NTNU
Innovation and Creativity

Text Mining in Health Records
Classification of Text to Facilitate Information Flow and Data Overview

Øystein Rose

Master of Science in Computer Science

Submission date:	July 2007
Supervisor:	Øystein Nytrø, IDI
Co-supervisor:	Thomas Brox Røst, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

A personal health record (PHR) is an electronic application that individuals can access, manage and use to share their health information with authorized people, in a private, secure, and confidential environment. This may give a more continuous flow of information between the doctor and the patient, ease surveillance of patients, and help disclose patient degree of compliance and satisfaction with the suggested treatment. Unlike the PHR, the electronic health record (EHR) is a tool that seeks to serve information needs of health care processionals.

Information flow between the mentioned systems is thought to ease and improve treatment of the patient. However, the information is often unstructured, making it difficult to extract the wanted information. In addition, health record information is often of such a character that it is very difficult to get an overview of a health history in a short matter of time.

The objective of this thesis is twofold. The first goal is to apply techniques from the field of text mining on encounter notes in the personal health record to help structure the different parts of the EHR encounter notes to ease information exchange with the PHR. The second goal is to apply the structures found in the encounter notes to investigate the applicability of structural information to present an overview of patient histories.

Assignment given: 22. January 2007
Supervisor: Øystein Nytrø, IDI

Abstract

This project consists of two parts. In the first part we apply techniques from the field of text mining to classify sentences in encounter notes of the electronic health record (EHR) into classes of *subjective*, *objective* and *plan* character. This is a simplification of the *SOAP* standard, and is applied due to the way GPs structure the encounter notes. Structuring the information in a subjective, objective, and plan way, may enhance future information flow between the EHR and the personal health record (PHR).

In the second part of the project we seek to use apply the most adequate to classify encounter notes from patient histories of patients suffering from diabetes. We believe that the distribution of sentences of a subjective, objective, and plan character changes according to different phases of diseases.

In our work we experiment with several preprocessing techniques, classifiers, and amounts of data. Of the classifiers considered, we find that Complement Naive Bayes (CNB) produces the best result, both when the preprocessing of the data has taken place and not. On the raw dataset, CNB yields an accuracy of 81.03%, while on the preprocessed dataset, CNB yields an accuracy of 81.95%. The Support Vector Machines (SVM) classifier algorithm yields results comparable to the results obtained by use of CNB, while the J48 classifier algorithm performs poorer.

Concerning preprocessing techniques, we find that use of techniques reducing the dimensionality of the datasets improves the results for smaller attribute sets, but worsens the result for larger attribute sets. The trend is opposite for preprocessing techniques that expand the set of attributes. However, finding the ratio between the size of the dataset and the number of attributes, where the preprocessing techniques improve the result, is difficult. Hence, preprocessing techniques are not applied in the second part of the project.

From the result of the classification of the patient histories we have extracted graphs that show how the sentence class distribution after the first diagnosis of diabetes is set. Although no empiric research is carried out, we believe that such graphs may, through further research, facilitate the recognition of points of interest in the patient history.

From the same results we also create graphs that show the average distribution of sentences of subjective, objective, and plan character for 429 patients after the first diagnosis of diabetes is set. From these graphs we find evidence that there is an overrepresentation of subjective sentences in encounter notes where the diagnosis of diabetes is first set. However, we believe that similar experiments for several diseases, may uncover patterns or trends concerning the diseases in focus.

Preface

This is a Master's thesis written at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU) during the spring semester of 2007. The project assignment is specified and written for the Norwegian Electronic Health Record Centre (NSEP). The supervisor of this thesis has been Øystein Nytrø, while Thomas Brox Røst has been my main advisor.

I would like to thank Øystein Nytrø for helping me specify the assignment and find new angles to the research questions, and for fruitful discussions concerning the experiments. I would also like to thank Thomas Brox Røst for his encouraging feedback during all the phases of planning, execution, and documentation, in addition to his coding help in Python. Finally, I would like to thank Ole Edsberg for his contributions with technical assistance and ideas, Anders Grimsmo for his contribution of the EHR data used in this project, and Torbjørn Nordgård for offering the use of his Norwegian computational dictionary.

Contents

List of Figures

List of Tables

1 Introduction

Due to increased use of the Internet, patients now call for the opportunity to be able to communicate with the public health service online in addition to traditional methods such as visiting the general practitioner (GP) (Andreassen et al., 2002). One kind of application that supports some of these new demands is the *personal health record* (PHR). This is an online health record where the patient him- or herself should be able to document health related actions, and how and to what extent medical treatment works (Markle Foundation, 2003; Tang et al., 2005).

GPs and other health care actors are supposed to be able to pay attention to, and give feedback on, the patient's treatment. A GP may consult the PHR both in cases where the GP wants to help a patient online, or when he wants to prepare him- or herself before a patient visit. This is thought to give increased follow-up of the patient and make it possible to gather both more correct and a larger amount of information from each patient (Brasethvik and Kofod-Petersen, 2006; Tang et al., 2005).

While the main target of the PHR is to capture health data entered by individuals and then give information related to the care of these patients, the *electronic health record* (EHR) seeks to serve the information needs of health care professionals. Even though the PHR and the EHR may be considered two separate systems, the flow of information between the two is important and beneficial (Tang et al., 2005; Crawford, 2006). However, extracting different parts of encounter notes in EHRs is not trivial due their free-text narrative nature (Røst et al., 2006b).

General usage of the PHR is likely to be dominated by people who suffer from chronic illness (Szolovits et al., 1994), hence creating a huge amount of information. The situation is the same for EHR systems, where little or nothing is offered with respect to future re-use of the data (Nilsson et al., 2003). Hence, the information load is often too extensive to be able to get a clear overview and understanding of the information available in a short matter of time for both systems.

In this report we seek to use text classification methods to help structure the data to ease the information flow between the EHR and the PHR, and then use the created classifiers to find ways of presenting the health record data in ways that may improve access to it and reveal trends.

In the next section we look in more detail into the research questions (RQs) of this project. In the latter sections of this chapter we present a list of limitations we consider important during the work, and an overview of the contents of this report.

1.1 Objective

In this project we seek to apply techniques from text classification, which is a subfield of text mining, to find possible solutions to some of the challenges described above. Text classification is defined as the activity of assigning predefined category labels to new documents based on the likelihood suggested by a training set of labelled documents (Sebastiani, 2002), and will be further elaborated in chapters to come.

Encounter notes in an EHR typically consist of four different kind of sentences: *Subjective*, *objective*, *assessment*, and *plan* sentences (Nilsson et al., 2003). Based on this structure and the help of techniques from the field of text classification we seek both to find ways to facilitate the flow of information between the EHR and the PHR, and use the created classifiers to find methods of presenting the huge amount of information in health record systems. The derived RQs are as follows:

RQ1: Are text classifiers able to discern between different kinds of sentences in the EHR in order to structure the information flow between the EHR and the PHR?

RQ2: Is the classifier derived from the work of RQ1 able to present histories of EHR encounter notes in ways that allow us to detect points of interest and trends?

Hence, we seek to find methods to structure and abstract the information to help information flow and information presentation. The background for these RQs will be further elaborated in Chapter 2.

1.2 Limitations

The field of health informatics is a complex field where one has to take both technical and domain specific aspects into consideration. Also, in the field of text mining, there are a number of ways to approach a problem, due to a wide range of algorithms and work methods. Thus, within the time schedule of this project, several limitations have to be made. We now look at the most important limitations in this project:

Data access: Due to limited access to real PHR data, data from an EHR system must be applied.

Selection of algorithms: There exists a vast amount of algorithms able to deal with the task of text classification. It is beyond the scope of this thesis to present a complete overview of the algorithms available, but rather present how a few applicable algorithms may solve the stated tasks.

Tweaking of algorithms: Many algorithms in the field of data mining take several parameters. Within the scope of the project we will not strive to find the *absolute optimal* set of tuned parameters for each algorithm.

SOAP standard: Even though we would like to structure the sentences strictly in an *SOAP* fashion, the way GPs have written their notes makes it far more feasible to classify *assessment* and *plan* sentences together, which we call *plan* sentences. Hence we classify the notes into the classes subjective (*S*), objective (*O*), and plan (*P*). This is further explained in the next chapter.

1.3 Overview

This section gives a short overview of the contents of the subsequent chapters. Description of each chapter:

Chapter 2, Text Mining in Health Records : Presentation of health record systems relevant to the RQs of this project, and incentives behind the RQs.

Chapter 3, Data in Profdoc Vision Allmenn : Overview of the EHR data to be applied in this project.

Chapter 4, Text Mining Techniques : Text classification techniques that may be used in the process of discerning between classes of sentences.

Chapter 5, Text Mining Tools : Tools that may facilitate the process of text classification.

Chapter 6, Text Classification Experiment Plan (RQ1) : Plan of text classification experiments performed in the work of RQ1, i.e., the choice of algorithms, preprocessing techniques, attribute sets and sizes of datasets.

Chapter 7, Text Classification Results : Results of experiments performed in the work of RQ1.

Chapter 8, Text Classification Evaluation and Discussion : Evaluation of results and statistics of the experiments performed in the work of RQ1, seeking to answer questions like which is the better classifier, both when considering preprocessing techniques and not.

Chapter 9, History Extraction Experiment Plan (RQ2): Plan of experiments performed in the work of RQ2, i.e., how the use of text classifiers derived from last experiments may be used to create a condensed view of health record data.

Chapter 10, History Extraction Results : Results of experiments performed in the work of RQ2.

Chapter 11, History Extraction Evaluation and Discussion : Evaluation of results and statistics of the experiments performed in the work of RQ2, seeking to answer questions like whether derived figures may be used to present a condensed view of the data.

Chapter 12, Conclusion : The most important conclusions of this project.

Chapter 13, Further Work : Overview of some of the further work that we have identified in this project.

2 Text Mining in Health Records

In this chapter we present in more detail the background of research question one and two (RQ1 and RQ2), as presented in Chapter 1. To get a better understanding of the RQs, we first give a short presentation of the *Personal Health Record* (PHR) and look at its content and some of its specific utilities. Then we give an overview of the *Electronic Health Record* (EHR) and the parts of it that we find relevant for this project. Afterwards we present incentives for RQ1 and RQ2, respectively, in light of the examined systems. Finally we present some related work, carried out in former research projects.

2.1 The Personal Health Record

In Norway, the research on PHR and the possibility of integrating a PHR system to both existing and new systems is an ongoing process (Brasethvik and Kofod-Petersen, 2006). However, implementations of PHRs do exist, like Indivo. Indivo[1](Riva et al., 2000; Tang et al., 2004) is a PHR system still under development. This project is carried out in cooperation between Children's Hospital Boston, Massachusetts Institute of Technology (MIT) and Harvard Medical School. Indivo's goal is to develop a secure, distributed and patient controlled storage for personal health related information online, but also envisions online communication between the GP and the patient (Tang et al., 2004).

Due to the immaturity of the PHR we present some background information concerning the PHR and what requirements it is thought to support, before we present some of its contents and expected utilities. We do not focus on matters specific to implementation, but rather on the background and expectations that the system is thought to meet, which are meant to specify the RQs of this project.

2.1.1 Background

In the current plan of action from the Norwegian public health service, Te@mwork 2007 (Norwegian Ministry of Social Affairs and Norwegian Ministry of Health, 2004), it is stated that both patients and next of kin should have the opportunity to get more involved in the treatment. Several research projects have found that online communication between the GP and the patient strengthens the patient's possibilities of getting insight into the treatment, among other aspects in favour of this way of communicating (Andreassen et al., 2006; Ball and Lillis, 2000; Stone et al., 2002). Some of the

[1]Former "Personal Internetworked Notary and Guardian" (PING), www.indivohealth.org

reported findings are:

- Ease of communication due to ubiquity and unsynchronised time.
- Easier to talk about personal problems, since sitting face-to-face with the GP is unnecessary.
- After having reported to the GP, the patients felt relaxed.
- Use of more personalized language, making the patients feel closer to their GP.
- Higher degree of specification in patient reports, due to optional amount of time to express themselves.
- More freedom of choice and control.
- More devoted and frequent reporting from patients.

The PHR is an application that supports this kind of communication. Markle Foundation (2003) defines a PHR as the following:

> *An electronic application through which individuals can access, manage and share their health information, and that of others for whom they are authorized, in a private, secure, and confidential environment.*

Crawford (2006) presents a list of different definitions of PHR, where the most basic ones are just a piece of paper, or a stand-alone computer. The PHR that theoretically requires less effort by the patient to populate and maintain is the "Health Bank" definition:

> **Health Bank:** *The "Health Bank" model aggregates data from multiple providers (insurance companies, hospitals, administrators of prescription drugs, labs, and the patient) in a centralized, patient controlled data repository. The health bank is responsible for making the record (or components of the record) available to authorized users, and for giving the patient a mechanism to identify who may access what parts of their record, and under what circumstances.*

This latter definition of PHR is in accordance with the definition presented by Markle Foundation (2003), and will be the definition referred to throughout this report when speaking of the *personal health record (PHR)*. In the following sections we seek to give a more thorough explanation of this application, presenting its content and utility.

2.1.2 Contents

The PHR may contain many types of data. Ideally, the PHR should contain as much relevant data as possible. Also, this data should come from as many relevant sources as possible. The more comprehensive the data is, the more useful the data will be to the patient (Tang et al., 2005). However, as stated by Nilsson et al. (2003), to avoid information overload effort has to be made to find ways to abstract the information necessary for the user. Tang et al. (2005) suggest that the PHR should at least contain health related problems, data about allergies, plans of treatment, family health history, lifestyle, medications, and lab results. Brasethvik and Kofod-Petersen (2006) picture a PHR solution consisting of three parts: One part for treatment plans, one part for patient reports, and one part for document handling.

A knowledge model (Rose, 2006) developed in *Unified Modeling Language* (UML) (Fowler, 2003), presented in Figure 1, supports the tripartition proposed by Brasethvik and Kofod-Petersen (2006). In this model one part is for document handling, another part for plans, and the last part for patient reports, represented by respectively *MyJournal*, *MyPlan*, and *MyReport*.

The patient reports are the focus of the figure, and should make it possible for the patient to report to what extent the treatment is in compliance with what was suggested, and what effects this treatment causes. These effects may be both objective or subjective, as suggested by Tang et al. (2005), and depicted in the figure as *Subjective effect* and *Findings*. The findings are further split into *Observations* and *Measure*. An effect may be linked to the execution of an action, i.e., a *Concrete action*, as depicted in the figure.

The patient should also be able to report if he or she has any hypotheses concerning the treatment. These hypotheses may for instance be what kind of treatment the patient finds best, what kind of treatment that does not work, and other relevant remarks (Rose, 2006).

The plans of the PHR in Figure 1 may further consist of a set of abstract and concrete actions. In addition each plan has a goal connected to each action, and may have a superior goal indicating the reasons for carrying out the plans.

2.1.3 Utility

The PHR is, as mentioned above, an online communication tool between the GP and the patient, and is thus likely to have the same advantages as presented in Section 2.1.1. There are, however, other reasons that advocate the usage of the PHR. An important point is that the patient gets better and more credible knowledge about his or her health, and may be helped quickly by the health service (Tang et al., 2005).

