, as well as the LinTrans and CNN architectures consider bad ($q \leq -0.2$), mediocre ($-0.2 < q < 0.2$) and good ($q \geq 0.2$). For the label model, good summaries are determined by $P_{\boldsymbol{\mu}}(y = 1 \mid \boldsymbol{\lambda}) \geq 0.5$, while bad summaries have $P_{\boldsymbol{\mu}}(y = 1 \mid \boldsymbol{\lambda}) < 0.5$. The percentages when taking the maximum and average of the quality measures from LSA+LinTrans, Doc2vec+LinTrans, EmbLayer+CNN and Word2vec+CNN are also included.

Model	Bad	Mediocre	Good	Abstain
Label model	39.6%	-	44.5%	15.9%
LSA+LinTrans	28.1%	42.3%	29.6%	-
Doc2vec+LinTrans	26.1%	45.0%	29.0%	-
EmbLayer+CNN	34.8%	28.8%	36.4%	-
Word2vec+CNN	35.1%	31.0%	33.9%	-
MaxEnsemble	20.0%	35.2%	44.8%	-
AvgEnsemble	30.7%	37.8%	31.6%	-

The correlation between the models' quality measures is shown in Table 5.4. The table shows that the correlation, in general, is high. Thus, we observe that the various models not only have a similar shape, but they also typically agree on the summary quality for samples. If the models truly can measure summary quality, then they must also have a high correlation. It is therefore a good result that we observe a high correlation between models.

Another interesting observation is that the various models seem to have a slightly lower correlation to the label model than to the other models. For example, the LinTrans models have a higher correlation to the CNN models than to the label model. The CNN models, on the other hand, have a higher correlation to the label model than to the LinTrans model, but an even higher correlation to each other. This is a very interesting result since this is the expected behaviour if the supervised models actually are able to generalize beyond

the weak supervision labels. We should, however, note that the supervised models are continuous quality measures on the domain $[-1, 1]$, while the labels are probabilities on the range $[0, 1]$. The lower correlation to the label model can also be explained by the fact that the probability has a different meaning than the quality measures, and thus, the qualities might be less comparable to the labels than to each other.

We have now seen how the various models evaluate the summary quality of the real estate condition reports. All models indicate that there are quite a few summaries that are bad. In particular, the results give us reason to believe that more or less 30% of summaries are not very good. Since the real estate condition reports are so long and technical, the summaries are very important to give an overview of the condition of the real estate. If so many summaries are bad in general, it is clear that the summary quality is a potential source of conflict.

Table 5.4: The correlation between the quality measures of the label model, LinTrans models and CNN models. Here, LT stands for LinTrans, D2v stands for Doc2vec, EL stands for EmbLayer and W2v stands for Word2vec.

	Lab.mod.	**LSA+LT**	**D2v+LT**	**EL+CNN**	**W2v+CNN**
Lab.mod	1.00	0.87	0.86	0.94	0.93
LSA+LT	0.87	1.00	0.91	0.91	0.90
D2v+LT	0.86	0.91	1.00	0.90	0.89
EL+CNN	0.94	0.91	0.90	1.00	0.97
W2v+CNN	0.93	0.90	0.89	0.97	1.00

5.4.2 Summary Examples

In order to get a better impression of how the various models actually work, we will now give a few examples of summary texts, and then investigate how the LinTrans and CNN models evaluate them. Consider the following five summaries. (English versions of these, translated by Google Translate, are given in Appendix B.)

1. **ID**: 5d80ea54-0ace-418b-95c7-2cad2dc93ec5
   Summary: Enebolig fra 1978 som er holdt vedlike og har god standard, tatt alder i betraktning. Den er noe påkostet over tid ellers er det originalt. Det er valmtak med bordtak. Renner og nedløp. Bindingsverkvegger som er isolert med stående panel og murforblending. Vinduer med karm og ramme i tre med isolerglass. Massiv utgangsdør i teak. Det er leca grunnmur og støpt dekke. Dreneringen er fra byggetiden. Innvendig er det panel og plater i himling, gulv har fliser, belegg, laminat, tepper og parkett. Baderom med fliser på gulv og vegger med sanitær utstyr som er fra byggetiden. Det er eget wc rom og dusjkabinett i fyr-rom og wc med servant i vaskerom. Eik kjøkkeninnredning med profiler på overskap og underskap fra byggetiden. Sentralfyr for olje og strøm som er ca 10 år. Oljetank under terrasse. Elektrisk anlegg med skrusikringer. Garasje fra 1986 den er oppført med støpt dekke, leca ringmur, stående kledning. Valmtak med betongstein, renner og nedløp i plastbelagt stål. Det er 2 stk leddporter. Det er registrert vanlig elde og bruksslitasje på eiendommen.

2. **ID**: e527b688-a1ec-4a7b-a04b-097868b1124a
   Summary: Boligen ligger i et etablert boligområde, med kort vei til skole, barnehage og forretning. Det er gjort bemerkninger som bør utbedres, som våtrom og oppgraderinger pga. normal bruksslitasje. Forøvrig les rapport.

3. **ID**: 84374a61-c152-49a4-bc0f-6dae06158ca7
   Summary: Enebolig med normal standard beliggende i Sveggen i Averøy kommune. Generelt oppført i gode og kjente konstruksjoner med den byggemåte som var vanlig ved oppføringstidspunktet. Av avvik kan nevnes: Det var indikasjon på fukt i gulv og yttervegger i dusjsonen på badet, grunnen kan være at dusjen nettopp var brukt. Det var ikke mulig å kontrollere under gulvet. Det kan være nødvendig med nærmere undersøkelser. Det er kun en sluk i gulvet og den er i dusjsonen. Dusjsonen er bygd opp av 2 vegger og en kant ned mot gulvet. Denne kanten på 7 cm som er tett gjør at eventuelt lekasjevann vil renne ut over dørstokken og til tilstøtende rom.

4. **ID**: d3479149-98e0-41af-9f68-26479c687a8d
   Summary: Enebolig med krypkjeller, 1. etasje og loft som er bygget i ca 1903. Boligen er oppført i grunnmur i tegl og stein, trekonstruksjon som utvendig er kledd med trepaneler, trebjelkelag. Saltak i trekonstruksjon tekket med skifertakstein. Boligen har et normalt vedlikehold men det er behov for noe oppgraderinger. Pipen er i dårlig stand utvendig og her er det behov for oppgraderinger, det er fuktig i krypkjelleren og det anbefales å montere kryperomsavfukter. Bygningen er i god stand med tanke på bygningens alder. Registrerte tilstandanmerkninger har hovedsakelig årsak i bygningens alder og vedlikehold samt konstruksjon.

5. **ID**: 50bdfbaf-7642-4a17-8358-070257214b98
   Summary: Boligen trolig med alt som fra byggeår og derfor med noe naturlig slitasje. Av vesentlig betydning angående bemerkninger ble det registrert noe fukt i dusjvegg i 2. etg grunnet utettheter avløp vaskemaskin. Alt vedlikehold utenfor seksjonen skal normalt være et felles ansvar i sameiet.

The first summary is very detailed and is expected to give a high quality score. The second summary is very short, and although it mentions that upgrades must be done, the reader of the summary will have no information regarding the size and type of upgrade that is required. The second summary is therefore expected to get a low score. The third summary talks a lot about the bathroom. We have also read the corresponding report, and we note that this is very short, and mainly talks about the bathroom as well. Thus, it is possible that the summary is not too bad, but we note that both the report and summary give a poor impression of the real estate in general. The fourth summary appears to be good, as it gives an overall impression with a few notes of recommended improvements. When reading the full corresponding report, we do, however, find that there are several points with condition degree TG2 or TG3 that the summary fails to mention. The most important parts seem to be in the summary, but we note that a very good summary should probably include a few more parts. Therefore, this summary is mediocre in our eyes. Finally, the fifth summary is very short and does not seem to be very good. When comparing with the corresponding

report, which is long and detailed, it is clear that the summary should contain much more information.

The predicted quality measures for the above summaries, when using the weak supervision label model, the LinTrans models and the CNN models are given in Table 5.5. We see that all models agree that the first summary is good, and the second summary is very bad. The third summary is evaluated as good according to the label model and LinTrans models. The CNN models, on the other hand, evaluate the third summary as mediocre. Thus, the CNN models find something in the third report and summary that lowers the quality. This can perhaps be the fact that the report and summary both give a poor impression of the real estate, but we cannot know for sure due to the black-box nature of neural network models.

Furthermore, the fourth summary is bad according to the label model. This model uses meta information about condition degrees, and it seems like the label model finds points that are lacking in the summary, that should be there according to the labelling functions. The other models do, however, evaluate the fourth summary as decent, though not very good. This is in accordance with our comments above, where we argue that the fourth summary is mediocre. It seems like the models are able to pick up some, but not that many good patterns for this summary. Finally, the fifth summary is evaluated as mediocre according to the Doc2vec+LinTrans and Emblayer+CNN models, and bad according to the rest. This summary is also bad in our eyes, but the summary does mention a few points of interest, which might be what the models pick up when they evaluate the summary as mediocre.

Table 5.5: A collection of results when using FFN and CNN to measure summary quality. The summaries that the models are applied to are given above.

Model	**Sum. 1**	**Sum. 2**	**Sum. 3**	**Sum. 4**	**Sum. 5**
$P_{\boldsymbol{\mu}}(y = 1 \mid \boldsymbol{\lambda})$	0.92	0	0.97	0.09	0
LSA+LinTrans	0.24	-0.68	0.54	0.15	-0.29
Doc2vec+LinTrans	0.46	-0.54	0.43	0.07	-0.10
EmbLayer+CNN	0.67	-0.62	0.13	0.14	-0.12
Word2vec+CNN	0.23	-0.68	-0.11	0.22	-0.26

We have now looked at the behaviour of the models on a few examples. It is not possible to draw any clear conclusions based on such a small set of examples, and we therefore note, once again, that a proper conclusion about the model performances can only be obtained by creating a high-quality, hand-labelled test set to evaluate the models on. Nevertheless, the above analysis gives us an impression. The results on these examples are reasonable, and even though the models differ in some cases, it seems like they work more or less as intended. Based on this, as well as on the other results in this chapter, we believe that the above models can give a good indication of summary quality for real estate condition reports.

Chapter 6

Conclusion

The objective of this work has been to measure summary quality for real estate condition reports. This task has been challenging and interesting, especially since there is little previous work on this particular topic. Therefore, there have not been any clear guidelines of how we should proceed to solve the problem at hand. We chose to find inspiration in the field of document similarity, and have therefore explored an approach where we create document embeddings that are appropriate for measuring summary quality. In other words, we map reports and summaries to a conceptual summary content space, where summary quality can be measured by the cosine similarity between the embedded report and summary.

We decided that we wanted to use supervised learning methods to create this mapping. We therefore acquired labels through the weak supervision system Snorkel. It is hard to evaluate exactly how effective this approach has been, since we do not have any samples where the ground truth summary quality is known. However, all results have indicated that the approach is effective, and that we indeed are able to measure summary quality, at least to some extent.

In this chapter, we will first attempt to draw conclusions about the various models' ability to measure summary quality. Then, we will summarize our results of the overall summary quality for the real estate condition report dataset, and attempt to conclude whether or not the high conflict level in real estate transactions can be explained by poor summary quality. Finally, we will discuss possible directions of future work that can improve the quality of our models, and more importantly, find better ways to evaluate how well the models actually work.

6.1 Model Assessment

We first created a label model $P_{\boldsymbol{\mu}}(y \mid \boldsymbol{\lambda})$, by using the weak supervision system Snorkel. From this, we acquired a labelled dataset of $M_{\text{lab}} = 81\,195$ samples. It is difficult to evaluate how precise the label model actually is since the ground truth is unknown for all data samples. The label model is, however, based on rules that are constructed specifically for being related to summary quality. Furthermore, previous results when using weak

supervision have shown that this often is an effective approach, and that it can be more time-efficient than creating hand-made labels. We therefore have reason to believe that the weak supervision label model truly can measure summary quality, at least to some degree.

Yet, we also expect the labels from the label model to be noisy, and we cannot know for certain if there is bias in the labels. We must therefore be careful when drawing conclusions based on the weak supervision labels, even though we have reason to believe that the labels are adequate. In this section, we will draw conclusions about the various models' ability to measure summary quality, but since these conclusions are based on the weak supervision labels, we must remember that they are only indications and not the ground truth.

The LinTrans models achieve very good results in terms of loss, accuracy scores and density plots. The LinTrans architecture is the simplest form of FFN with only a single, linear layer, and there are several good properties with this model. It is heavily based on reliable document embedding techniques that are trained without using the weak supervision labels. The LinTrans architecture is also simple since it only consists of a linear transformation of the full semantic vector space. The good performance, simplicity and interpretability of this model make it a desirable choice. We must also keep in mind that we hope our models can become superior to the weak supervision label model. It is possible that the more complex FFN, LSTM and CNN models obtain a higher performance because they are better at imitating the label model, but worse at generalizing summary quality. We therefore cannot really know which model is better, even though the FFN, LSTM and CNN models get better loss and classification scores. We therefore argue that there are many good reasons to choose the LinTrans models over the other models.