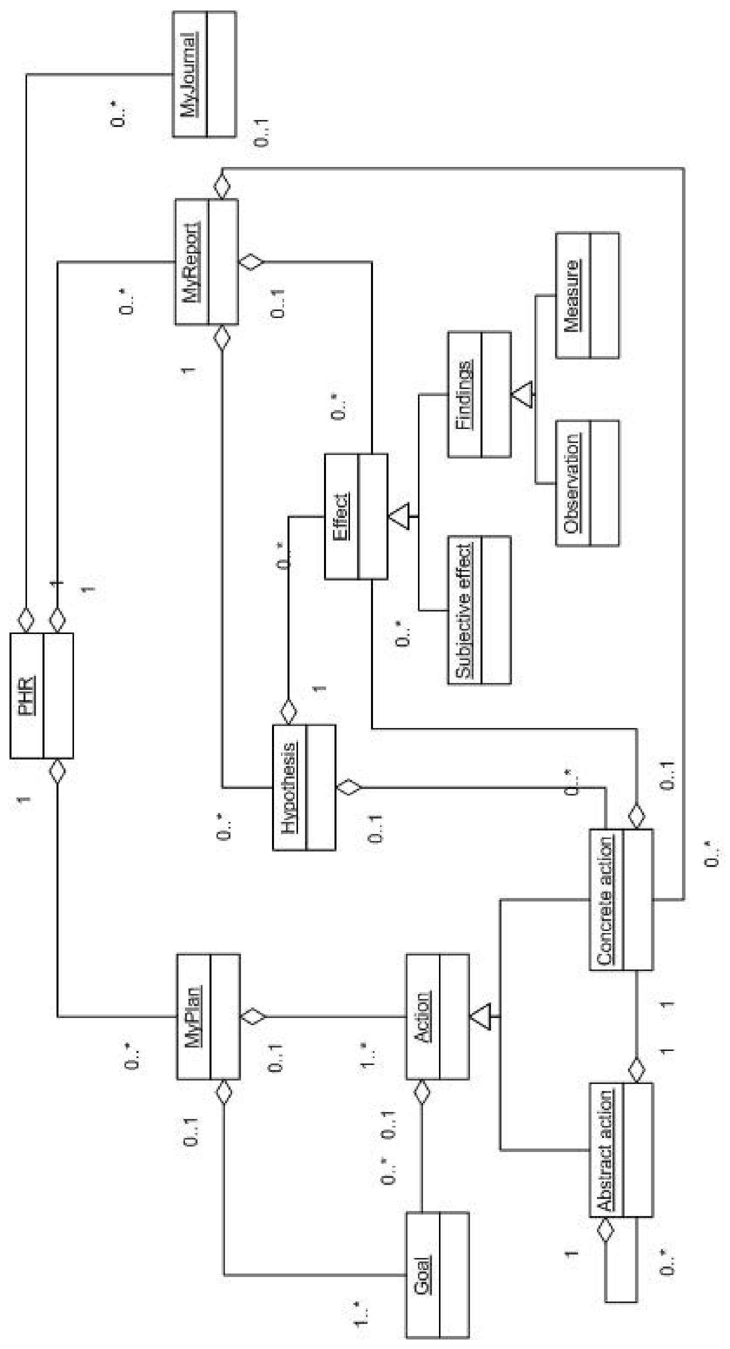


Figure 1: Knowledge model of a PHR. Figure from Rose (2006).

Another point is that communication is continuous with the usage of the PHR, in comparison to previous more episodic communication (Tang et al., 2005). Szolovits et al. (1994) advocate that patient reports may even be used for ordering appointments at a GP's office instead of following a predefined schedule.

Another actor identified to get benefits from the usage of the PHR is the society as a whole, the treatment payer. It is assumed that increased access to results of tests and lab results will decrease the number of duplicate testing, and hence save a lot of money (Walker et al., 2005; Markle Foundation, 2003). Markle Foundation (2003) also has other ideas about what the PHR data may be used for, like looking for pattern in disease- and treatment history, and surveillance of PHRs making it possible to detect outbreaks of diseases at an early stage.

The last actor that will benefit from the usage of the PHR is the GP. For example, if the increased amount of information available is presented to the GP in a proper way, decisions may be based on more data. This will not only make it easier for the GP to make decisions, but the decisions made will also be far more empirically founded. Equally, it will be easier for the GP to include patient thoughts in the EHR, when patient reports may be digitally moved between the PHR and the EHR. The latter point is also likely to far more accurately reproduce the patients thoughts and ideas, as stated above. In the next section we present the EHR and the parts of it that we find relevant to this information exchange.

2.2 Electronic Health Record

The PHR, defined in Section 2.1.1, is managed by the individual. In contrast, the clinician's record of patient encounters, i.e., the EHR, is managed by the clinician him- or herself and/or a health care institution. Bemmel and Musen (1997) define an EHR as:

> *Administrative and medical patient data electronically stored in a consistent way. A computer-based patient record may contain characters, signals, images, and sounds.*

Today, generally all GPs in Norway use EHR systems (Norwegian Ministry of Social Affairs and Norwegian Ministry of Health, 2004), while, as stated in Section 2.1, PHR systems still remain to be fully implemented. Tang et al. (2005) picture a solution, over time, in which information may flow seamlessly among systems used by authorized health professionals, caregivers, and the patient, where the patient is granting the authorization to the respective users. An example of an EHR system, *Profdoc Vision Allmenn*, is presented in Chapter 3.

We have now shortly presented the EHR and how it relates to the PHR. In the coming sections we present the parts of the EHR that we find relevant in the work of the RQs of this project.

2.2.1 Encounter Notes

Encounter notes in the EHR denoting the diagnostic process usually follow a predefined structure (Nilsson et al., 2003). We now explain this structure in conjunction with the *diagnostic process*, presented by Baerheim (2002). In the diagnostic process, the patient first typically explains what his or her problem is to the GP, how the development of the problem has been, and its duration. The information presented by the patient is written in the EHR by the GP. This part of the note is known as the *subjective (S)* part.

Next, the GP comes up with a set of hypotheses of what might have caused the problems presented by the patient, and elects the most probable hypothesis according to former knowledge about the patient and the subjective experience presented by the patient. Then the GP looks for findings or tries to discover characteristics which support the elected hypothesis. If the hypothesis seems to be wrong, the GP will consider other possible hypotheses. In the other case, if the hypothesis seems to be supported by the findings, the GP makes a diagnosis. The findings or characteristics found by the GP are written in the EHR. This part of the note is called the *objective (O)* part of the note. The probable diagnosis is also written in the EHR, and is called the *assessment (A)* part (Baerheim, 2002).

Based on the diagnosis made by the GP, the GP will suggest a plan of treatment. This treatment may for instance be how the patient should behave due to the findings, or for instance writing out a prescription for a drug. The planned treatment is written in the encounter note of the EHR in its own *plan (P)* part (Baerheim, 2002). This note description standard is known as the *SOAP (subjective, objective, assessment, plan)* standard, and is followed by most GPs (Nilsson et al., 2003).

Even though most notes follow this *SOAP* standard, the input format of the patient-doctor communication in most commercial EHR systems is rather unstructured syntactically. This makes it almost impossible to retrieve any specific part of the note, making it difficult to use for both research and clinical practice. However, writing unstructured text is easy, and is the traditional way of documenting patient treatment (Røst et al., 2006b). An example of an EHR system that does not provide any note structure is the *Profdoc Vision Allmenn* EHR system, mentioned above.

2.2.2 ICPC Codes

Brage et al. (1996); The Norwegian Directorate for Health and Social Affairs (2004) present an overview of the coding system International Classification for Primary Care (ICPC). In 1978 the World Health Organization (WHO) took the initiative to implement a new coding system, the ICPC, to strengthen the service of general practice. The diagnostic components in ICPC are based on earlier classification systems.

In Norway ICPC was introduced to general practice in 1986. Initially the use of ICPC codes was limited, but with the introduction of computers in general practice in about 1990, the use increased dramatically. Since 1992 the use of ICPC codes in general practice has been compulsory.

Over the years, some errors and potential improvements to the ICPC system was identified. Hence, a new version of ICPC, ICPC-2, has now replaced the old coding system. This is the coding system which is applied in the data presented in Chapter 3.

ICPC-2 has a biaxial structure where symptoms, procedures and diseases are coded into chapters and components represented respectively by a letter and a two-digit number. In Table 1, the ICPC coding system is presented, where the horizontal axis represents chapters, and the vertical axis represents components. In the figure the ICPC code "T90" is given as an example, which represents a diagnosis for "organ systems". "T90" indicates "Diabetes non-insulin dependent".

	General conditions (A)	Organ systems (14 chapters)	Psychological problems (P)	Social problems (Z)
Symptoms (1-29)				
Processes (30-69)				
Diagnoses (70-99)		Example: *T90*		

Table 1: The ICPC coding system. "T90" is an example of an ICPC diagnosis code. Figure adapted from The Norwegian Directorate for Health and Social Affairs (2004).

2.3 Incentives for RQ1: Data Mining in EHRs

In this section we seek to explain the incentives behind RQ1. There are several degrees of interconnectivity between the PHR and the EHR. Some solutions implement the PHR as a stand-alone application, while others seek to interconnect the PHR to several sources of information (Tang et al., 2005; Crawford, 2006). An example of the latter is the “Health bank” definition of the PHR, described in Section 2.1.1.

The integrated approach, where the PHR and the EHR operate in tandem, can convey more relevant data to the patient. In addition, integrating the PHR with the EHR may utilize the good backup systems that exist today for the EHR systems. Due to the reasons stated above, Tang et al. (2005) reckon that PHRs integrated with the EHR provide much greater benefits than stand-alone PHRs. However, concerning this approach, it is important to note that current Norwegian legislation may impede the dissemination of such systems. We expect, though, that increased Internet security and knowledge about the benefits of information exchange may relax this impeding legislation.

The patient reports of the PHR may have a structure as the one explained in Section 2.1.2, consisting of either subjective or objective effects, or statements of actions and hypotheses of the treatment carried out. On the other hand, the EHR normally has a structure as the one explained above, consisting of a subjective part, an objective part, an assessment part, and a plan part (Nilsson et al., 2003). When exchange of information is to happen between the mentioned systems, we picture a solution where there may for instance be a flow of information between the subjective and objective parts of the EHR and the PHR, respectively.

The idea of information exchange is illustrated in UML in Figure 2. The lower part of the figure is a pruned version of the PHR knowledge model, presented in Figure 1. The upper part of the figure depicts an EHR encounter note, with its belonging subjective, objective, assessment and plan parts. The bold units represent information that may be exchanged, and the illustrative non-UML double arrows in between denote what respective parts of the EHR and the PHR the information exchange may take place. As one may note from the Figure, we picture a solution where the plan part of the PHR resembles the sum of the assessment and plan part of the EHR. This is done due to the nature of the documentation of encounter notes, where the assessment and plan part are often written together, also noted in a Swedish research project by Nilsson et al. (2003). This trend is confirmed by the data that we seek to apply in the work of RQ1, presented in Chapter 3. Hence, we picture a solution where information may be exchanged between *Subjective effect* and *Subjective part*, *Findings* and *Objective part*, and *Plan* and the sum of *Assessment part* and *Plan part*. For simplicity, the latter merging will be referred to as the *Plan*.

However, due to the unstructured nature of most EHR notes, we find it difficult both to extract particular parts of the notes, or make additions to

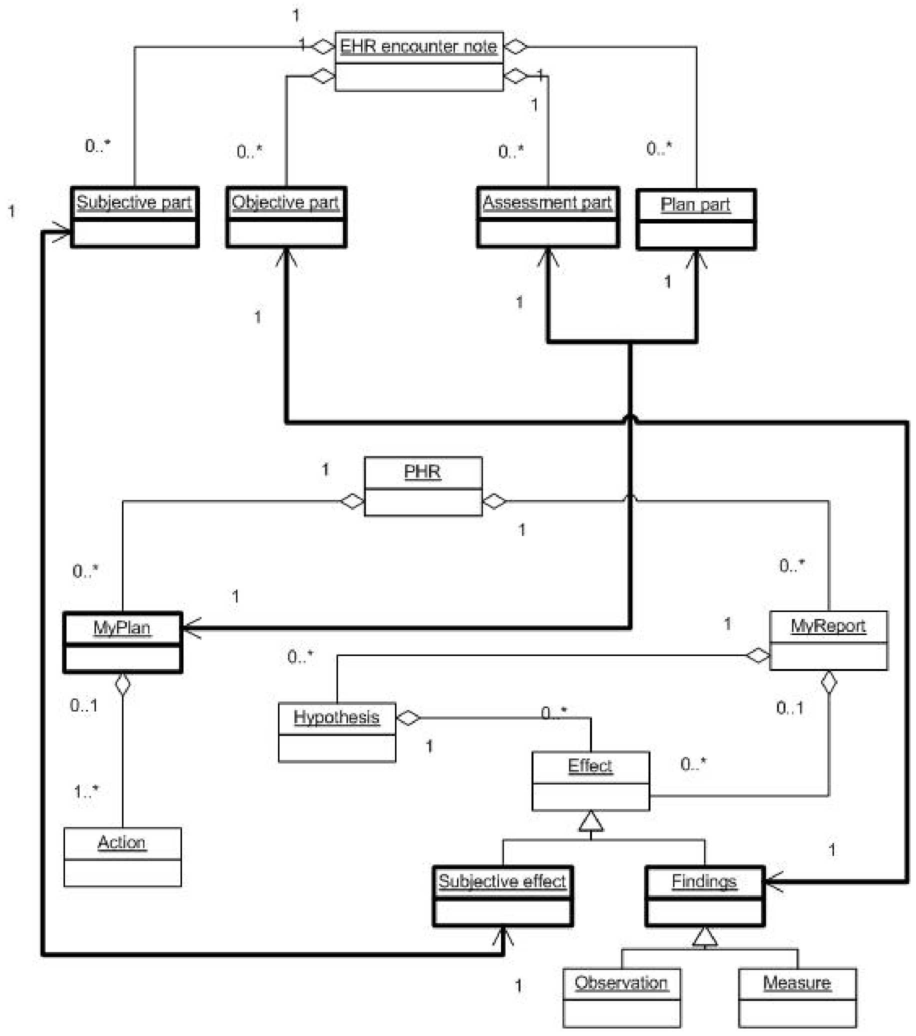


Figure 2: Overview of how we picture the flow of information between the EHR and the PHR. The arrows between the bold parts represent what parts of the respective systems that may be swapped. The lower part of the figure is adapted from Figure 1 by Rose (2006).

particular parts of the notes from a PHR. Hence, to make this exchange of information possible, efforts have to be made to understand the intentions of the different parts of the EHR encounter notes, to be able to tag each part of the note to its respective part. E.g., sentences that describe how the patient feels should be tagged as subjective and measurements should be tagged as objective.

In this project we seek to apply methods from the field of text mining to be able to discern between different parts of the EHR notes. In particular, we seek to apply different techniques to create a good classifier in conjunction with various preprocessing techniques.

As stated in the previous section, there are several advantages to the free-text way of documenting patient treatment. It is likely that some of these advantages apply for patients using PHRs as well. Hence, the text classification methods evaluated in this report, are also likely to be applied on raw text from the PHR.

We have now presented the main incentives for creating a classifier that may classify the sentences in encounter notes according to a partitioning similar to the *SOAP* standard. However, there are also other incentives for carrying out such experiments, which we will see in the section to come.

2.4 Incentives for RQ2: Data Condensation

In this section we seek to explain the incentives behind RQ2. The sickness history for a patient having a particular disease is likely to possess a various number of sentences of subjective, objective, and plan kind. As stated by Nilsson et al. (2003), little or nothing is done with respect to future re-use of the data in most EHR systems. Norwegian Ministry of Social Affairs and Norwegian Ministry of Health (2004) are also concerned about this problem, stating that the increased access to information also makes some of the information less available due to information overload. In this second part of the project we seek to utilize the classifier derived from the work on RQ1, and investigate if the classifier is able to present histories of EHR encounter notes in ways that allow us to detect points of interest and trends, thus seeking to create abstracts of the EHR data.

When performing the experiments associated with RQ2, we assume that the relative distribution of these notes is dependent on the stage of the disease in focus. I.e., we expect that for some parts of the patient history, the frequency of subjective, objective and plan sentences will vary based on the event taking place.

We picture a solution where we may firstly detect special points of interest for individual patients, and secondly see trends when evaluating the average subjective, objective and plan distribution of patients' history over time.

E.g., a disease where the number of subjective sentences is decreasing, may mean that there is little or constant change in how one feels, while the opposite may mean that the patient is going through a lot of changes. The first contribution may be helpful through the daily work of the GP, in an EHR setting, or for a patient, in a PHR setting. The second contribution, however, will not give any information about specific patients, but may rather disclose some patterns concerning the disease in focus. To be able to extract such patient histories related to a disease in particular, we may utilize the ICPC codes, as presented in Section 2.2.2.

In this study we only consider EHR data, but similar studies would be interesting to perform for PHR data as well. In the next section we present some related work that has been carried out.