The non-linear FFN and LSTM models are also very good in terms of loss and accuracy scores, but when we evaluate the distribution of quality measures, we see that these models do not get a desirable behaviour. We would like a model to pick up patterns that are implications of either a good or a bad summary, and we want the final quality measure to be a combination of these patterns. However, it seems like the non-linear FFN and LSTM models only find patterns that define *whether* a summary is good or not, and it thereby behaves more like a classification model. If the goal was to classify summaries as good or bad, these model architectures would be good. However, since we are interested in measuring quality on a continuous scale, we do not recommend using the non-linear FFN and LSTM models, despite the good performance they get in terms of loss and classification scores.

The CNN models get the best classification results, and also have very desirable density plots. Thus, it is tempting to conclude that these are the best performing models. If we are confident in our weak supervision labels, it makes sense to draw this conclusion. However, we do not really know how accurate the labels are, and thus, we do not know if the higher performance of CNN over LinTrans is because it truly is better, or because it simply is better at imitating the weak supervision label model. If the latter is the case, the LinTrans models might actually be better, even though the performance results indicate that the CNN models are better. We do, however, note that the CNN architecture seems to be very effective for discovering patterns in long documents. If labels with a known high quality were available, there is reason to believe that the CNN models would outperform the LinTrans models.

Finally, we note that an ensemble model also can be a good choice for measuring summary quality. For example, by taking the maximum or average of the output quality from LSA+LinTrans, Doc2vec+LinTrans, EmbLayer+CNN and Word2vec+CNN, we will get a final model that can include patterns from all of the models, and which are based on different embedders. Furthermore, single models are expected to make errors every now and then, but it is less likely that all the models make the same error at the same time, and thus, an ensemble model might be more robust. Since we do not really know whether the LinTrans or CNN models are better, an ensemble model can be a natural choice for evaluating summary quality.

6.2 Are Bad Summaries a Source of Conflict?

The motivation behind this work comes from the following facts:

1. Studies have suggested that less than 50% of buyers actually read the full real estate condition reports. This implies that many rely on the summaries to give them necessary information about the condition of the real estate they are buying.

2. Huseiernes Landsforbund reported in 2017 that 10% of real estate transactions end in conflict.

We want to understand why so many real estate transactions end in conflict. Since we also know that many buyers only read the summary, we wanted to investigate if the high conflict level could be explained by bad summaries. In this work, we have therefore created models that measure summary quality.

The various models have been applied to the real estate condition reports dataset, in order to measure the general quality of the summaries. The analysis has shown that 28.1%, 26.1%, 34.8% and 35.1% of the summaries are predicted to have a bad summary, according to the models LSA+FFN, Doc2vec+FFN, EmbLayer+CNN and Word2vec+CNN, respectively. If we use the average of the models, 30.7% would be estimated as bad, and if we use the maximum of the models, this number would be 20%.

These numbers are rather high. If we assume that the AvgEnsemble is precise, and that 50% only read the summaries, then 15.35% of real estate buyers are, in fact, not properly informed about the condition of the real estate they are buying. This can certainly be a part of the reason why so many real estate transactions end in conflict: Readers might expect the summaries to give them an accurate description of the real estate condition. However, the models indicate that this is often not the case.

A measure to reduce the conflict level could therefore be to ensure that the summary quality improved. The models of this work could be used by real estate agents, as a tool to inform them of the quality of the summaries they write. We do, however, expect that real estate agents are able to write good summaries without such a tool. Instead, it seems like there is not enough focus on the importance of the summaries, and the first attempt to reduce the problem could therefore be to enlighten real estate agents of the current problematic situation.

Finally, models of this work could also be useful for buyers in real estate transactions. In particular, the models give an indication of the summary quality. If this was given to

the reader of a real estate condition report, he/she would know what to expect from the summary. This would possibly reduce the risk of being surprised by the condition of the real estate.

6.3 Future Work

There are several directions in which this work could be taken to improve our ability to measure summary quality. First of all, we have only investigated a few model architectures, within the general approach of making embeddings into the conceptual summary content space. New model architectures could always be proposed, with or without using the idea of the conceptual summary content space.

Furthermore, we have only tried a handful of pre-existing embedding techniques. We have generally chosen embedders for their ability to embed arbitrarily long documents. However, the LSTM and CNN architectures are not based on embeddings of long documents, and therefore, state-of-the-art methods like BERT could be incorporated into these architectures, which we expect would increase the performance.

If one were to use more complex embedding techniques, we note that it would be an advantage if they could be trained on unlabelled data. We are not really sure of the quality of the weak supervision labels, and there is no point in training very complex models if the labels are limiting the quality of the models anyway. But models like BERT, which are self-supervised, can be (pre-)trained on unlabelled data. We therefore expect that the performance of for example Word2vec+CNN could be improved by replacing Word2vec with BERT.

However, as we have previously discussed, we do not know what happens when we maximize the performance on the weak supervision labels. We can expect that the labels are not perfect, and thus, we will at some point no longer benefit from increasing the performance on the labels. We therefore do not think that the best way to improve upon this work is to keep maximizing the performance on the weak supervision labels.

Instead, we recommend putting more effort into finding better evaluation methods. The biggest challenge of this work has been the fact that whenever we get results, we still do not really know how well we are performing. The natural way to solve this problem is to make an expert-made hand-labelled dataset of the true summary quality. Ideally, this labelled dataset would be sufficiently large to train supervised models, but that is probably not feasible. However, by making a smaller labelled dataset, only meant for testing purposes, it would be possible to properly evaluate the performances of the models. We could then properly conclude how well the weak supervision approach works. We believe that this is the right way to proceed if the goal is to improve our ability to measure summary quality.

Bibliography

Bach, Stephen H., et al. 2017. “Learning the Structure of Generative Models Without Labeled Data”. In *Proceedings of the 34th International Conference on Machine Learning*, 70:273–282. Sydney, NSW, Australia: JMLR.org.

Bach, Stephen H., et al. 2019. “Snorkel DryBell: A Case Study in Deploying Weak Supervision at Industrial Scale”. In *Proceedings of the 2019 International Conference on Management of Data*, 362–375. New York, NY, USA: Association for Computing Machinery.

Blei, David M., Andrew Y. Ng, and Michael I. Jordan. 2003. “Latent Dirichlet Allocation”. *The Journal of Machine Learning Research* 3:993–1022.

Bringer, Eran, et al. 2019. “Osprey: Weak Supervision of Imbalanced Extraction Problems without Code”. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning*, 1–11. New York, NY, USA: Association for Computing Machinery.

Cer, Daniel, et al. 2017. “SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation”. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 1–14. Vancouver, Canada: Association for Computational Linguistics.

Deerwester, Scott, et al. 1990. “Indexing by Latent Semantic Analysis”. *Journal of the American Society for Information Science* 41 (6): 391–407.

Devlin, Jacob, et al. 2018. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. *arXiv preprint arXiv:1810.04805*.

Dolan, Bill, Chris Quirk, and Chris Brockett. 2004. “Unsupervised Construction of Large Paraphrase Corpora: Exploiting Massively Parallel News Sources”. In *Proceedings of the 20th International Conference on Computational Linguistics*, 350–356. Geneva, Switzerland: Association for Computational Linguistics.

Gong, Hongyu, et al. 2019. “Document Similarity for Texts of Varying Lengths via Hidden Topics”. *arXiv preprint arXiv:1903.10675*.

Harris, Zellig S. 1954. “Distributional Structure”. *Word* 10 (2): 146–162.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. “Long Short-Term Memory”. *Neural Computation* 9 (8): 1735–1780.

Hoogeveen, Doris, Karin M. Verspoor, and Timothy Baldwin. 2015. "CQADupStack: A Benchmark Data Set for Community Question-Answering Research". In *Proceedings of the 20th Australasian Document Computing Symposium*, 1–8. Parramatta, NSW, Australia: Association for Computing Machinery.

Jurafsky, Daniel, and James H. Martin. 2019. *Speech and Language Processing (3rd Edition Draft)*. Visited on 04/22/2020. `https://web.stanford.edu/~jurafsky/slp3/`.

Kingma, Diederik, and Jimmy Ba. 2014. "Adam: A Method for Stochastic Optimization". *International Conference on Learning Representations*.

Kiros, Ryan, et al. 2015. "Skip-Thought Vectors". *arXiv preprint arXiv:1506.06726*.

Lau, Jey Han, and Timothy Baldwin. 2016. "An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation". *arXiv preprint arXiv:1607.05368*.

Le, Quoc V., and Tomas Mikolov. 2014. "Distributed Representations of Sentences and Documents". *arXiv preprint arXiv:1405.4053*.

Lin, Chin-Yew. 2004. "ROUGE: A Package for Automatic Evaluation of Summaries". In *Text Summarization Branches Out*, 74–81. Barcelona, Spain: Association for Computational Linguistics.

Liu, Ming, et al. 2017. "Measuring Similarity of Academic Articles with Semantic Profile and Joint Word Embedding". *Tsinghua Science and Technology* 22 (6): 619–632.

Lloret, Elena, Laura Plaza, and Ahmet Aker. 2018. "The Challenging Task of Summary Evaluation: An Overview". *Language Resources and Evaluation* 52 (1): 101–148.

Loh, Po-Ling, and Martin J. Wainwright. 2013. "Structure Estimation for Discrete Graphical Models: Generalized Covariance Matrices and Their Inverses". Publisher: Institute of Mathematical Statistics, *The Annals of Statistics* 41 (6): 3022–3049.

Loper, Edward, and Steven Bird. 2002. "NLTK: The Natural Language Toolkit". *arXiv preprint arXiv:cs/0205028*.

Mikolov, Tomas, et al. 2013a. "Distributed Representations of Words and Phrases and their Compositionality". *arXiv preprint arXiv:1310.4546*.

Mikolov, Tomas, et al. 2013b. "Efficient Estimation of Word Representations in Vector Space". *arXiv preprint arXiv:1301.3781*.

Olsen, Joakim. 2020. "Measuring Summary Quality using Document Distance". *Technical Report (Project Thesis) from Norwegian University of Science and Technology, Department of Mathematical Sciences*.

Paszke, Adam, et al. 2019. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". *Advances in Neural Information Processing Systems* 32:8026–8037.

Ratner, Alexander, et al. 2016. "Data Programming: Creating Large Training Sets, Quickly". In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 3574–3582. Red Hook, NY, USA: Curran Associates Inc.

Ratner, Alexander, et al. 2017. “Snorkel: Rapid Training Data Creation with Weak Supervision”. *Proceedings of the VLDB Endowment* 11 (3): 269–282.

Ratner, Alexander, et al. 2019. “Training Complex Models with Multi-Task Weak Supervision”. *Proceedings of the AAAI Conference on Artificial Intelligence* 33:4763–4771.

Rehurek, Radim, and Petr Sojka. 2010. “Software Framework for Topic Modelling with Large Corpora”. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 45–50.

Reimers, Nils, and Iryna Gurevych. 2019. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. *arXiv preprint arXiv:1908.10084*.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. 1986. “Learning Internal Representations by Error Propagation”. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, 318–362. Cambridge, MA, USA: MIT Press.

Rus, Vasile, Nobal Niraula, and Rajendra Banjade. 2013. “Similarity Measures Based on Latent Dirichlet Allocation”. In *Computational Linguistics and Intelligent Text Processing*, 459–470. Berlin, Heidelberg: Springer.

Själander, Magnus, et al. 2020. “EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure”. *arXiv preprint arXiv:1912.05848*.

Vrbanec, Tedo, and Ana Meštrović. 2020. “Corpus-Based Paraphrase Detection Experiments and Review”. *Information* 11 (5): 241.

Wieting, John, et al. 2016. “Towards Universal Paraphrastic Sentence Embeddings”. *arXiv preprint arXiv:1511.08198*.

Appendix A

Hyperparameters

In this work, optimal hyperparameters for all models are found by a thorough testing of various configurations. The main goal is to minimize the noise-aware cosine embedding loss, as defined in (4.5), on the validation set. A secondary goal is to maximize the classification scores, as defined in (4.9). Note that the baseline models LSA and Doc2vec will not be evaluated on the loss, since they are not trained on this type of loss function. In the following sections, the results of the various configurations will be given, and the optimal hyperparameter choice will be discussed.

A.1 LSA

The results with various configurations of the baseline LSA model is given in Table A.1. The following hyperparameters are tested:

K Dimensionality of topic vectors.

`nb` Minimum number of documents a word has to appear in, for it to be included in the vocabulary.

`rs` Boolean indicator of whether stop-words are removed, where True indicates that stop-words are removed.

Table A.1 shows that the no-below-parameter (`nb`) influences the performance of the LSA model a lot. In particular, LSA seems to benefit from a high value of `nb`, which results in a small vocabulary. Surprisingly, we find that the model performs slightly better when stop-words are included, that is, with `rs` $=$ False. Finally, we see that the model seems to benefit slightly from having a higher dimensionality K, up until $K = 500$. We therefore choose to use $K = 500$, `nb` $= 15000$, and `rs` $=$ False in our final model in Section 4.3.2.