2.5 Related Work

Several methods of text classification have been applied on medical data. In EHR systems, attempts have been made to classify the EHR encounter note itself, according to its corresponding diagnosis codes. This has been done on both ICPC and ICD diagnosis code systems, with both an algorithmic and a table look-up approach (Røst et al., 2006a,b; Letrilliart et al., 2000; de Freitas Vale et al., 2003). Other studies have extracted diagnoses and procedures from patient discharge summaries (Long, 2006), while others have tried to classify the contents of discharge summaries into different categories (Sibanda et al., 2006). Related work concerning the selection of text classifier algorithms and preprocessing techniques is further presented in Chapter 4.

A limited amount of literature has been found concerning research of the distribution of *SOAP* sentences. Nilsson et al. (2003), however, have investigated the distribution of *SOAP*, focusing on *words* to find ways to aggregate information from EHRs. This work was conducted manually, but may still serve as a reference of comparison to our work. We have not, however, found any literature of experiments applying automatic text classification to find subjective, objective, assessment or plan oriented data to neither PHR nor EHR data.

Due to the fact that no automatic text classification has been executed in order to classify encounter notes in the context of *SOAP*, as far as we are concerned, we have not been able to find any literature that *visualizes* this distribution. However, research has been done on visualizing other condensed parts of the EHR, like the LifeLines application (Plaisant et al., 1996), which is capable of presenting an overview of events in the patient history. The LifeLine application and this research project share the goal of making patient data more available, but the data presented is of different character and hence we acknowledge that techniques from this application has limited value for this project.

2.6 Summary

In this chapter we have presented the PHR, and what benefits the use of it may provide. We have also shortly presented the EHR, and its encounter notes and ICPC codes. Then we have presented the two RQs of this project in more detail, and the incentives behind these. Finally we have presented some related work carried out in former research projects. In the next chapter we present the data that we are to use in this project.

3 Data in Profdoc Vision Allmenn

In this chapter we shortly present the data that we seek to use in this project. First, we give an overview of the EHR application *Profdoc Vision Allmenn*, and then we present some characteristics of data available for RQ1 and RQ2, respectively.

3.1 Profdoc Vision Allmenn

Profdoc Vision Allmenn[2] is an application for GP's offices, and works as an interface to a database with EHR data, containing all aspects of patient treatment. Examples of data units that may be stored in this database may be family health history, encounter notes, lab results, medical certificates, medications, prescriptions, diagnosis codes, and economical issues.

In this project we particularly apply two of the information units stored in the Profdoc Vision Allmenn EHR system: The encounter notes and the ICPC diagnosis codes, described in the previous chapter. At the *Norwegian Centre for Electronic Health Records (NSEP)* we have been given access to a Profdoc Vision Allmenn health record system from a medium sized GP's office in Norway. This system contains data from all consultations between 1992 and 2006. We now look at each of these information units, in light of respectively RQ1 and RQ2.

3.2 The Data

The Profdoc Vision Allmenn database contains data from 11,944 patients, constituting 423,288 encounter notes. However, the sentences of the encounter notes in the database are by default not classified to be neither subjective, objective, or plan sentences. In the sections to come we present the data relevant to RQ1 and RQ2, respectively.

3.2.1 RQ1 Data

To be able to make a text classifier for RQ1, we need preclassified data to train the classifier. Through manual inspection of encounter notes in the EHR database, we find that most encounter notes are written in a manner inadequate for automated class tagging. However, we do find that *one* GP in the EHR database has consistently written his journal notes according to the *SOAP* structure, dedicating one paragraph to each class. However, this GP does not follow the *SOAP* structure in a strict manner. Rather, the GP seems to structure his encounter notes in an *S*, *O*, and *A* and *P* together

[2]http://www.profdoc.com/norge/allmennlege/programvare/

fashion. This is a common way of writing the EHR encounter notes (Nilsson et al., 2003).

The *S*, *O*, and *A* and *P* together structure, provided by this GP, is likely to greatly reduce the effort by letting us apply automatic methods for classification, and later check the assigned class values by manual inspection. In Table 2, a presentation of the distribution of the classified sentences in the data contributed by this GP may be found. From this table we may state that the classes are approximately uniformly distributed, with a slight overrepresentation of the subjective class. The total number of instances is 5,646.

Class	Number	Percentage
S	2,442	42.9 %
O	1,833	32.5 %
P	1,389	24.6 %
Total	5,646	100.0 %

Table 2: Distribution of the classes subjective (*S*), objective (*O*) and, plan (*P*) in the data material.

Nilsson et al. (2003) have also carried out research on the relative distribution of *SOAP* from another collection of encounter notes. However, this work was conducted manually, and hence they had the opportunity to separate the mixtures of assessment and plan information. This study did not focus on *sentences* belonging to the different classes, but rather *words*. Their findings were that 48.4% of the words belonged to the subjective class, 23.5% were objective words, 6.8% were assessment words, and 18.4% were plan words. The relative sum of assessment and plan words is hence 25.2%, yielding a distribution similar to the one found by our automatic classification. The dataset associated to RQ1 is referred to as *D1*.

3.2.2 RQ2 Data

For RQ2 we focus specifically on patient histories containing encounter notes with the ICPC codes of diabetes, i.e., T89 (diabetes insulin dependent) and T90 (diabetes non-insulin dependent). We find that 429 patients have at least *one* encounter note with a pertaining diabetes diagnosis code, while 221 patients have at least ten encounter notes with a pertaining diabetes diagnosis code. In total there are 6,924 encounter notes, containing 33,505 sentences, that have a pertaining diabetes diagnosis code. Encounter notes not associated with a diabetes code are excluded in this dataset. This dataset is in the following referred to as *D2a*.

An almost similar dataset, which we have named *D2b*, differs in only one way from the data in D2a: Encounter notes associated with ICPC codes other than T89 and T90 are also included. As above, 429 patients have at least one diagnosis code of diabetes, but in this dataset there are 336 patients that have ten following encounter notes after the diagnosis code of diabetes first occurs in their patient history. In this dataset there is in total 26,252 encounter notes, containing 119,289 sentences.

3.3 Summary

In this chapter we have briefly presented an EHR application whose data we seek to apply for our research. Table 3 shows an overview of the data presented in this chapter. In the next chapter we present text mining techniques that may be applied on this data.

Name	Sentences	Description
D1	5,646	Automatically tagged data based on the structure provided by GP.
D2a	33,505	Patient history of exclusively encounter notes associated with the diagnosis codes of diabetes, excluding other encounter notes.
D2b	119,289	Patient histories encounter notes associated with the diagnosis codes of diabetes, including other encounter notes.

Table 3: Overview of the datasets we are to use in this project. First column denotes the name of the dataset, the second column denotes the number of sentences present in that dataset, and the third column gives a description of the dataset.

4 Text Mining Techniques

In this chapter we present techniques from the field of *text mining* to use in experiments in the work of RQ1 and RQ2. *Text classification* is a subfield of text mining, and is defined as the activity of assigning predefined classes to new documents based on the likelihood suggested by a training dataset of preclassified instances (Sebastiani, 2002). The classifier may either be evaluated against an own test dataset, or other techniques may be applied. Text classification has gained popularity in recent years due to the increase in availability of digital text and better computer hardware capable of performing classification (Sebastiani, 2005, 2002; Yang and Liu, 1999).

Knowledge engineering, the task of manually defining a set of rules encoding expert knowledge on how to classify documents under given categories, was until the late 1980's the most popular approach to text classification. In recent years, however, the *machine learning* approach has gained popularity. The machine learning approach of text classification is the process of automatically building an automatic text classifier by learning from a set of previously classified documents. The latter approach has several advantages. First, the accuracy achieved is often comparable to that achieved by human experts. Second, since no expertise from neither domain experts nor knowledge engineers is needed to carry out the task, the machine learning approach of text classification contributes considerable savings in terms of expert manpower (Sebastiani, 2002). This project seeks to deal with this latter approach, automatically trying to create a classifier for text classification. Sebastiani (2005) defines text classification formally as:

> *The task of approximating the unknown "target function"* $\Phi : D \times C \rightarrow \{T,F\}$ *(that describes how documents ought to be classified, according to supposedly authoritative expert) by means of a function* $\Phi : D \times C \rightarrow \{T,F\}$ *called the "classifier", where* $C = c_1,\ldots,c_{|C|}$ *is a predefined set of categories and* D *is a (possibly infinite) set of documents. If* Φ $(d_j, c_i) = T$*, then* d_j *is called a "positive example" (or a "member") of* c_i*, while if* Φ $(d_j, c_i) = F$ *it is called a "positive example" of* c_i*.*

Text classification has been applied to many different tasks. Sebastiani (2002) gives an overview of some of them, which we now present:

Document organization: Examples of document organization may be filing newspaper articles under appropriate sections like politics, home, news, or lifestyle.

Text filtering: Text filtering is the classification of a dynamic collection of texts, e.g., an e-mail filter, trained to identify junk-mail, and classifying non-junk mail into classes appropriate for the user.

Automatic metadata indexing: Assignment of keywords or keyphrases to a document, describing its content.

Hierarchical categorization of web pages: Recognizing the contents of web pages and placing the page in a content tree of branches of special semantic meaning.

Language identification: Deciding the language of a text of unknown language.

Author identification: Authors of texts of unknown or disputed authorship may be decided based on text classification techniques.

There are not only a number of applications that text classification may be applied on, but also a number of ways text classification may be performed. Sebastiani (2002) gives an overview of some of the most important ones. For example, there is a distinction between single- and multi-label text classification. *Single-label* text classification is the case where exactly *one* category must be assigned to each document, while *multi-label* text classification is the case where a document may be assigned to several categories. Single-label text classification is often referred to as *binary-classification* in cases where a document must be either assigned to a class or else its complement. Multi-label classification may be achieved with the usage of algorithms for binary classification, but not vice versa.

Another option is whether the text classification is category-pivoted or document-pivoted. *Category-pivoted* text classification is the act of finding all documents that should be filed under a category, while *document-pivoted* text classification is the act of finding all categories that a document may be filed under.

There is also a difference between whether the classification is hard or ranked. In most automatic categorization processes, the "hard" values of *true* or *false* are required, while a *ranked* decision would be of great help to a human expert in charge of taking the final categorization decision.

In the following sections we give a presentation of relevant phases in text classification. An overview of how these phases are related is presented in the UML (Fowler, 2003) activity diagram in Figure 3. The "forks" in the figure denote the logic operator *OR*, while the opposite forks are *joins*. The figure depicts the chronological order of which the text classification operations often take place. However, due to programming feasibility and tools used, some preprocessing techniques, like stemming, may in reality take place *prior* to indexation.

Figure 3 also presents an overview of the sections to come. First we present the indexation of the documents, before we look deeper into the extensive phase of preprocessing. In this report we have chosen to count stemming, the use of n-grams, the removal of stopwords, tf-idf weighting and transformation of all letters to lowercase as methods of preprocessing. These preprocessing techniques are, however, not mandatory in order to perform text classification. When it comes to classifiers, we look at the probabilistic *Complement Naive Bayes (CNB)*, *Support Vector Machines (SVM)*, and the decision tree *C4.5*. Finally we look at ways to evaluate the results obtained by the classifiers.

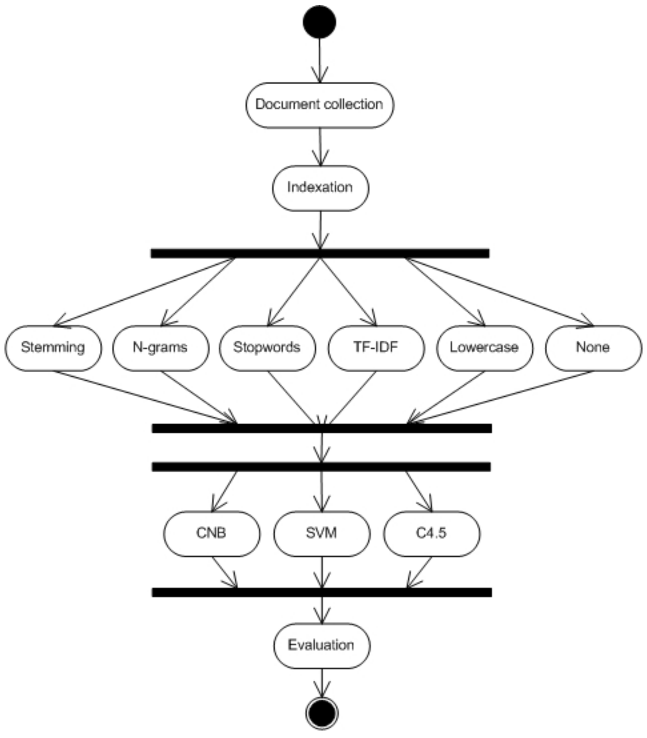


Figure 3: UML activity diagram presenting both the flow of operations in text classification, and the order of the sections to come.

4.1 Document Indexing

A text classification algorithm cannot be used directly on text documents. Documents have to be *indexed*. Indexation of a document is a procedure that maps a text into a compact representation, consisting of the relevant words, i.e., terms, of the document in addition to its belonging *weight*. At the same time as performing indexation, irrelevant tags may be removed from the corpus in order to ease further data management.

An index term may either be identified with a single word in the text or n-grams, explained in Section 4.2.3. The counting of the terms may be referred to as *set of words* or *bag of words* approach, depending on whether they are counted binary or not (Sebastiani, 2002).

The weight of an index term is its relevance to the document compared to the other index terms in the document. In other words, it describes how well that specific index term represents the contents of the document. A document d_j may be represented as a *vector* of weights $\vec{d_j} = <w_{1j},...,w_{|\tau|j}>$, where τ is the set of indexed terms (Sebastiani, 2002; Baeza-Yates and Ribeiro-Netoziri, 1999). The weight of a term may be values like a boolean value indicating whether the word is present in the document or not, or more advanced like the *tf-idf* weighting scheme, presented in Section 4.2.4. An example of the use of indexed terms is shown in Table 4, where sentence 1 to *n* is indexed.

Sentence 1: "This is the first sentence."

Sentence 2: "This is another sentence that we will consider."

Sentence *n*: "This is the last sentence."

Sentence #	1	2	...	n
This	1	1	...	1
is	1	1	...	1
the	1	0	...	1
first	1	0	...	0
sentence	1	1	...	1
another	0	1	...	0
that	0	1	...	0
we	0	1	...	0
will	0	1	...	0
consider	0	1	...	0
last	0	0	...	1

Table 4: Example of a binary indexed document, where "1" denotes presence and "0" denotes absence of a term in the sentence. No preprocessing techniques, like stemming or removal of stopwords, have been applied on the data.

4.2 Preprocessing of Data

In this project we define “preprocessing” to include any action on the data between the document indexation and the text classification. Several experiments report that preprocessing of the data give boost to the classification results (Sebastiani, 2002). We now present some of the preprocessing techniques that we seek to use in this project, except from turning all letters to lowercase, due to its triviality.

4.2.1 Stemming

Stemming is a technique to reduce words to their grammatical roots. A *stem* is the part of a word which is left after having removed its affixes (i.e., prefixes and suffixes). For instance, the words “connected”, “connecting”, “connection”, and “connections” all share the stem “connect” (Baeza-Yates and Ribeiro-Netoziri, 1999).

There are several ways of finding the stem of the words, we will here look at two of them (Frakes and Baeza-Yates, 1992):

Algorithmic affix removal stemming: This kind of stemming is intuitive, simple, and can be implemented efficiently. Most important is the removal of suffixes, because most word variants are created by adding different suffixes (as seen above). An example of a popular stemming algorithm is the *Porter* stemmer (Pomikálek and Rehurek, 2007).

Table lookup: Table lookup consists of looking up the word in a dictionary, and then returning the root of the word. This kind of stemming may also deal with *word mutations*, which are very common in Norwegian, such as the word “book”: “bok”, “boken”, “bøker, and “bøkene”.

Performing stemming has especially two benefits. First, when words are represented by their stems, variants are reduced to the same common concept. Thus, one may be able to see new relations between words. Second, reducing the variants of the words reduces the size of the indexing structure because the number of distinct index terms, shown in Table 4, is reduced (Baeza-Yates and Ribeiro-Netoziri, 1999; Sebastiani, 2002).

4.2.2 Stopwords

Stopwords are words which occur frequently in the text and are topic neutral words such as articles, prepositions or conjunctions (Sebastiani, 2002; Baeza-Yates and Ribeiro-Netoziri, 1999). The high frequency of stopwords makes them unsuitable for discriminating between classes, and removal of these words also reduces the size of the indexed document. These words are almost always removed (Sebastiani, 2002). A list of Norwegian stopwords may be found at the Snowball website[3], reproduced in Table 25 in the Appendix.

4.2.3 n-Grams

When only considering single words, we will miss some contextual information (Hersh et al., 1998). A solution to this may be to apply n-grams. An *n-gram* is a word sequence of length *n*. In text classification one can add the most frequent n-grams to the list of index terms. Producing n-grams for values of *n* up to three has been reported to improve results in text classification (Johannes, 1998; Mladenic and Globelnik, 1998), and has also been used in other text classification experiments (Røst et al., 2006a,b; Sibanda et al., 2006).

4.2.4 Term Weighting

The term-weighting scheme of *tf-idf* is by far the most popular one (Sebastiani, 2002), and all best term-weighting schemes use weights which are given by a variation of this formula (Baeza-Yates and Ribeiro-Netoziri, 1999). The tf-idf formula both describes how well the term describes the contents of the document itself, and takes into account whether it is a discriminating term or a term that occurs in the majority of the documents.

Baeza-Yates and Ribeiro-Netoziri (1999) give an overview of tf-idf weighting. The *term frequency (tf) factor* is the factor that measures how well the term describes the document it occurs in. For term k_i in the document d_j, this factor is given by

$$f_{i,j} = \frac{freq_{i,j}}{max_l freq_{l,j}}, \tag{1}$$

where $freq_{i,j}$ is the frequency of term k_i in document d_j. The $max_l freq_{l,j}$ factor is the maximum number of times a term occurs in a document d_j.

The *inverse document frequency (idf) factor* is the factor that takes into account the discriminative power of the term considering the other documents in the collection. This factor is given by

$$idf_i = log\frac{N}{n_i}, \tag{2}$$

where N is the total number of documents in the collection, and n_i is the

[3]http://snowball.tartarus.org/index.php

number of documents in which the index term k$_i$ appears. We can thus calculate tf-idf by

$$w_{i,j} = f_{i,j} \cdot idf_i = \frac{freq_{i,j}}{max_l freq_{l,j}} \cdot log \frac{N}{n_i}. \tag{3}$$

4.2.5 Overview

One feature of many preprocessing techniques is their effect of reducing the dimensionality of the dataset. *Dimensionality reduction* is the act of reducing the dimensionality of the vector space, i.e., removing terms. There are several reasons that high dimensionality of the term space in a text classification setting is problematic. Firstly, high dimensionality of the term space reduces the efficiency of the classifier algorithm. Secondly, reducing the dimensionality prevents overfitting. *Overfitting* of classifiers is the act of making the classifier too fit for the training data, making it perform poorer on the test data, but good at re-classifying the data it has been trained on.

Of the techniques to be mentioned, there are several that may be thought of as dimensionality reduction techniques, reducing the dimensionality of the vector space. One example may be the act of turning all letters into lowercase, hence making letters of words that are written both in upper- and lowercase correspond. E.g., if the word “arm” is present in the classifier, we get a “match” with the word “Arm”, that may exist in a sentence to be classified. However, dimensionality reduction must be done with care because one always risks to remove potentially useful information (Sebastiani, 2002).

Techniques like the addition of n-grams, expands the set of attributes. We are, however, only interested in the most discriminative terms that are created by this latter method. In addition, many other less useful terms may be present in the set of attributes. Hence, one may apply other techniques to further reduce the dimensionality.

Sebastiani (2002) presents a number of other techniques to reduce the dimensionality of the vector space. Among these are methods that reduce the dimensionality by term selection based on document frequency, or more complex techniques like *information gain*, *chi-square*, *mutual information*, among others (Pomikálek and Rehurek, 2007; Sebastiani, 2002). Dimensionality reduction may also be achieved by term extraction, achieved with techniques like *term clustering* or *latent semantic indexing*. However, it is still an open issue which of the feature selecting methods that is the best (Pomikálek and Rehurek, 2007). Among these methods, performing the selection of features based on *document frequency* is both simple and effective (Sebastiani, 2002). In the next section we present some popular algorithms for use in text classification.

4.3 Text Classification Algorithms

Several classification algorithms are suitable for text classification (Witten and Frank, 2000; Sebastiani, 2002; Joachims, 1998; Sebastiani, 2005). The data that we would like to classify, presented in Chapter 3, have multiple classes (i.e., more than two classes), and we must hence choose algorithms according to this.

We here present three algorithms that we seek to apply for our classification tasks: Naive Bayes, Support Vector Machines, and C4.5. These algorithms are from three different classifier families; probabilistic, vector machines, and decision trees (Sebastiani, 2002), and have all given satisfactory results in earlier experiments (Joachims, 1998; Dumais and Chen, 2000; Røst et al., 2006b; Rennie et al., 2003). In addition, these algorithms all possess different features that we find important in this project. Naive Bayes is a very fast algorithm, SVM has been reported to outperform all other algorithms in many applications, and the C4.5 decision tree makes it easy to visualize the classification process, and has also been reported to perform satisfactorily.

4.3.1 Probabalistic Classification

The *Naive Bayes (NB)* classifier is widely used for text classification (Han, 2006; Jain et al., 2004; Joachims, 1998; Pomikálek and Rehurek, 2007). NB uses the assumption that all attributes of the examples are independent of each other. In spite of this assumption, which may be false in many real world settings, NB often gives a satisfactory result. There are many benefits with this classifier such as its low computational cost, its relatively low memory consumption, and its ability to handle heterogeneous features and multiple classes. Most important, however, is that it often yields a very high accuracy, compared to other good classifiers (Witten and Frank, 2000; Pavlov et al., 2004).

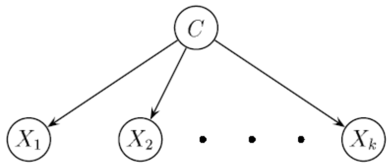


Figure 4: The predictive attributes (X_1, X_2,...X_k) are conditionally independent given the class attribute (C). Figure from (John and Langley, 1995).

What makes NB so efficient is the fact that attributes may be learned separately, and this simplifies learning. This is advantageous especially in settings where the number of attributes is large (McCallum and Nigam, 1998).

This is often the case in text classification, where the attributes are terms in the instances to be classified.

The idea of NB is to apply Bayes' rule to compute the probability of each class given the vector of observed values for the predicted attributes. McCallum and Nigam (1998) present an overview of how classification in NB is performed. Say, if we are to classify a test case x, then we compute the probability of each class c,

$$p(C = c|X = x) = \frac{p(C = c)p(X = x|C = c)}{p(X = x)}. \tag{4}$$

In this equation X=x represents the event that $X_1=x_1 \wedge X_2=x_2 \wedge \ldots X_k=x_k$. Since attributes are assumed to be conditionally independent and the event is simply a conjunction of attribute value assignments, we get

$$p(X = x|C = c) = p(\bigwedge_i X_i = x_i|C = c), \tag{5}$$

which is the probability that a term x belongs to a class c.

The original NB algorithm takes a vector of binary attributes, indicating the presence of words. When calculating the probability of a document, the probabilities of all the present and non-present attribute values are multiplied. Good examples of this algorithm may be found in Han (2006).

In another algorithm, *Naive Bayes Multinomial* (NBM) (McCallum and Nigam, 1998), the number of the occurrences of each word in the document is captured. In this case the probability of a document is calculated by multiplying the probability of the words that occur. When comparing NB and NBM one has found that NB outperforms NBM on small vocabulary sizes, but is outperformed by NBM when vocabulary sizes are large (McCallum and Nigam, 1998).

Rennie et al. (2003) suggest several techniques to improve NB and NBM further. Their version of the NB algorithm is known as *Naive Bayes Complement* (NBC). Firstly, they consider the problem that when one class has more training examples than another, NB will select poor weights, due to shrinkage of weights for classes with few training samples. This is solved by Rennie et al. (2003) by introducing a "complement class" formulation of NB. Secondly, they suggest normalizing classification weights to keep classes with more dependencies from dominating. This is due to the fact that the magnitude of the weights for classes with strong word dependencies are larger than for classes with weak word dependencies. Thirdly, they state that NBM does not model text well and present a simple transform that enables NB to emulate a power law distribution that is more similar to distributions of real term frequency.

4.3.2 Support Vector Machines

Support Vector Machines (SVM) was first introduced by Vapnik (1995), and was later introduced to text classification by Joachims (1998). The algorithm has since been a popular algorithm for text classification tasks (Røst et al., 2006a,b; Sibanda et al., 2006; Pomikálek and Rehurek, 2007), and has been reported to be both accurate and less prone to overfitting than other classifiers. However, the training time tends to be slow even for smaller datasets (Han, 2006).

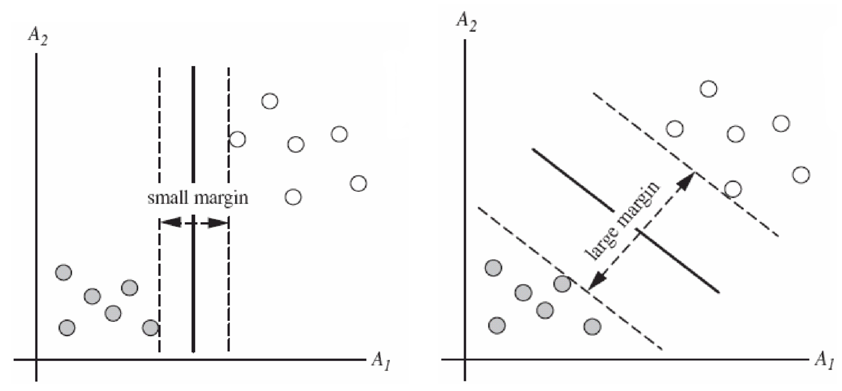
The idea of SVM, explained by Han (2006), is to find all the decision surfaces that best separate positive from negative training examples in a dimension. The decision surface should be at the maximum distance between the closest training examples from the classes. The training examples that determine the best decision surface are called *support vectors*. The idea can be best understood in a setting where all dimensions of the vectors are linearly separable, as shown by the straight line in the two-dimensional data in Figure 5. Figure 5(a) presents a bad separating surface, and Figure 5(b) presents the best possible separating surface. However, if the data is three-dimensional, we would be looking for the best separating *plane*. Similarly we would be looking for the best separating *hyperplane* in a multidimensional dataset.

In the case of linear separable vectors, the classifier of n dimensional vectors will consist of n-1 decision surfaces. In non-linear hypothesis spaces, as depicted in Figure 6, decision surfaces are created by separating the dimension into multiple dimensions so that each new dimension is linearly separable.

There are several *kernels* that may be used in the computation of SVM, like the *linear*, the *polynomial*, or the *radial basic function (RBF)*. However, the resulting accuracy is generally not very dependant on the choice of kernel (Han, 2006).

4.3.3 Decision Tree Classifiers

C4.5 (Quinlan, 1993) is a successor of the *ID3* decision tree algorithm, and is also widely used in text classification applications (Jain et al., 2004; Joachims, 1998; Pomikálek and Rehurek, 2007). In C4.5, the tree is constructed in a top-down recursive manner, where the training set is recursively partitioned into smaller subsets as the tree is being built (Han, 2006). Decision trees are generally easy to understand and interpret, and give a clearer picture of the decisions made, in comparison to other classifiers. An example of a decision tree may be found in Figure 7. If we were to classify a given sentence we could start at the very top of the tree, and move down the tree according to whether the current word is present or not in the sentence. At



(a) A bad separating decision surface. (b) The best possible separating decision surface.

Figure 5: A bad and the best possible separating decision surface is indicated by the lines, as the minimum distance to any training example is at the maximum. Figure from Han (2006).

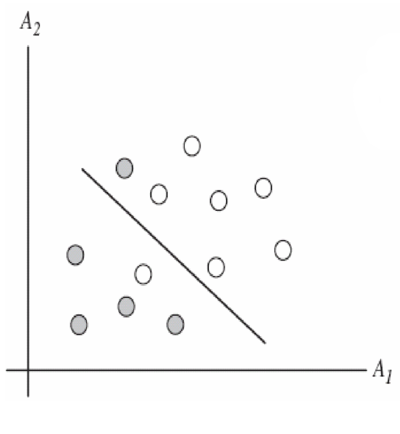


Figure 6: Example of a non-linear space. Figure from Han (2006).

the leaf nodes, at the very bottom of the tree, the class of the sentence is decided.

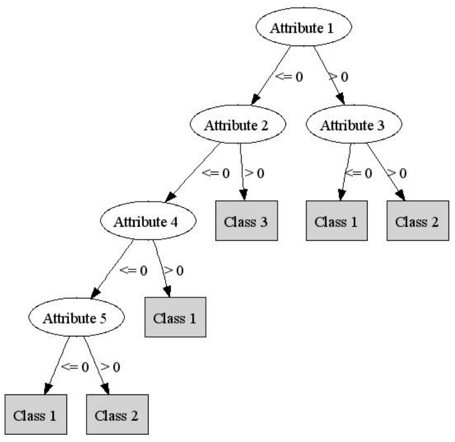


Figure 7: An example of a decision tree of five attributes and three classes, where "<= 0" means that the attribute is not present, and "> 0" means the opposite. The decision tree is manually created in *.dot* code in Graphviz.

At each node of the tree, an *attribute selection method* is applied to select the attribute which best separates a given data partition of class-labeled training data into individual classes. The ideal partition is the one that makes each partition *pure*, in the sense that all tuples of a partition belong to the same class. One such attribute selection method is the *information gain* method, where the attribute that minimizes the information needed to classify the tuples in the resulting partitions is selected. This approach guarantees that a simple tree is found (Han, 2006).

Gain ratio (Quinlan, 1993), which is applied by C4.5, is a successor of information gain. A problem with information gain is that it prefers to select attributes having a large number of values, hence making a large number of pure small partitions. Gain ratio, on the other hand, solves this problem by taking the *split information* into consideration as well, and takes into account the total number of tuples in the training data when considering the number of tuples having a special outcome (Han, 2006; Quinlan, 1993).

In the derived tree, many of the branches will reflect anomalies in the training data, and the classifier may be overfitted (Han, 2006), as explained in Section 4.2. Hence, pruning techniques are often applied to reduce overfitting. In C4.5 a technique known as *pessimistic pruning* is applied, which evaluates how the training data behaves on the unpruned tree. Since the unpruned tree is often biased to the training data, a penalty is added, and the branches performing poorest, e.g., measuring the accuracy, are removed (Han, 2006). In the next section we present some techniques to evaluate a classifier.

4.4 Classifier Evaluation and Comparison

In this section we first present an alternative way of evaluating a classifier without using an own predefined test set, as described above. Then we present several measures to evaluate a classifier, before we finally present statistical ways to compare the results of the classifiers, and see whether one result differs *significantly* from another.

4.4.1 k-Fold Cross Validation

It is often the case that there is only a limited amount of data available. Then the solution is to reuse the data to get a high number of samples. One technique that makes this possible is the *k-fold cross validation*. When applying this technique, the data is split into k equal sized parts and then the classification algorithm is trained on all data but one part for each part (Sebastiani, 2002; Salzberg, 1997). Hence, the dataset is respectively split into a *training* and a *test* set for each iteration. Setting $k = 10$ has proved reasonable in most situations.

Finally, after k rounds, the results of each of the k rounds are averaged or combined in some fashion. This result represents the result of the classification algorithm on the given dataset (Sebastiani, 2002; Salzberg, 1997).

4.4.2 Classifier Evaluation

Sebastiani (2005, 2002) presents different measures to evaluate the success of a classifier. We now look at those presented by Sebastiani (2005):

Training efficiency: Average time to build a classifier from a given corpus.

Classification efficiency: Average time required to classify a document by means of the classifier.

Effectiveness: Average correctness of the classifier's behaviour.

When *research* on text classification is performed, effectiveness is often emphasized because of its adequacy for classifier comparison, due to the fact that efficiency depends on too volatile parameters like hardware and software platforms. However, in text classification *applications* all three evaluation strategies are considered important, but effectiveness tends to be the primary criterion in such contexts too (Sebastiani, 2005).