Note that with `nb` $= 15000$, the resulting vocabulary has 513 words. This is a very small vocabulary, and it seems like the best way to control the dimensionality of LSA in fact is to limit the vocabulary, instead of keeping only the K most significant topics. To

Table A.1: Influence in performance of various configurations of hyperparameters on the validation set for LSA. A few results with TF-IDF are also included for comparison.

Embedder	K	`nb`	`rs`	**Acc**	$\mathbf{F}_1$
LSA	100	1	True	0.683	0.721
·	·	3	·	0.683	0.712
·	·	5	·	0.681	0.727
·	·	10	·	0.682	0.711
·	·	20	·	0.682	0.711
·	·	40	·	0.682	0.727
·	·	100	·	0.683	0.727
·	·	500	·	0.687	0.725
·	·	1000	·	0.693	0.721
·	·	2000	·	0.702	0.713
·	·	5000	·	0.716	0.738
·	·	10000	·	0.719	0.737
·	·	20000	·	0.722	0.749
·	·	50000	·	0.700	0.725
·	·	30000	·	0.700	0.727
·	·	22000	·	0.721	0.746
·	·	15000	·	0.720	0.736
·	·	18000	·	0.717	0.731
·	·	20000	False	0.723	0.744
·	50	·	·	0.716	0.739
·	200	·	·	0.725	0.756
·	300	·	·	0.726	0.759
·	500	·	·	0.727	0.747
·	1000	·	·	0.727	0.747
·	500	·	True	0.724	0.748
LSA	**500**	**15000**	**False**	**0.733**	0.759
·	·	10000	·	0.728	0.758
·	·	17000	·	0.730	0.755
·	·	13000	·	0.730	0.763
·	1000	·	·	0.731	**0.765**
·	·	15000	·	**0.733**	0.762
TFIDF	-	20	False	0.720	0.736
·	-	15000	·	0.732	0.761
·	-	10000	·	0.729	0.759
·	-	20000	·	0.726	0.752

investigate further, we also perform a few experiments with TF-IDF alone. The results indicate that TF-IDF is equal in performance as LSA, and we obtain similar results for TF-IDF with $\texttt{nb} = 15000$ as we do for LSA with $\texttt{nb} = 15000$ and $K = 500$. This is natural since LSA barely removes any dimensions from the TF-IDF model in this case, and thus, the two models are based on the same information. We therefore note that using TF-IDF alone is also a very good baseline model on the summary quality problem.

A.2 Doc2vec

The results with various configurations of Doc2vec are given in Table A.2. The following hyperparameters have been tested:

K Dimensionality of feature vectors.

`mc` The minimum number of times word must appear for it to be included in the vocabulary.

R Window size.

`e` Number of training epochs.

Table A.2 shows that 50 epochs seems to be sufficient for the Doc2vec model to converge. Furthermore, the window size R and minimum count-parameter `mc` does not seem to influence the performance very much. These are therefore, rather arbitrarily, set to $R = 6$ and $\texttt{mc} = 20$ in the final model in Section 4.3.2. The model also seems to perform better if K is kept relatively low, and is therefore set to $K = 100$ in the final model.

A.3 FFN

For the fully connected feed-forward neural network, the following hyperparameters are optimized over:

`emb` Embedder that the FNN is placed on top of.

K Dimensionality of the embedder.

`nb` When using the LSA embedder, a few configurations of this hyperparameter is tested since it influences the performance of LSA a lot. See Appendix A.1 for a description.

`layers` A vector of how many nodes to use in each layer. The length of the vector will be equal to the number of layers. Note that the network always has K input neurons.

`d` Dropout, which is a regularization hyperparameter. For each training batch, the values in a fraction of nodes (given by `d`) are set equal to zero. This is an effective technique against over-fitting to training data. Note that dropout is only used in the hidden layers, and therefore does not apply when the network is a single, linear layer.

Table A.2: Influence in performance of various configurations of hyperparameters on the validation set for Doc2vec.

K	mc	R	e	**Acc**	$\mathbf{F}_1$
100	20	6	10	0.672	0.675
·	·	·	20	0.671	0.675
·	·	·	30	0.674	0.682
·	·	·	40	0.679	0.679
100	**20**	**6**	**50**	**0.683**	0.681
·	·	·	100	**0.683**	0.692
·	0	·	50	0.676	0.681
·	5	·	·	0.680	0.699
·	10	·	·	0.680	0.668
·	50	·	·	0.680	0.698
·	1000	·	·	0.680	0.669
·	10000	·	·	0.660	0.677
·	20	1	·	0.680	0.669
·	·	3	·	0.677	0.657
·	·	10	·	0.682	0.680
80	·	6	·	0.680	0.673
200	·	·	·	0.682	**0.700**
300	·	·	·	0.672	0.685
500	·	·	·	0.650	0.650

`lr` Learning rate, which determines how large steps the Adam learning algorithm should take. Note that the learning rate will be reduced by a factor of 0.1 after one third of the epochs, and again after two thirds of the epochs, such that the model takes smaller steps towards the end of the training process. Thus, the `lr` parameter determines how large the step size should be initially.

B Batch size, that is, the number of samples used by the network for each calculation of the gradient in the Adam learning algorithm.

Note that the values of `rs` for LSA, as well as `mc`, R and `e` for Doc2vec, are the same as in the baseline models in Appendix A.1 and A.2. Note also that the number of epochs is not included as a hyperparameter. This is because we monitor the training process, and ensure that the model converges with respect to the loss, within the given number of epochs. In particular, we ensure that the model converges for all values of the learning rate, which updates after one third and two thirds of the epochs. We observe that using 30 epochs normally is sufficient for convergence, while a higher number of epochs sometimes is necessary.

We also consider using batch normalization in the neural networks, which is a commonly used regularization method that often speeds up training. However, we find that this technique consistently decreases performance notably, and we therefore omit this technique.