The effectiveness must be calculated differently for single- and multiple-labeled text classification. When considering single-labeled text classification, the percentage of correct classification decisions can be calculated. This is called calculating the *accuracy*. However, in a multiple-label text classification setting the situation is different. In this case categories tend to be *unbalanced*, meaning that one category contains far more members than the other.

In the multiple-label text classification setting, one often considers another measure, a combination of the precision and the recall. *Precision* (π) is defined as the probability that if a random document is classified under a category, this decision is correct. On the other hand, *recall* (ρ) is defined as the probability that, if a random document ought to be classified under a category, this decision is made (Sebastiani, 2002). Precision and recall are calculated using the terms *true positive (TP)*, which is the number of documents that are classified under a category correctly, and *true negative (TN)*, *false positive (FP)*, and *false negative (TN)*, which are defined similarly. Table 5 gives an overview of the mentioned terms.

		Expert judgments	
		Yes	**No**
Classifier judgments	**Yes**	TP	FP
	No	FN	TN

Table 5: The global contingency table. Table adapted from Sebastiani (2002).

There are, however, *two* ways precision and recall may be calculated when the effectiveness is calculated for several categories. Sebastiani (2002, 2005) presents an overview of these calculations. When performing *microaveraging*, categories (c) count proportionally to the number of their positive training examples and is calculated as follows:

$$\pi = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} TP_i + FP_i} \tag{6}$$

$$\rho = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} TP_i + FN_i} \tag{7}$$

Macroaveraging, on the other hand, assumes that all categories count the same, and is often the method of choice in text classification. Macroaveraging is calculated as follows:

$$\pi = \frac{\sum_{i=1}^{|C|} \pi_i}{|C|} = \frac{\sum_{i=1}^{|C|} \frac{TP_i}{TP_i+FP_i}}{|C|} \tag{8}$$

$$\rho = \frac{\sum_{i=1}^{|C|} \rho_i}{|C|} = \frac{\sum_{i=1}^{|C|} \frac{TP_i}{TP_i+FN_i}}{|C|} \tag{9}$$

It is important to look at combinations of recall and precision because most classifiers can be tuned to emphasize one of these measures at the expense of the other. A popular way to combine them is by use of the function

$$F_\beta = \frac{(\beta^2 + 1)\pi\rho}{\beta^2\pi + \rho}, \tag{10}$$

where $0 \leq \beta \leq \infty$. Usually β is set to "1", and thus we get

$$F_1 = \frac{2\pi\rho}{\pi + \rho}, \tag{11}$$

which is called the *harmonic mean* of precision and recall, or simply the *F measure*. Precision, recall, and F measure are often used as measures to evaluate a classifier (Sibanda et al., 2006). However, in cases where the classes are not too unbalanced, like when deciding whether a web page should be filed under the category *NuclearWasteDisposal*, accuracy is a good measure. This is due to the *trivial rejector*, i.e., assigning each instance to the most popular class, yielding a high accuracy (Sebastiani, 2005).

4.4.3 Cross-Classifier Comparison

There are many ways to statistically test whether one text classification algorithm performs better than another. One solution is to use the *sign test* (Yang and Liu, 1999; Wadsworth, 1990) for both micro and macro testing. The *micro level significance test* is dominated by the results of common categories, while the *macro level significance test* is dominated by the results of rare categories. The *micro level significant sign* test is based on the binary decisions of every document in the classification. This test has the following notation (Yang and Liu, 1999):

N: Number of binary decisions by each system.

a_i: Binary value that measures the success for a system A on the ith decision, where $i = 1, 2, ..., N$.

b_i: Binary value that measures the success for a system B on the ith decision.

n: Number of times that a_i and b_i differ.

k: Number of times that a_i is larger than b_i.

The *macro level significant sign test* may use paired F_1 or accuracy values for individual categories. It is computed like the *micro* level significant sign test, and has the following notation (Yang and Liu, 1999):

M: Number of unique categories.

a_i: F_1 score of system A on the ith category, where $i = 1, 2, ..., M$.

b_i: F_1 score of system B on the ith category.

n: Number of times that a_i and b_i differ.

k: Number of times that a_i is larger than b_i.

The *null hypothesis*, H_0, of these tests is $k = 0.5n$. This means that half of the observed a_i should be larger than b_i. The alternative hypothesis, H_1, is $k > 0.5n$, meaning that more than half of the observed a_i should be larger than b_i. With a confidence level $\alpha = 0.05$ of a *1-sided* sign test, we are satisfied if there is a 95% probability of an event to occur or not. I.e., the null hypothesis, H_0, is rejected for P values < 0.05, while for P values > 0.05, H_0 cannot be rejected (Yang and Liu, 1999; Wadsworth, 1990).

For large values of n, which is most likely to be the case when observing the corpus presented in Chapter 3, the *P value* can be approximately computed using the standard normal distribution for

$$Z = \frac{k - 0.5n}{0.5\sqrt{n}}, \tag{12}$$

which is called the *z score*. To find the P value from this score, a standard normal distribution table must be used. This table may be found in books of statistics, but is not reproduced in this report. For smaller values of n and k, one may use a binomial distribution table directly. Table 24, in the Appendix, shows an example of such a table where $n = 10$ and $\theta = 0.5$. The latter means that, given H_0, there is no difference between the observations.

4.5 Summary

In this chapter we have presented several techniques that may be considered important to perform text classification. More specifically we have looked at techniques of indexation, preprocessing, the algorithms themselves, and evaluation techniques. In the next chapter we present tools that may facilitate the text classification phases.

5 Text Mining Tools

There exist several tools that ease the process of text mining and information presentation. These tools make both the mining itself more feasible, due to the fact that algorithms do not have to be implemented, and facilitate the extraction of graphs and trees. We here shortly present some tools that we consider important in this project.

5.1 Weka

Waikato Environment for Knowledge Analysis (Weka)[4] is an open source toolkit, consisting of a collection of state-of-the-art machine learning algorithms for data mining, written in Java[5]. It provides support for several phases of data mining, including preprocessing, the mining itself, and the visualization of the input data and the result of learning. The toolkit possesses algorithms for classification, clustering, extraction of association rules, and attribute selection, where especially the classifying part is extensively developed.

The toolkit has both a command line interface which makes scripting of research possible and a graphical user interface (GUI) (Witten and Frank, 2000). The standard Weka file format is *arff*[6].

We think that the fact that Weka is an open source programme is especially important, making it possible to both tweak implemented methods, or create our own, contributing to the Weka project. Use of the Weka packages has obtained good results in several text classification experiments (Jain et al., 2004; da Silva et al., 2004; Pomikálek and Rehurek, 2007). There are, however, also other packages of code that one may apply through the Weka package structure. One of these packages is the *LibSVM*[7] (Fan et al., 2005) package.

In Table 6, Weka classes found relevant to this project are presented. The descriptions of the classes are extracted from Witten and Frank (2000). In this table we have included the entire package path of the classes, while in the following chapters we will only refer to the classes by their names.

[4]http://www.cs.waikato.ac.nz/ml/weka/

[5]http://java.sun.com/

[6]Example of fictitious training data in this file format may be found in Figure 24 in the Appendix.

[7]http://www.csie.ntu.edu.tw˜cjlin/libsvm

Class	Description
weka.classifiers.bayes **.ComplementNaiveBayes**	Builds a Complement Naive Bayes classifier.
weka.classifiers.meta **.GridSearch**	Optimises classifier parameters.
weka.classifiers.trees **.J48**	C4.5 decision tree learner (implements C4.5 revision 8)
*weka.core.***Stopwords**	Removes a set of predefined English stopwords.
weka.filters.instance **.RemoveRange**	Removes a range of instances from a dataset.
weka.filters.supervised .attribute. **AddClassification**	Classifies a dataset and adds the new classification to the same set based on a classifier model.
weka.filters.unsupervised **.StringToWordVector**	Converts a string attribute to a vector that represents word occurrence frequencies with several options.
*weka.filters.unsupervised .instances.***Randomize**	Randomises the order of instances in a dataset.

Table 6: Weka classes relevant in this project. The path shows where the class, whose name is written in boldface, is located in the Weka package structure. The descriptions of the classes are extracted from Witten and Frank (2000).

5.2 NORKOMPLEKS

NORKOMPLEKS[8] (Nordgård, 1996) is a Norwegian computational dictionary used in computational linguistics applications. This dictionary contains conjugations of Norwegian words, covering all linguistically legal forms of words. Table 7 gives an example of a pattern in NORKOMPLEKS for the Norwegian verb “å suge”, which in English has the meanings “to suck”, “to absorbe”, “to sap” or “to exploit”. Another example may be the conjugation “the sun”, which can be written both as “solen” and “sola” in Norwegian.

Pattern	**Infinitive**	**Imperative**	**Present**	**Past**	**Past participle**	**Present participle**
1	suge	sug	suger	suget	suget	sugende
2	suge	sug	suger	sugde	sugd	sugende
3	suge	sug	suger	saug	sugd	sugende
4	suge	sug	suger	suga	suga	sugende

Table 7: Conjugation in NORKOMPLEKS. The table is adapted from Nordgård (1996).

Stemming (see Section 4.2.1) is an integral part of the preprocessing steps of text classification. The idea of stemming is to reduce the word to its “stem”, and may be accomplished by the use of this tool. If we look again at Table 7, we see that all forms of the word “suge” (i.e., “suge”, “sug”, “suger”, “suget”, “sugde”, “saug”, “suga”, “sugd” and “sugende”) would be represented in infinitive as “suge”, i.e., the “stem” of the word. This would not likely be the case when using an algorithmic stemmer, where for example the mutated word “saug” would not result in the stem “suge”. Mutation of words is quite normal in Norwegian, and we find it plausible that ignoring these words would depreciate the result. Due to the reasons mentioned above we assume that applying NORKOMPLEKS for stemming will produce the most satisfactory result.

5.3 Visualization Tools

SigmaPlot[9] is a graph tool. These graphs are of high quality, and may be extracted from *Microsoft Excel* spreadsheets. The graphs used to present the results in the subsequent chapters are constructed by use of this tool.

Graphviz[10] is another tool that makes visualization of data possible. This tool enables the creation of decision trees from *.dot* code.

[8]Norwegian acronym: NORsk KOMPutasjonelt LEKSikon
[9]http://www.systat.com/products/sigmaplot/
[10]http://graphviz.org/

5.4 Computer Specifications

When executing operations in the field of text mining, the computing power available is of utmost importance. The increase of computing power available during the last years has been a prerequisite for recent contributions in this field (Sebastiani, 2002). However, through our work we find that the computing power is still of importance, and determine which operations that are feasible or not. The specifications of the computer used in our experiments are presented in Table 8.

Feature	Value
Processor	AMD Athlon(tm) 64 Processor 3500+, 1000 MHz, 512 KB cache
Memory	4 GB
Operating System	Fedora Core 6, 2.6.16 kernel

Table 8: Specifications of experiment computer.

5.5 Summary

In this chapter we have presented various tools that we assume both ease the process of text mining and the visualization of the results. In the following chapters we present the plan, the results, and the evaluation of the results obtained during our research on RQ1 and RQ2.

6 Text Classification Experiment Plan (RQ1)

This chapter presents the plan of our experiments for the text classification performed in the work of research question one (RQ1). More specifically we intend to evaluate different classifiers on different amounts of both attributes and data. This plan includes both *what* we intend to do, and *how* we intend to carry out the experiments. Throughout this chapter we refer to various Weka classes that we seek to use. An overview of these classes may be found in Figure 6 in Chapter 5. In this part of the project we only consider the D1 dataset, presented in Chapter 3.

The plan presented will to a large extent follow the same structure as Chapter 4. However, we only seek to cover the topics that we find important to obtain the raw results that will be presented in the next chapter, while the use of the classifier evaluation and comparison techniques will be presented in Chapter 7. Hence, in this chapter we seek to present the plan of document indexation, text preprocessing, and the text classification itself.

6.1 Document Collection

To be able to train and test the classifier we need preclassified data. In the experiments to be described we apply the dataset named D1, presented in Chapter 3.

In Figure 25 in the Appendix, methods to retrieve the different datasets are presented. Before performing automatic tagging of each sentence, the paragraphs of the notes are inspected manually to confirm that they follow the *S*, *O*, and *A* and *P* together structure.

Subsequently we automatically tag the different paragraphs of the retrieved data, D1, so that the paragraphs are assigned to their respective classes. An example of a classified note is presented in Table 9. The Python source code for automatic classification is given in Figure 26 in the Appendix. Afterwards we split the paragraphs into sentences marked with a class-tag corresponding to the inherent paragraph, as shown in Table 10.

Eventually, after sentence tagging, each sentence's class-tag is approved by manual inspection to validate the process. The distribution of the *S*, *O*, and *P* classes of this classification is presented in Figure 2 in Chapter 3.

The retrieved data is then processed: All punctuation, sentences that consist only of numbers, and "windows newline" are removed to ease both further classification and processing. The source code of these operations is shown in Figure 27 in the Appendix. However, in this figure we also replace all single digits by the letter "N". This was not performed in the work of RQ1, because the sentences contained relatively little numbers, but was of importance

Class	Paragraph p_i	Sentences s_j
S	p_1	s_1, s_2, s_3
O	p_2	s_4, s_5
P	p_3	s_6, s_7, s_7

Table 9: Presentation of the class-tagged paragraphs and inherent sentences in a note in the EHR record. The classes are subjective (S), objective (O), and plan (P). The denotation of the paragraphs and sentences are respectively p_i, where $1 \leq i \leq 3$ and s_j, where $1 \leq j \leq 7$.

Class	Sentence s_i	Class	Sentence s_j
S	s_1	O	s_5
S	s_2	P	s_6
S	s_3	P	s_7
O	s_4	P	s_8

Table 10: Class-tagged sentences extracted from the note presented in Table 9. The denotation is the same as in Table 9.

when dealing with the diabetes data in the work of RQ2.

Manual inspection of the EHR data reveals that there are many spelling errors. Spelling errors may degrade the result, since word recognition will decrease. However, tools exist that may compare each word to words in a dictionary, i.e., a medical dictionary, and find the most probable word if an exact match is not found. However, GPs apply many abbreviations that are typically not present in such dictionaries. Hence, words that are not present in the dictionary, but still appear a number of times in the corpus should be included like any other word. However, the application of such techniques is outside the scope of this project.

6.2 Indexation and Preprocessing of Data

In this section we look at how we index the dataset and how we apply different preprocessing techniques in the work on RQ1. *Document indexing* may be considered a prerequisite for text classification, as described in Chapter 4. To be able to apply classification algorithms on the corpus, we make an *indexed file*, consisting of each word and its belonging weights by applying Weka functions. For both the indexation and selection of the attributes with the highest weight we apply the *StringToWordVector* class. We now present an overview of what preprocessing techniques we wish to apply, and how we may perform them.

Stemming: *NORKOMPLEKS*, explained in Section 5.2, will be applied to perform *table lookup stemming*. We find it best to perform this prior to indexation, since this computational dictionary has no interface to Weka, and is thus performed during classification, as shown in Figure 26 in the Appendix.

Lowercase: Transforming all letters to lowercase is a trivial transition that may be performed by a single method call in Python prior to indexation.

Stopwords: Through manual inspection we notice that there are many word in the corpus that may be considered "stopwords". Weka does have the opportunity to add additional lists of stopwords, but it seems to malfunction. Hence, the list of stopwords may be manually added to Weka in the *Stopwords* class. A list of the Norwegian stopwords may be found in Table 25 in the Appendix.

n-grams: We have not been able to find any open source project that provides the feature of producing n-grams from the corpus. Hence, we implement an addition to the Weka open source project. This is possible by performing some modifications to the *StringToWordVector* class, where the different n-grams are added to the set of attributes. Due to reports (see Section 4.2.3) of best results by using up to tri-grams, i.e., $n = 3$, we do accordingly. The additions to the Weka class are presented in Figures 30 and 31 in the Appendix.

Weighting: We measure the tf-idf weight of each term. This may also be done by use of the *StringToWordVector* class.

To vary the number of dimensions, i.e., attributes, taken into consideration, we also apply the dimensionality reduction technique of selecting the attributes with the highest weights. These weights will normally be the term frequency of an attribute, i.e., the number of times an attribute occurs in an instance to be classified. However, when the weighting scheme of tf-idf is applied, the attributes will be selected accordingly.

6.3 Text Classification

As stated in Chapter 4, Complement Naive Bayes (CNB), Support Vector Machines (SVM), and C4.5 are algorithms that are adequate for use in text classification applications. In this project these are the algorithms that we choose to apply for the further classification due to their different characteristics and features, as described in Chapter 4. Of the probability based algorithms, we choose to apply CNB over plain NB and MNB due to its promised superiority when it comes to text classification. Likewise, SVM is also known as an algorithm that may produce good results, while C4.5, in particular, facilitates visualization of classifier features, in addition to being a classifier used in several classification projects, as presented in Chapter 4.

In this project we seek to carry out the classification of CNB and C4.5 with the Weka classes *J48* and *ComplementNaiveBayes*, respectively. Hence, from now on the C4.5 algorithm will be referred to as J48. For SVM, we apply the *LibSVM* package through the Weka package structure. We will not apply the Weka GUI, but rather apply calls to different classes from scripts. This eases the process of automation, the execution of repetitive calls, and later control of what was actually done. Some of the scripts utilized are later reproduced in this report.

To select the most appropriate kernel for the LibSVM algorithm, we run some preliminary experiments. Of the kernels we test, the *linear* seems to bias the result too much towards the training data, achieving an accuracy of over 98% for the training data, but poorly for the test data. The *polynomial* kernel, however, does not seem to bias the results that much, but does not perform as well as the *radial basis function.* Due to the observed, we apply the radial basis function kernel for the later LibSVM experiments. This kernel is also suggested as a good first choice for many applications (Hsu et al., 2003). However, the choice of kernel does not usually have a large difference when it comes to the achieved accuracy (Han, 2006).

In general, all data available from the D1 corpus will be used, i.e., all 5,646 sentences, unless otherwise is stated. We now present each of the experiments that we seek to carry out in the work of RQ1. For each of the experiments we perform cross validation, presented in Section 4.4.1. This is performed by each of the Weka classifier classes, and makes evaluation possible without defining a proper test set. An overview of each of the experiments presented may be found in Table 11, presented at the end of this chapter.

6.3.1 E1: SVM, J48 and CNB with No Preprocessing

In the first experiment, *E1*, we do not perform any of the preprocessing techniques. Instead we run the algorithms CNB, SVM, and J48 directly on the complete raw data of D1. For each of these algorithms, we want to vary the number of dimensions (i.e., attributes) per class taken into consideration by considering 10, 20, 50, 100, 250, 500, 750, 1,000, 1,250, and 1,500 attributes per class.

6.3.2 E2: CNB, SVM and J48 with Preprocessing

In the second experiment, *E2*, we seek to do the same as in E1, except from one aspect: All of the preprocessing techniques presented in Section 6.2 are performed. These results are to be presented as in E1. However, due to the adding of n-grams, the attribute space is larger than in E1. Hence, we will, for the algorithms capable, try to include larger attribute sets than what is stated for E1. In Figure 32 in the Appendix we have appended a script that shows how the different preprocessing techniques, including use of the implementation of n-grams, may be run in Weka. Note that in this script, both the techniques of stemming and letter transformation are performed apriori.

6.3.3 E3: Comparison of Preprocessing Techniques

The third experiment, *E3*, focuses on the evaluation of each of the preprocessing techniques, and how they may contribute to the result alone. The number of attributes per class will be varied like in E1 and E2, but in this experiment we only consider the algorithm CNB. However, as in E2, we will try to look at bigger attribute sets when considering n-grams due to the increased attribute set.

There are many reasons why we only consider the CNB algorithm, where the most important ones are that it generally produces a very good result, in addition to its efficiency (Pavlov et al., 2004; Witten and Frank, 2000; Han, 2006). We also assume that most of the features discovered with the different preprocessing techniques on CNB may be transferred to the other classification algorithms. This is due to the assumption that a preprocessing technique, for instance reducing the dimensionality of the data material, will produce the same result on different algorithms, but maybe to a different extent. It would also be interesting to see how each of the preprocessing techniques appear for example in tandem, but this is also considered to be outside of the scope of this project.

6.3.4 E4: Varying the Amount of Data

The fourth experiment, *E4*, considers how different sizes of the datasets influence the result of the classifiers. In this experiment we consider datasets consisting of 1,000, 2,000, 3,000, 4,000, and 5,000 instances (i.e., sentences), for each of the algorithms CNB, SVM, and J48. To remove ranges of the data instances, we apply the *RemoveRange* class in Weka.

These experiments will be carried out both when the number of attributes per class is set to be 100 and 1,000. This is to account for the possibility that the classifiers act differently on the same dataset when considering different numbers of attributes. To account for changes in reporting over time and distribution of the classes S, O, and P, we randomise the selection of instances for each sub experiment by use of the *Randomize* class in Weka.

6.4 Summary

In this chapter we have looked at how we intend to carry out text classification experiments in the work of RQ2, based on the text categorization techniques presented in Chapter 4. In Table 11, an overview of each of the experiments presented may be found. In the next chapter we present the results of these experiments.

Exp.	**Sentences**	**Attributes**	**Algorithms**	**Prep.**
E1	All	0, 10, 20, 50, 100, 250, 500, 750, 1000, 1250, 1500	CNB, SVM, J48	no
E2	All	0, 10, 20, 50, 100, 250, 500, 750, 1000, 1250, 1500	CNB, SVM, J48	all
E3a	All	0, 10, 20, 50, 100, 250, 500, 750, 1000, 1250, 1500	CNB	stemming
E3b	All	0, 10, 20, 50, 100, 250, 500, 750, 1000, 1250, 1500	CNB	n-grams
E3c	All	0, 10, 20, 50, 100, 250, 500, 750, 1000, 1250, 1500	CNB	stopwords
E3d	All	0, 10, 20, 50, 100, 250, 500, 750, 1000, 1250, 1500	CNB	td-idf
E3e	All	0, 10, 20, 50, 100, 250, 500, 750, 1000, 1250, 1500	CNB	lowercase
E4a	1000, 2000, 3000, 4000, 5000	100	CNB, SVM, J48	no
E4b	1000, 2000, 3000, 4000, 5000	1000	CNB, SVM, J48	no

Table 11: Overview of experiments carried out in association with RQ1. The denotation of "Exp.", "Sent.", "Attrib.", "Alg.", and "Prep" is the following: "Experiment number", "Number of sentences from corpus", "Number of attributes (dimensions)", "Algorithms" and "Preprocessing"; stating whether preprocessing is to be carried out. When there is more than one attribute in a table window present, this calls for another run-through of the experiment. E.g., in experiment I we get the number of different sentence numbers multiplied with the number of different algorithms run-throughs. In other words experiment 1 needs 11 * 3 = 33 run-throughs.

.

7 Text Classification Results

In this chapter we present the results obtained from the execution of the experiment plan of RQ2, presented in Chapter 6. In comparison to the results presented, the *majority class baseline* accuracy, which is the accuracy achieved when classifying all instances as the majority class, S, achieves an accuracy of 42.9%, as stated in Table 2 in Chapter 3.

All values of F measure, precision, and recall in this report are *macro* values (see Chapter 4), unless otherwise stated. This is due to the fact that the classes of D1 are rather *balanced*, as presented in Chapter 3, and the average measures will be almost identical whether they are calculated based on classes or weighted based on the number of inherent instances. In addition, when it comes to producing the right result, all classes are of equal value to us, and should hence count equally.

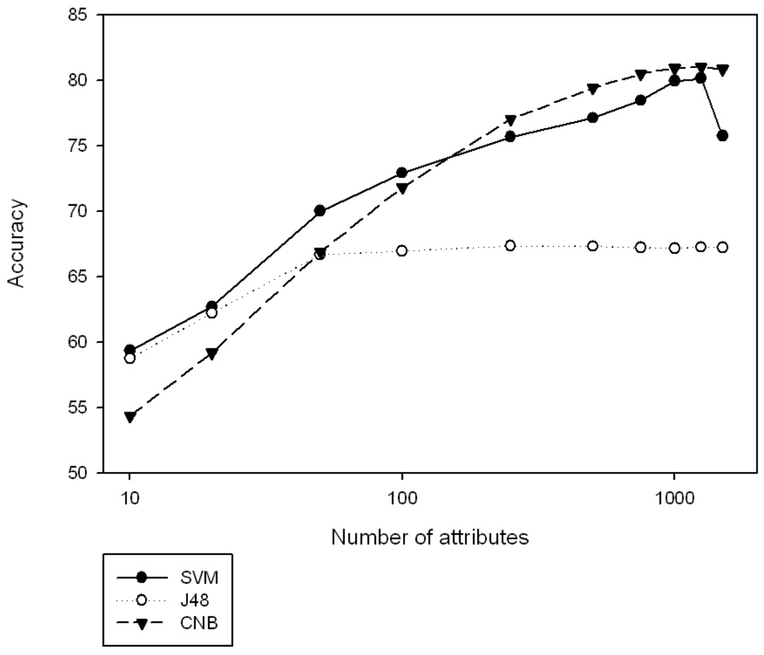
It is also worth noticing that most graphs in this chapter are presented logarithmically, meaning that the x-values of the graphs are respectively 10, 100, 1,000 and so forth. This is due to the observation that the performance of most algorithms changes dramatically between 10 and 250, while the performance subsequently remains more stable for bigger attribute sets. The outline of this chapter is based on Table 11 in Chapter 6. We now present the results of each of the experiments presented in the previous chapter.

7.1 E1: SVM, J48 and CNB with No Preprocessing

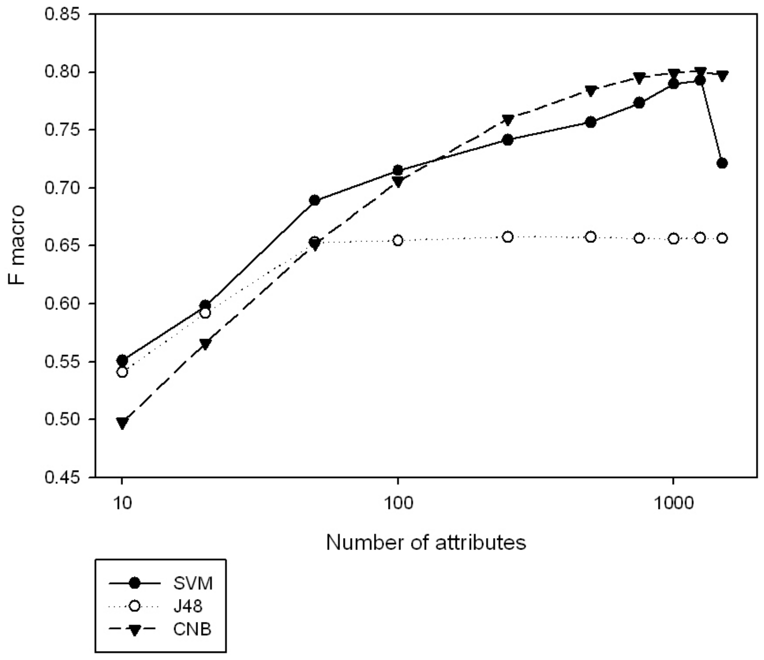
In the first experiment we compare the classification algorithms SVM, J48 and CNB on the complete raw dataset of D1. Figures 8 and 9 show the graphs of the results obtained in E1. The algorithms are compared in terms of *accuracy* in Figure 8(a), *F measure* in Figure 8(b), *precision* in Figure 9(a), and *recall* in Figure 9(b). In these graphs the mentioned measures are presented as functions of the number of attributes per class utilized. The F measure, precision, and recall are calculated as explained in Chapter 4.

The data values of the graphs described above may be found in Table 26 in the Appendix. Table 26(a) presents the different values obtained for accuracy and F-measure, while Table 26(b) presents the different values obtained for precision and recall.

Taking the observed results of all algorithms into consideration, the results tend to reach a maximum when the number of attributes per class is 1,250. Hence, in Table 12 we present more detailed results from the different algorithms, when we take this number of attributes per class into consideration. In particular, the table presents accuracy, F measure, precision and recall for the classification in general, in addition to the F measure, precision, and recall for each of the classes S, O, and P.

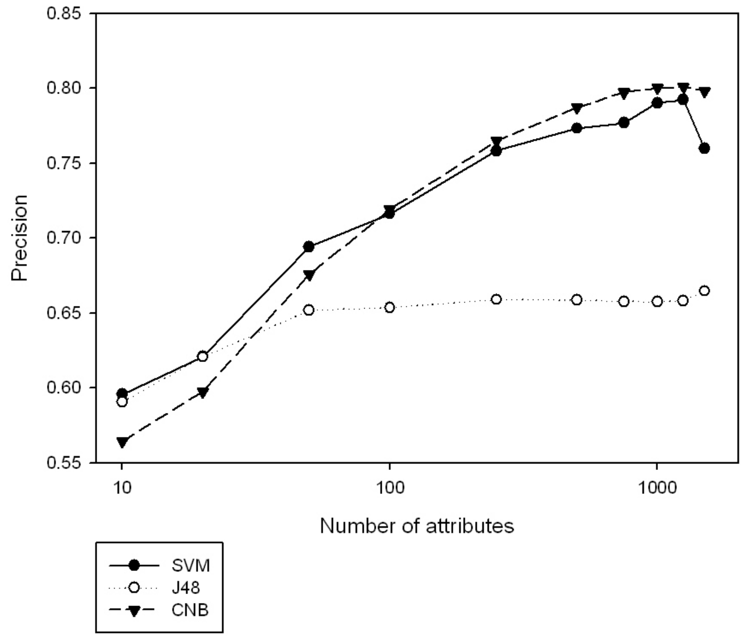


(a) Accuracy

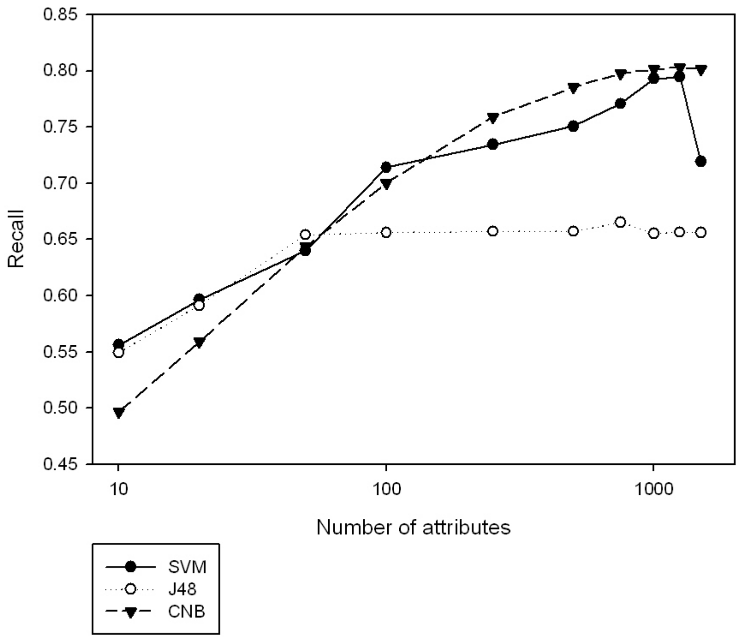


(b) F-measure

Figure 8: Accuracy and F-measure from E1.



(a) Precision



(b) Recall

Figure 9: Precision and recall from E1.

	SVM			J48			CNB		
Acc.	80.15%			67.25%			81.03%		
F	0.793			0.657			0.801		
P	0.792			0.658			0.801		
R	0.795			0.656			0.803		
Class	P	R	F	P	R	F	P	R	F
S	0.849	0.860	0.854	0.767	0.743	0.755	0.853	0.873	0.863
O	0.780	0.728	0.753	0.608	0.660	0.633	0.811	0.733	0.770
P	0.748	0.796	0.771	0.600	0.566	0.582	0.739	0.803	0.769

Table 12: Results obtained from the complete raw dataset of D1, when the number of attributes per class taken into account is 1,250. The table presents accuracy (Acc), F measure (F), precision (P), and recall (R) for the classification in general, for each of the algorithms SVM, J48, and CNB. In addition, precision, recall, and F measure for each of the inherent classes *S*, *O*, and *P* is presented. The total number of instances classified is 5,646.