The results for various configurations are given in Table A.3. For the embedders,

Table A.3: Influence in performance of various configurations of hyperparameters on the validation set for FFN.

emb	nb	K	layers	B	d	lr	**T-Loss**	**V-Loss**	**Acc**	$\mathbf{F_1}$
LSA	15K	500	[100]	64	-	1e-3	0.075	0.094	0.871	0.882
·	20K	100	·	·	-	·	0.097	0.098	0.852	0.863
·	20	·	·	·	-	·	0.099	0.099	0.850	0.861
·	·	500	·	·	-	·	0.079	0.093	0.866	0.876
·	5K	·	·	·	-	·	0.076	0.093	0.873	0.882
·	15K	1K	·	·	-	·	0.075	0.094	0.869	0.880
·	·	500	·	32	-	·	0.075	0.094	0.869	0.882
·	·	·	·	128	-	·	0.075	0.094	0.871	0.882
·	·	·	·	64	-	0.01	0.075	0.094	0.870	0.881
LSA	**15K**	**500**	**[100]**	**64**	-	**1e-4**	0.081	0.092	0.873	0.883
·	·	·	·	·	-	1e-5	0.095	0.097	0.858	0.870
·	·	·	[500]	·	-	1e-4	0.079	0.092	0.874	0.884
·	·	·	[500, 100]	·	0.3	·	0.074	0.089	0.884	0.890
·	·	·	[1K, 100]	·	·	·	0.070	0.088	0.885	0.892
·	·	·	[1K, 1K, 100]	·	·	·	0.066	0.078	**0.888**	**0.896**
·	·	·	[1K x 3, 100]	·	·	·	0.067	0.078	0.881	0.889
·	·	·	[1K, 1K, 100]	·	0.4	·	0.068	0.079	0.886	0.891
LSA	**15K**	**500**	**[1K, 1K, 100]**	**64**	**0.2**	**1e-4**	0.064	0.078	**0.888**	**0.896**
·	·	·	·	·	0.1	·	0.062	0.079	0.886	0.894
·	·	·	·	·	0	·	**0.061**	0.087	0.884	0.892
TF-IDF	15K	513	[1K, 1K, 100]	64	0.2	1e-4	0.064	**0.077**	0.885	0.894
Doc2vec	-	100	[100]	64	-	1e-3	0.100	0.102	0.839	0.847
·	-	500	·	·	-	·	0.074	0.100	0.851	0.861
·	-	·	·	32	-	·	0.074	0.100	0.853	0.865
·	-	·	·	128	-	·	0.074	0.100	0.853	0.863
·	-	·	·	64	-	0.01	0.074	0.100	0.851	0.863
Doc2vec	-	**500**	**[100]**	**64**	-	**1e-4**	0.080	0.099	0.854	0.867
·	-	·	·	·	-	1e-5	0.094	0.101	0.843	0.854
·	-	·	[500]	·	-	1e-4	0.077	0.097	0.861	0.871
·	-	·	[500, 100]	·	0.3	·	0.071	0.090	0.885	0.893
·	-	·	[1K, 100]	·	·	·	0.067	0.090	0.883	0.892
·	-	·	[1K, 1K, 100]	·	·	·	0.066	0.085	**0.890**	**0.897**
·	-	·	[1K x 3, 100]	·	·	·	0.064	0.077	0.886	0.893
·	-	·	[1K x 4, 100]	·	·	·	0.067	0.077	0.883	0.889
Doc2vec	-	**500**	**[1K x 3, 100]**	**64**	**0.2**	**1e-4**	0.062	**0.076**	0.888	0.894
·	-	·	[1K x 3, 100]	·	0.1	·	0.060	0.078	0.885	0.893
·	-	·	[1K x 3, 100]	·	0	·	**0.057**	0.080	0.883	0.890

we see that the no-below parameter nb for LSA does not influence the result that much when used together with the FFN. Thus, it seems like the FFN can filter out the important information concerning the vocabulary in a better way than LSA alone can do. However, the results with a high value of nb are still slightly better, and we therefore use $\texttt{nb} = 15000$ in the final model for LSA+FFN. Furthermore, we see that increasing the dimensionality K of the embedder increases the performance for both LSA and Doc2vec, again up until $K = 500$. We therefore use $K = 500$ in the final models for LSA+FFN and Doc2vec+FFN.

The batch size parameter B does not seem to influence the performance, and we therefore, rather arbitrarily, use $B = 64$ in the final models. The initial learning rate lr, on the other hand, influences the performance, and we choose to use an initial value of $\texttt{lr} = 1 \times 10^{-4}$ in the final model since this gives the best validation loss. Increasing the number of nodes in the output layer, and thereby the dimensionality of $\mathbf{z_r}$ and $\mathbf{z_s}$, does not seem to influence the result very much, and we therefore use 100 neurons in the output layer. However, we see that increasing the number of hidden neurons, and the depth of the FFN increases performance a lot, up until a certain point. In particular, we find that LSA+FFN is best with two hidden layers of 1000 neurons each, while Doc2vec+FFN is best with three hidden layers of 1000 neurons each. Finally, we tune the dropout parameter, and we get the best validation loss with $\texttt{d} = 0.2$. Note that for all the highlighted configurations, which are included in Chapter 5, 30 epochs is used, which seems to be sufficient for convergence.

We also include a result with TF-IDF+FFN. Again, we observe that the results of TF-IDF and LSA are similar since they are based on more or less the same amount of information. Hence, we note that TF-IDF is a very good alternative to LSA and Doc2vec when working with long documents.

A.4 LSTM

For the LSTM network, the following hyperparameters are optimized over:

emb Embedder that the LSTM is placed on top of.

K Dimensionality of the embedder.

nb When using the LSA embedder, a few configurations of this hyperparameter are tested since it influences the performance of LSA a lot. See Appendix A.1 for a description.

lay Number of LSTM layers that are placed on top of each other.

dim Number of neurons in the LSTM layers.

bd Boolean indicator of whether LSTM layers are bi-directional or not.

on Number of output neurons in the final, linear layer, that is, the dimensionality of $\mathbf{z_r}$ and $\mathbf{z_s}$.

d Dropout. See Appendix A.3 for a description.

Table A.4: Influence in performance of various configurations of hyperparameters on the validation set for LSTM.

LSA+LSTM

nb	K	dim	lay	bd	on	B	d	lr	**T-Loss**	**V-Loss**	**Acc**	**F_1**
15K	500	100	1	False	100	64	0.1	1e-3	0.074	**0.078**	0.882	0.892
20K	100	·	·	·	·	·	·	·	0.083	0.084	0.862	0.877
20	100	·	·	·	·	·	·	·	0.086	0.084	0.859	0.873
20	500	·	·	·	·	·	·	·	0.075	**0.078**	0.883	**0.893**
15K	·	·	·	·	·	32	·	·	0.074	0.079	0.879	0.889
·	·	·	·	·	·	128	·	·	0.076	0.081	0.878	0.889
·	·	·	·	·	·	64	·	0.01	**0.072**	0.079	0.878	0.888
·	·	·	·	·	·	·	·	1e-4	0.076	0.082	0.880	0.891
15K	**500**	**100**	**1**	**False**	**100**	**64**	**0**	**1e-3**	**0.072**	**0.078**	**0.885**	**0.893**
·	·	·	·	·	·	·	0.3	·	0.076	0.080	0.874	0.885
·	·	·	·	True	·	·	0	·	**0.072**	0.079	0.882	0.889
·	·	·	·	False	500	·	·	·	0.074	**0.078**	**0.885**	**0.893**
·	·	500	·	·	100	·	·	·	0.073	0.080	0.876	0.883
·	·	100	2	·	·	·	·	·	0.074	0.079	0.875	0.883
·	·	·	·	·	·	·	0.3	·	0.079	0.081	0.867	0.876
·	·	500	·	·	·	·	0.1	·	0.074	0.079	0.874	0.882