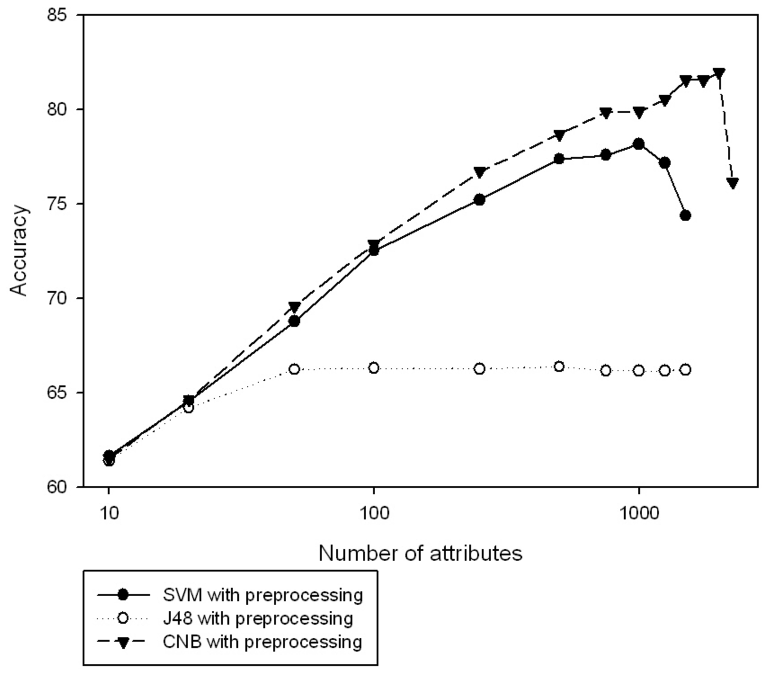
7.2 E2: CNB, SVM and J48 with Preprocessing

This section is organized quite similar to the previous section. The only difference is that the experiments are not carried out on the complete raw dataset of D1, but instead on the preprocessed dataset of D1. An overview of the preprocessing techniques applied may be found in Section 6.2.

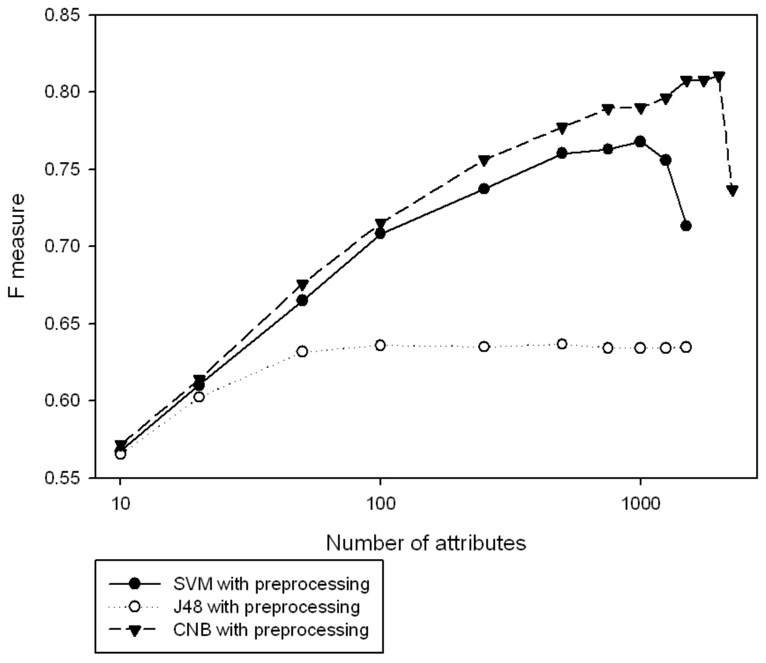
In Figures 10 and 11, the graphs of the results obtained in E2 are presented. The algorithms are compared in terms of *accuracy* in Figure 10(a), *F measure* in Figure 10(b), *precision* in Figure 11(a), and *recall* in Figure 11(b). The data values of these graphs may be found in Table 27(a) and 27(b) in the Appendix.

Due to the addition of n-grams we tried to carry out experiments on attribute sets bigger than 1,500. However, because of the increased computational effort of considering attribute sets of sizes bigger than 1500, and the fact that both SVM and J48 seemed incapable of taking advantage of further additions of attributes, only CNB was considered in this setting. Hence, the graphs of the other algorithms do not have any data values for results obtained when considering more than 1,500 attributes.

In this experiment we do the same observation as in Section 7.1, recognizing that the overall best results, taking all algorithms into consideration, seem to be obtained when the number of attributes per class is set to be 1,250. Hence, in Table 13, we present more detailed results from the different algorithms, when we take this number of attributes per class into consideration, as in Section 7.1.

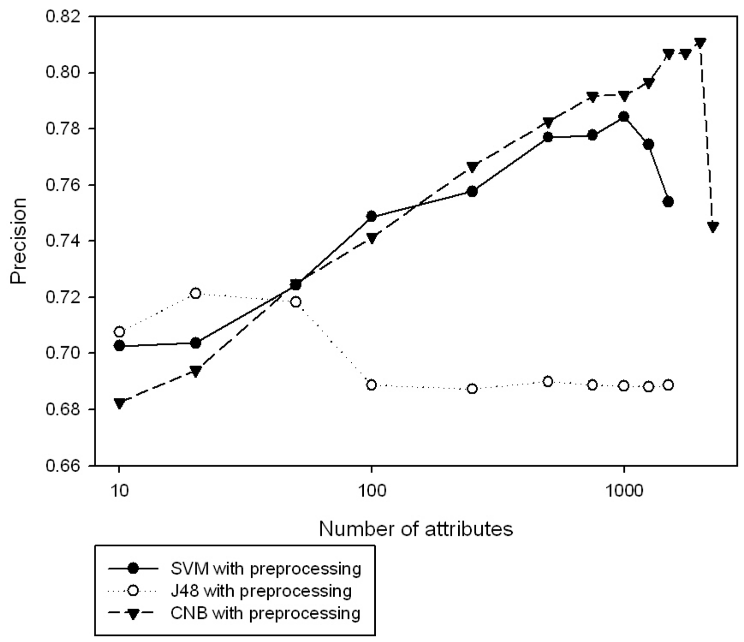


(a) Accuracy

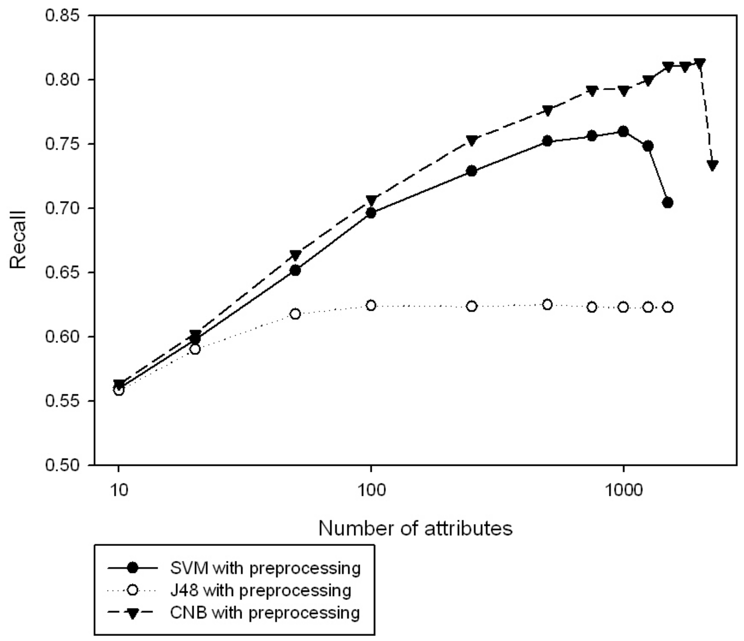


(b) F measure

Figure 10: Accuracy and F measure from E2.



(a) Precision



(b) F measure

Figure 11: Precision and recall from E2.

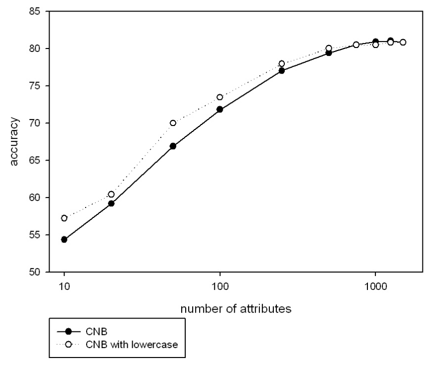
	SVM			J48			CNB		
Acc.	77.152%			66.15%			80.53%		
F	0.756			0.634			0.797		
P	0.774			0.688			0.797		
R	0.748			0.623			0.800		
Class	P	R	F	P	R	F	P	R	F
S	0.758	0.906	0.826	0.626	0.858	0.724	0.848	0.858	0.853
O	0.805	0.676	0.735	0.695	0.567	0.624	0.817	0.730	0.771
P	0.760	0.662	0.707	0.744	0.443	0.555	0.725	0.812	0.767

Table 13: Results obtained from the preprocessed dataset of D1, when the number of attributes per class is set to be 1,250. The denotation is the same as in Table 12. The total number of instances classified is 5,646.

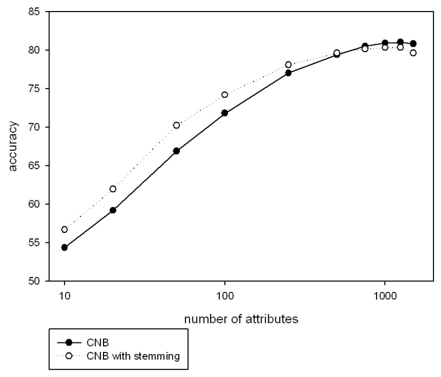
7.3 E3: Comparison of Preprocessing Techniques

In this section we present how the use of different preprocessing techniques may influence the result when applying the classification algorithm CNB. Observing the results obtained in E1 and E2, we assume that there is no practical difference between the results presented in "Accuracy", and the results presented as "F-measure". This is in accordance with Sebastiani (2005), where it is stated that it is only necessary to consider precision and recall, and hence F-measure, in cases where the classes are *far* unbalanced, as described in Section 4.4.2. An overview of the class distribution may be found in Chapter 3, where we may acknowledge that the class distribution is not *far* unbalanced. Hence, of the results obtained in E3 and E4, we only present the obtained accuracy.

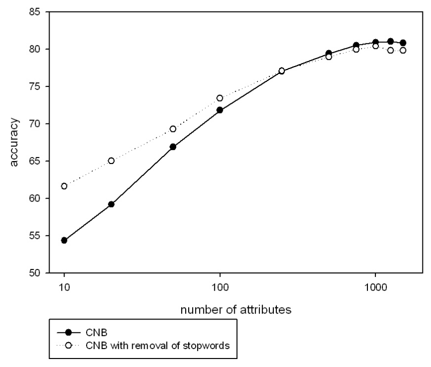
In Figure 12, the different preprocessing techniques are compared to the CNB classification algorithm without any preprocessing technique. This figure is a constellation of five sub figures, each comparing the CNB classification on the complete raw dataset with CNB and the following isolated preprocessing techniques: All letters transformed to lowercase in Figure 12(a), dataset is stemmed in Figure 12(b), stopwords are removed from the dataset in Figure 12(c), the terms in the dataset are tf-idf weighted in Figure 12(d), and n-grams are added to the dataset in Figure 12(e). This is in accordance with Table 11 in Chapter 6. Note that in the latter part of this experiment we considered sets of attributes up to 2,500 due to the increased attribute set caused by the addition of n-grams. The data values that these graphs are based upon may be found in Table 14.



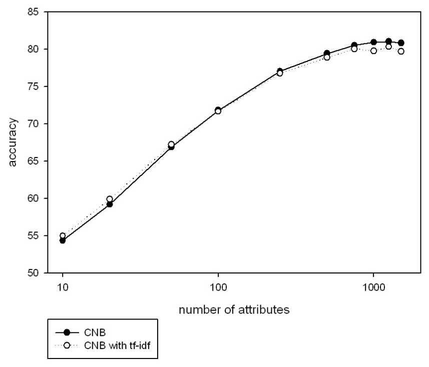
(a) CNB on dataset in lowercase.



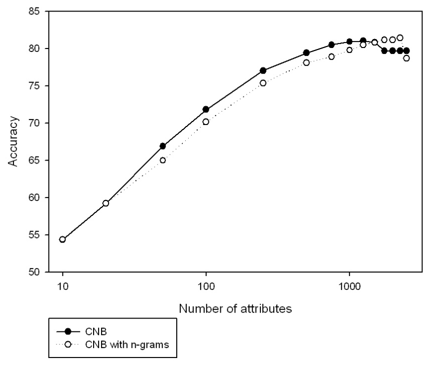
(b) CNB on stemmed dataset.



(c) CNB on dataset without stopwords.



(d) CNB on tf-idf weighted dataset.



(e) CNB on dataset with n-grams.

Figure 12: Comparison of CNB with different preprocessing techniques and plain CNB on the complete raw dataset of D1.

Attrib.	Preprocessing technique					
	No	Lower-case	Stemm-ing	Stop-words	tf-idf	n-grams
10	54.37	57.21	56.70	61.64	54.98	54.37
20	59.19	60.41	61.94	65.02	59.88	59.19
50	66.91	69.98	70.23	69.27	67.22	64.98
100	71.82	73.45	74.18	73.41	71.64	70.12
250	77.03	77.93	78.09	77.10	76.78	75.36
500	79.40	80.02	79.61	78.96	78.87	78.11
750	80.50	80.46	80.16	79.97	80.02	78.91
1000	80.92	80.50	80.32	80.39	79.76	79.79
1250	81.03	80.82	80.36	79.81	80.32	80.48
1500	80.84	80.82	79.63	79.81	79.68	80.80
1750	79.68					81.15
2000	79.68					81.15
2250	79.68					81.42
2500	79.68					78.69

Table 14: Data from the comparison of CNB with different preprocessing techniques and plain CNB on the complete raw dataset of D1. In the table, "No" simply denotes that no preprocessing technique is applied.

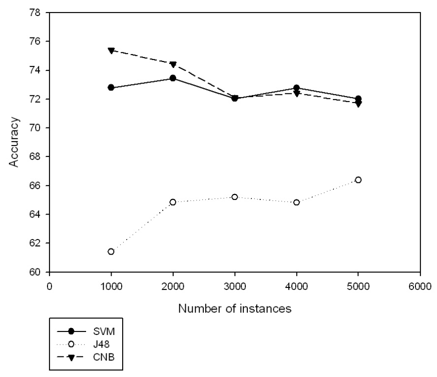
7.4 E4: Varying the Amount of Data

In this section we present the results of E4: How the result varies depending on the amount of data available for classification training. We vary the amount of data from 1,000 to 5,000 instances, running experiments at leaps of 1,000. These experiments are carried out both when the number of attributes per class is set equal to 100 and 1,000.

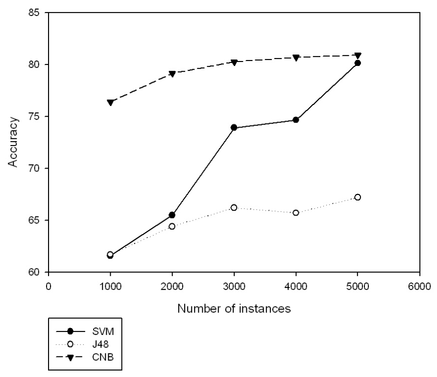
The results of this experiment are presented in Figure 13. Figure 13(a) considers the case when the number of attributes per class is set to 100, and Figure 13(b) considers the case when the number of attributes per class is set to 1,000. The data values of Figure 13 are presented in Table 15.

7.5 Summary

In this chapter we have presented the results of the experiments presented in Chapter 6. In the next chapter we evaluate the results and see what, if any, conclusions we can draw from the results obtained.



(a) Accuracy for 100 attributes per class.



(b) Accuracy for 1,000 attributes per class.

Figure 13: Accuracy for the number of data instances between 1,000 and 5,000. We run the experiment with respectively 100 and 1,000 attributes per class.