Doc2vec+LSTM

nb	K	dim	lay	bd	on	B	d	lr	**T-Loss**	**V-Loss**	**Acc**	**F_1**
-	100	100	1	False	100	64	0.1	1e-3	0.079	0.081	0.871	0.883
-	500	·	·	·	·	·	·	·	0.073	0.078	0.880	0.890
-	·	·	·	·	·	32	·	·	0.072	0.078	0.877	0.888
-	·	·	·	·	·	128	·	·	0.075	0.079	0.879	0.890
-	·	·	·	·	·	64	·	0.01	0.074	0.079	0.875	0.883
-	·	·	·	·	·	·	·	1e-4	0.073	0.081	0.881	0.888
-	**500**	**100**	**1**	**False**	**100**	**64**	**0**	**1e-3**	**0.071**	**0.077**	**0.883**	**0.891**
-	·	·	·	·	·	·	0.3	·	0.075	0.079	0.878	0.888
-	·	·	·	True	·	·	0.1	·	0.075	0.079	0.878	0.888
-	·	·	·	False	500	·	·	·	0.073	**0.077**	0.882	**0.891**
-	·	500	·	·	100	·	·	·	0.073	0.078	0.877	0.885
-	·	100	2	·	·	·	·	·	0.077	0.079	0.874	0.883
-	·	500	·	·	·	·	·	·	0.078	0.079	0.873	0.882
-	·	·	·	·	·	·	0.3	·	0.076	0.078	0.876	0.885
-	·	·	·	·	·	·	0	·	0.079	0.081	0.868	0.879

`lr` Learning rate. See Appendix A.3 for a description.

B Batch size. See Appendix A.3 for a description.

Again, the embedder parameters `rs`, `mc`, R and `e` are the same as in the baseline models. Furthermore, the number of epochs is not included as a hyperparameter since we monitor the training process and ensure that the model converges. We find that 30 epochs generally is sufficient for most configurations. However, for the more complex configurations, 60 epochs is used instead.

The results are given in Table A.4. We see again that for LSA, the `nb` parameter does not influence the result very much, which means that the LSTM network also manages to filter out the important information concerning the vocabulary. We also see again that the network benefits from using a higher embedder dimensionality. In the final models

for LSA+LSTM and Doc2vec+LSTM, we therefore choose to use $K = 500$, as well as $\texttt{nb} = 15000$ for LSA.

Furthermore, we again find that the batch size B does not influence the results, while the learning rate parameter $\texttt{lr}$ slightly does. In the final models use $B = 64$ and $\texttt{lr} = 1 \times 10^{-3}$, since the latter gives the best validation loss. The LSTM model also gets the best results for $\texttt{d} = 0$, that is, without dropout employed. This is interesting, and shows that this type of regularization does not boost performance for the LSTM model. Finally, we observe that increasing the complexity of the LSTM model by increasing $\texttt{dim}$, $\texttt{lay}$, $\texttt{bd}$ and/or $\texttt{on}$ does not increase performance. Hence, we use $\texttt{dim} = 100$, $\texttt{lay} = 1$, $\texttt{bd} =$ False and $\texttt{on} = 100$ in the final LSA+LSTM and Doc2vec+LSTM models.

A.5 CNN

For the CNN network, the following hyperparameters are optimized over:

V Number of words in the vocabulary. For the EmbLayer+CNN model, we use the V most frequent words in the vocabulary, while for Word2vec, this is controlled by the minimum count parameter $\texttt{mc}$, which was described in Appendix A.2.

R For Word2vec, we consider different window sizes R.

K Dimensionality of the word embeddings.

$\texttt{dim}$ Number of filters for each filter size. We also use this many neurons in the final, linear output layer.

$\texttt{filters}$ A list of filter sizes to use in the CNN.

$\texttt{d}$ Dropout. See Appendix A.3 for a description.

$\texttt{lr}$ Learning rate. See Appendix A.3 for a description.

B Batch size. See Appendix A.3 for a description.

For the Word2vec embedder, we use 50 epochs in the training process. This is a natural choice since the Word2vec network architecture is very similar to the Doc2vec architecture, and it seems like Doc2vec converges within 50 epochs. We have also tested increasing the number of epochs, and observe similar results. For the CNN model, we monitor the training process to ensure that we use a sufficient amount of epochs. As before, we find that 30 epochs generally is enough.

The results from different configurations of CNN are given in Table A.5. We observe that the vocabulary size V does not seem to influence the result very much. We choose to use $V = 20\,000$, since this is a sensible choice of vocabulary size. Furthermore, since V does not seem to be that important, we do not expect the minimum count parameter $\texttt{mc}$ to be that important either. We choose to use $\texttt{mc} = 20$ in the Word2vec model, which results in a vocabulary size of $V = 26\,533$. For the EmbLayer+CNN model, we observe an improvement in performance by increasing the dimensionality to $K = 500$. However, for Word2vec+CNN, the results do not improve for $K = 500$ over $K = 100$. We therefore

Table A.5: Influence in performance of various configurations of hyperparameters on the validation set for CNN.

EmbLayer+CNN

V	R	K	dim	filters	B	d	lr	**T-Loss**	**V-Loss**	**Acc**	$\mathbf{F_1}$
10K	-	100	100	[5]	64	0.1	1e-3	0.073	0.089	0.887	0.896
5K	-	·	·	·	·	·	·	0.074	0.089	0.886	0.895
20K	-	·	·	·	·	·	·	0.073	0.090	0.887	0.895
50K	-	·	·	·	·	·	·	0.073	0.090	0.885	0.892
20K	-	50	·	·	·	·	·	0.077	0.090	0.881	0.890
·	-	200	·	·	·	·	·	0.070	0.088	0.888	0.897
·	-	500	·	·	·	·	·	0.068	0.087	0.885	0.895
·	-	1K	·	·	·	·	·	0.067	0.087	0.889	0.898
·	-	500	·	·	32	·	·	0.067	0.087	0.888	0.896
·	-	·	·	·	128	·	·	0.069	0.089	0.885	0.892
·	-	·	·	·	64	·	0.01	0.064	0.088	0.887	0.895
·	-	·	·	·	·	·	1e-4	0.077	0.091	0.885	0.894
·	-	·	·	·	·	·	0.1	0.063	0.090	0.883	0.892
20K	-	**500**	**500**	**[5]**	**64**	**0.1**	**1e-3**	0.063	**0.085**	**0.893**	0.900
·	-	·	200	[3, 5, 7]	·	·	·	0.062	**0.085**	**0.893**	**0.902**
·	-	·	·	·	·	0.2	·	0.064	**0.085**	**0.893**	**0.902**
·	-	·	·	·	·	0	·	**0.060**	0.087	**0.893**	0.901

Word2vec+CNN

V	R	K	dim	filters	B	d	lr	**T-Loss**	**V-Loss**	**Acc**	$\mathbf{F_1}$
27K	3	100	100	[5]	64	0.1	1e-3	0.073	0.085	0.894	0.900
·	6	·	·	·	·	·	·	0.073	0.084	0.893	0.901
·	10	·	·	·	·	·	·	0.073	0.083	0.897	0.904
·	3	500	·	·	·	·	·	0.068	0.085	0.893	0.900
·	6	·	·	·	·	·	·	0.067	0.084	0.893	0.902
·	10	·	·	·	·	·	·	0.068	0.084	0.895	0.903
·	·	100	·	·	·	0.3	·	0.080	0.086	0.892	0.901
·	6	500	·	·	·	·	·	0.075	0.086	0.898	0.905
·	10	100	·	·	32	0.1	·	0.073	0.084	0.891	0.902
·	·	·	·	·	128	·	·	0.074	0.085	0.894	0.901
·	·	·	·	·	64	·	0.01	0.073	0.084	0.896	0.904
·	·	·	·	·	·	·	1e-4	0.080	0.087	0.890	0.900
·	·	·	·	·	·	·	0.1	0.075	0.085	0.892	0.900
·	·	·	500	·	·	·	0.01	0.074	0.083	0.894	0.899
·	·	·	100	[2, 3, 5, 7, 10]	·	·	·	0.066	0.083	0.896	0.905
27K	**10**	**100**	**200**	**[2, 3, 5, 7, 10]**	**64**	**0.1**	**0.01**	0.064	**0.082**	**0.900**	**0.906**
·	·	·	·	·	·	0.2	·	0.066	**0.082**	0.898	0.905
·	·	·	·	·	·	0	·	**0.061**	0.083	0.898	**0.906**

use $K = 100$ for Word2vec+CNN, with $R = 10$ since this gives slightly better validation loss.

As before, we find that the batch size B influences the performance very little, and we use $B = 64$ in the final model. The initial learning rate has a slight impact on the performance, and we find that $\texttt{lr} = 1 \times 10^{-3}$ performs best for the EmbLayer+CNN model, while $\texttt{lr} = 1 \times 10^{-2}$ performs better for the Word2vec+CNN model. This difference can be explained by the fact that the word embeddings are trained in the EmbLayer+CNN model, while they are not in the Word2vec+CNN model. It is therefore sensible that the EmbLayer+CNN model benefits from smaller initial step sizes. When we increase the complexity of the models, by introducing several different filter sizes with more filters per filter size, we also observe a slight performance increase. We choose to use $\texttt{filters} = [5]$ and $\texttt{dim} = 500$ in the final EmbLayer+CNN model, while we use $\texttt{filters} = [2, 3, 5, 7, 10]$ and $\texttt{dim} = 200$ in the final Word2vec+CNN model. Finally, we observe that using dropout is beneficial with respect to validation loss, but there is no particular gain in using a high dropout value $\texttt{d}$. We therefore use $\texttt{d} = 0.1$ in the final CNN models.

Appendix B

Examples of Real Estate Condition Report Summaries

The following summaries are English translations of the Norwegian summary examples given in Section 5.4.2, as translated by Google Translate.

1. **ID**: 5d80ea54-0ace-418b-95c7-2cad2dc93ec5
   Summary: Detached house from 1978 which is maintained and has a good standard, age taken into account. It is somewhat lavish over time otherwise it is original. There is a hipped roof with a table roof. Gutters and downspouts. Bonding walls insulated with standing panel and wall cladding. Windows with wooden frame and frame with insulating glass. Solid exit door in teak. There is leca foundation and cast cover. The drainage is from the time of construction. Inside there are panels and panels in the ceiling, floors have tiles, coatings, laminate, carpets and parquet. Bathroom with tiles on the floor and walls with sanitary equipment that is from the time of construction. There is a separate toilet room and shower cubicle in the boiler room and a toilet with washbasin in the laundry room. Oak kitchen furniture with profiles on top cabinets and base cabinets from the construction period. Central heating for oil and electricity which is about 10 years. Oil tank under terrace. Electrical system with screw fuses. Garage from 1986 it is built with cast cover, leca ring wall, standing cladding. Hinged roof with concrete stone, gutters and downspouts in plastic-coated steel. There are 2 articulated gates. Normal aging and use wear has been registered on the property.

2. **ID**: e527b688-a1ec-4a7b-a04b-097868b1124a
   Summary: The home is located in an established residential area, with a short way to school, kindergarten and business. Remarks have been made that should be improved, such as wet rooms and upgrades due to normal wear and tear. Otherwise read the report.

3. **ID**: 84374a61-c152-49a4-bc0f-6dae06158ca7
   Summary: Detached house with normal standard located in Sveggen in Averøy

municipality. Generally built in good and well-known constructions with the construction method that was common at the time of construction. Deviations can be mentioned: There was an indication of moisture in the floor and outer walls in the shower zone in the bathroom, the reason may be that the shower had just been used. It was not possible to check under the floor. Further investigations may be necessary. There is only one drain in the floor and it is in the shower zone. The shower zone is made up of 2 walls and one edge down to the floor. This edge of 7 cm, which is tight, means that any leakage water will flow out over the doorstep and into adjacent rooms.

4. **ID**: d3479149-98e0-41af-9f68-26479c687a8d
   Summary: Detached house with crawl space, ground floor and attic which was built in about 1903. The house is built in a foundation wall in brick and stone, wooden construction which is clad on the outside with wood panels, wooden beams. Salt roof in wooden construction covered with slate roof tiles. The home has a normal maintenance but there is a need for some upgrades. The chimney is in poor condition on the outside and here there is a need for upgrades, it is damp in the crawl space and it is recommended to install a crawl space dehumidifier. The building is in good condition considering the age of the building. Registered condition remarks are mainly due to the building's age and maintenance as well as construction.

5. **ID**: 50bdfbaf-7642-4a17-8358-070257214b98
   Summary: The home probably with everything from the year of construction and therefore with some natural wear. Of significant importance regarding remarks, some moisture was registered in the shower wall on the 2nd floor due to leaks drain washing machine. All maintenance outside the section shall normally be a joint responsibility in the condominium.

Joakim Olsen

Measuring Summary Quality using Weak Supervision

NTNU
Norwegian University of
Science and Technology