Attrib.	Inst.	SVM	J48	CNB
100	1000	72.80	61.40	75.40
	2000	73.45	64.85	74.45
	3000	72.03	65.20	72.10
	4000	72.78	64.83	72.43
	5000	72.01	66.39	71.70
1000	1000	61.60	61.70	76.40
	2000	65.50	64.40	79.15
	3000	73.90	66.20	80.27
	4000	74.65	65.70	80.70
	5000	80.14	67.19	80.90

Table 15: Data values for E4, as presented in Figure 13.

8 Text Classification Evaluation and Discussion

In this chapter we evaluate the results achieved in the experiments of RQ2, presented in the last chapter. In addition we discuss other issues concerning the procedures applied, before we eventually present a conclusion of the findings.

8.1 Comparing the Algorithms (E1 and E2)

In this section we evaluate the results of E1 and E2. First, we present some general characteristics from the results of these experiments, and then we compare the results obtained by each of the classifiers in E1 and E2. Finally, we execute a *sign test* to evaluate some of the findings statistically.

8.1.1 Characteristics of E1

Several interesting characteristics of the data values in Figures 8 and 9, extracted from Table 26 in the Appendix, may be found by manual inspection. Concerning the accuracy, we observe that initially, with only a few attributes to help, SVM seems to perform slightly better than J48, while CNB performs poorer. At about 50 attributes per class, the J48 classifier seems unable of benefiting from more attributes, and its performance stabilizes at an accuracy of approximately 67%. Han (2006) explains how proper calibration of the pruning may improve the accuracy. Hence, during the experiment setup, we tried to adjust the level of pruning to see if this could improve the result. However, a heavier pruning resulted in a too general classifier, and hence performing worse, while the less pruned tree resulted in overfitting, explained in Section 4.2, and again yielding worse performance.

Both SVM and CNB manage to achieve higher values of accuracy than the J48 algorithm. CNB, however, seems to be the best classifier when considering 250 or more attributes per class. Both CNB and SVM perform best when the number of attributes per class is set to 1,250. The performance of CNB and SVM decreases for further addition of attributes. This is probably due to overfitting of the classifier. When the number of attributes is set to 1,250, SVM, J48, and CNB obtain accuracies at respectively 80.15%, 67.23%, and 81.03%.

Observing the F measure achieved in E1, we note that these values to a large extent correspond to the achieved accuracy, as stated in Chapter 7. The factors deciding the F measure, i.e., the precision and recall, also behave in the same manner, with some minor variations.

Table 12 in the previous chapter presents precision, recall and F measure of each of the classes *S*, *O*, and *P*, when the number of attributes per class is set to 1,250. In general, for all classifiers, the results achieved for the *S* class is higher than for the other classes. Many reasons for this exist. First, as presented in Chapter 3, there is a higher percentage of *S* classes than the other two, and hence one may assume that the classifier is better fit at classifying instances of *S*, respectively.

Second, because of the *S* class' already domination, it is likely to dominate even more. In cases where the words of a sentence are not present in the classifier, the sentence is always classified as *S*. This decision is based on the assumption that when there are no words to help the classifier, there is a bigger probability that the sentences belongs to the *S* class than the *O* or *P* class since the *S* class is the major class of the training corpus. Hence, the domination of *S* causes an even increased domination of this class.

Third, one may assume that terms represented by the *S* class, representing the subjective thoughts of a patient, may consist of a more condensed vocabulary, and may hence easier correspond to attributes present in the attributes set represented by the *S* class. Example of such words may be "feel" and "think". Hence, we may assume that there is a higher probability that a sentence of the classes *O* or *P* contain words that are not present in the classifier, and hence may be falsely classified as *S* in cases where there is no "match" between the words present in the sentence and the classifier.

Other deviant observations one may make is the low precision and recall, and hence the *F measure*, of the *P* class of the J48 classifier. This means that this classifier both misclassifies many *P* sentences as *O* or *S*, and misclassifies many instances of *S* and *O* as *P*.

8.1.2 Characteristics of E2

There are several interesting characteristics that may be found by inspection of the results of E2 as well. The results of E2 are presented in Figures 10 and 11 in Chapter 7, extracted from Table 27 in the Appendix. When applying the preprocessing techniques, some techniques make the set of available attributes decrease, while other methods make the set of available attributes increase. Techniques like applying only lowercase, stemming, and removing stopwords are techniques that make the set of attributes decrease, while addition of n-grams makes the set increase. This is further investigated in Section 8.2.1. The outcome of applying the preprocessing techniques is, nevertheless, additive concerning the number of attributes available.

Initially, when the number of attributes per class is set equal to 10, all algorithms perform equally, achieving an accuracy of approximately 61.5%. Increasing the number of attributes, J48 seems again to be unable of taking advantage of this, and stabilizes at an accuracy of approximately 66.2%. This happens when considering more than 50 attributes per class.

On the other hand, SVM and CNB are both able to further classify on the attributes added, and continue to increase the accuracy. SVM performs best at 1,000 attributes per class, obtaining an accuracy of 78.16%. Due to the computational effort, and the fact that SVM and J48 seems incapable of handling attribute sets of sizes bigger than 1500, these algorithms are not taken into account for bigger attribute sets per class than 1,500. Running such experiments with SVM would also most likely take several days to complete. CNB, on the other hand, manages to further increase the accuracy obtained until taking 2,000 attributes per class into consideration, yielding an accuracy of 81.95%. The accuracy decreases dramatically if we take more than 2,000 attributes per class into consideration.

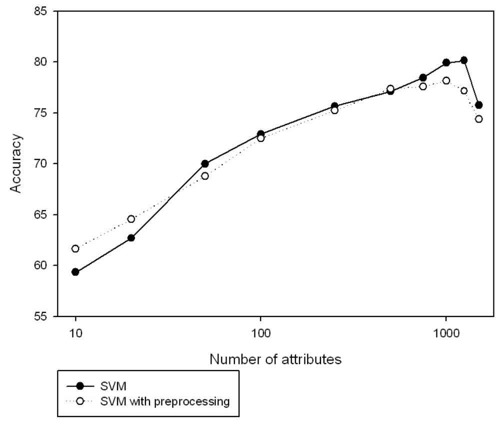
Precision and recall, and hence the F measure, of all algorithms behave to a large extent like the achieved accuracy. However, an exception is the precision of the J48 classifier, which initially slightly increases, before it decreases considerably and stabilizes. This indicates that the probability that a random document is classified correctly decreases.

Like in E1, we look at how the different classes behave when the number of attributes per class is set to 1,250. These data values may be found in Table 13 in the previous chapter. Similar to the conclusions drawn in the previous section, we may conclude that both precision and recall is generally higher for the *S* class, and that the results are poorest for the *P* class. J48 performs particularly poor classifying the *P* class and the recall is quite low. This means that the probability that a random document that should be classified under this class is actually classified as *P* is low.

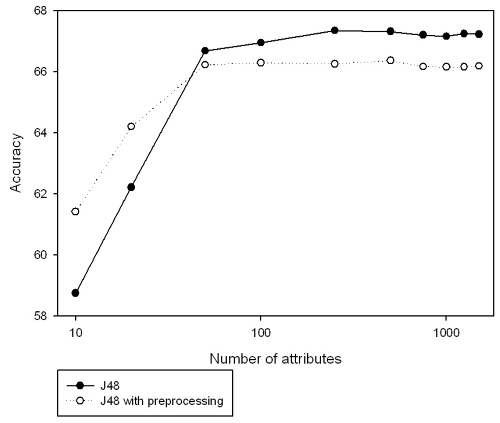
8.1.3 Comparing E1 and E2

In this section we seek to compare the results obtained by each classifier in respectively E1 and E2. Figure 14 presents the result of different classifiers when the result is preprocessed and not. This figure is extracted based on Table 26 and 27 in the Appendix. Here we look at how the *sum* of all preprocessing techniques influence the result, but we do not try to give any explanations of the results obtained. Instead this will be done for each of the preprocessing techniques in Section 8.2.1.

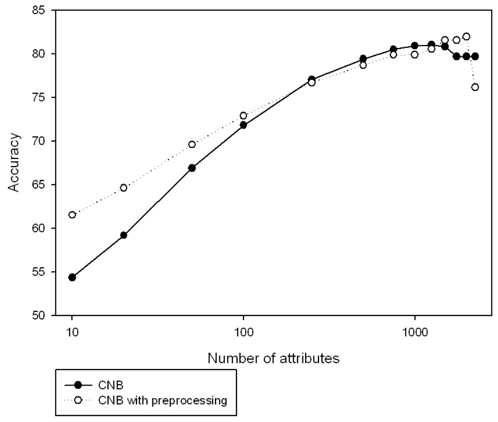
In Figure 14(a), the result of the SVM classifier on the raw and preprocessed dataset of D1 is presented. Initially, the classifier of the preprocessed data obtains an accuracy of 61.65%, about 2% better than its accuracy on the raw data. The accuracy of both graphs in the figure increases with the number of attributes added per class. However, the accuracy of the classifier built on the raw data reaches its maximum at 1,250 attributes (80.15%), while the accuracy of the classifier built on the preprocessed data reaches its maximum at 1,000 attributes (78.16%).



(a) SVM



(b) J48



(c) CNB

Figure 14: Comparison of algorithms with and without the use of preprocessing techniques.

The result of the J48 classifier on the raw and preprocessed dataset is presented in Figure 14(b). Initially the classifiers share the very same characteristics, yielding an accuracy increasing with the number of attributes added, while the accuracy stabilizes above 50 attributes per class. However, the classifier built on the preprocessed data seems more stable than the other, performing better initially, but stabilizes at a slightly lower level than the other, just above 66%. The classifier built on the raw data, on the other hand, yields in general an accuracy just above 67%.

In Figure 14(c), the result of the CNB classifier on the raw and preprocessed dataset is presented. Like the classifiers described above, classification of the preprocessed data achieves a considerably higher accuracy when the number of attributes is low. Both classifiers in this figure perform better the more attributes that are added, to a certain point. Between 250 and 1,250 attributes, the classifier of the raw data performs slightly better than the other. However, beyond 1,250, the classifier of the preprocessed data performs better, before decreasing dramatically beyond 2,000 attributes. The other classifier remains stable beyond 1,750 attributes per class, not having more attributes to take advantage of. The best result for the classifier built on the raw dataset is achieved at 1,250 attributes (81.03%) while the other performs best at 2,000 attributes (81.95%).

8.1.4 Sign Test

In this section we evaluate the statistical likelihood of some of the trends observed above. It is outside of the scope of this project to evaluate each and every observation made, but we evaluate some of the conclusions that we find most valuable. More specifically, we would like to evaluate which algorithm produces the best results in respectively E1 and E2, and whether the preprocessing techniques produce a better classifier or not, by comparing each of the algorithms from both E1 and E2. However, we only give attention the “peaks” of the curves and do not consider the other areas of the graphs.

As stated in Chapter 6, the classes of the corpus are relatively balanced. Hence, no classes may be called *rare*, and a weighted statistical test emphasizing the common classes, i.e., a micro sign test, will in practice produce the same statistical result as a macro sign test, valuing the rare classes more profoundly (Yang and Liu, 1999).

In the corpus, there are only three classes: S, O, and P. A binomial distribution table for $n = 3$ may give a P value of 0.1250 at best (Wadsworth, 1990). If we choose to use $\alpha = 0.05$ as the critical value in the statistical test, there is no way that we may obtain a result below this value, $\alpha < 0.05$. This means that the outcome of the classes themselves may not be used as statistical evidence of a classifier’s superiority or deficiency.

During the creation of the classifier, the ten-fold cross validation tests the classifier against ten different folds, i.e., portions, of the data, producing the final results. This gives us the opportunity to evaluate each of the ten outcomes of the cross validation, as performed by Kienzle et al. (2006). Using the result of the folds of the cross validation is vouched for by Han (2006), and states that the results of each fold may be considered as different and independent samples.

As stated above, the SVM and CNB classifiers of E1 perform best when taking 1,250 attributes per class into consideration. The performance of J48, on the other hand, does practically not change when considering more than 250 attributes per class. Hence, we evaluate the classifiers taking 1,250 attributes per class into consideration.

Even though the average of the accuracy obtained in E2 is slightly higher when considering 1,000 attributes than 1,250 attributes, we choose to focus the sign test on 1,250 attributes, to be able to compare the values of the folds more easily with the results obtained in E1. Also, the distance between the result obtained in E2 for respectively SVM and CNB is larger when the number of attributes equals 1,250 than 1,000. Hence, a statistical valid result found when the number of attributes per class equals 1,000 is even more likely to be true when the number of attributes equals 1,250. The accuracy of the folds of the cross validation of each of the algorithms for E1 and E2 may be found in Table 28 in the Appendix, where Table 28(a) presents the results of E1, and Table 28(b) presents the results of E2.

By manual inspection of Table 28, we observe that for both E1 and E2 the accuracy achieved by every fold of SVM and CNB is better than the accuracy achieved by J48. Similarly, each fold of CNB also performs better than those of SVM. Hence, recollecting the sign test method of Section 4.4.2, for all of these observations we get $n = 10$ and $k = 10$. With these values, consulting Table 24 in the Appendix, we achieve P = 0.010. In Table 16, the different algorithms are compared as proposed by Yang and Liu (1999).

Alg. 1	Alg. 2	E1	E2
CNB	SVM	$\gg$	$\gg$
CNB	J48	$\gg$	$\gg$
SVM	J48	$\gg$	$\gg$

Table 16: Comparison of the different algorithms. Denotation: "$\ll$" or "$\gg$" means P value ≤ 0.01, "<" or ">" means P value ≤ 0.05, and "~" means P value > 0.05.m

Combining the results presented in Table 16, we may conclude that CNB is superior of both algorithms, and that SVM is superior of J48. Hence, we get CNB $\gg$ SVM $\gg$ J48, when the number of attributes equals 1,250, indifferent to whether preprocessing of the data is performed or not.

In Section 8.1.3, we compared each of the classifiers of E1 and E2, to see if the classifiers yield better results when they are applied on data that are preprocessed. We now seek to statistically compare each of the algorithms when the number of attributes per class is set to 1,250. Table 28 in the Appendix presents the accuracy for each of the folds in the ten-fold cross validation of classifiers both when the data is preprocessed and not. However, since the CNB algorithm manages the use of more than 1,500 attributes, we carry out a statistical test with the number of attributes set to 2,000 as well. This data is presented in Table 29 in the Appendix.

Through manual inspection of Table 28 and 29, we observe that none of the folds of SVM produce better results when they are preprocessed, while for J48, three folds achieve better results when being preprocessed. CNB, on the other hand, achieve better results for four out of six folds when the data is preprocessed, taking 1,250 attributes into consideration. All folds produce better result for CNB when the number of attributes per class equals 2,000. Applying the same techniques as above, this produces the results presented in Table 17. Observing the results, we may acknowledge that use of preprocessing techniques does not necessarily increase the accuracy obtained. CNB performs better on the preprocessed dataset taking 2,000 attributes per class into account. Taking 1,250 attributes per class into account, both CNB and J48 are indifferent to whether preprocessing is carried out. SVM, on the other hand, performs poorer on the preprocessed dataset than on the raw dataset.

Alg. 1	Alg. 2	Attrib.	E1/E2
SVM	SVM preprocessed	1,250	$\gg$
J48	J48 preprocessed	1,250	˜
CNB	CNB preprocessed	1,250	˜
CNB	CNB preprocessed	2,000	$\ll$

Table 17: Comparison of the result of the algorithms with the raw dataset and the preprocessed dataset. The denotation is the same as in Table 16.

A number of different techniques exist to ascertain which classifier is the better. The sign test states that among the tested classifiers, Complement Naive Bayes is the better one, both when the data is preprocessed or not. However, two numbers of attributes, 1,250 and 2,000 produce different results when it comes to the gain of preprocessing the data. This, in addition to the poor influence of the preprocessing techniques on the other algorithms, suggest that preprocessing techniques should be used with caution. In the next section we evaluate how each of the preprocessing techniques influence the result of CNB.

8.2 Preprocessing Techniques

In this section we evaluate how each of the optional preprocessing techniques influence the result of the CNB algorithm. We consider each of the sub figures of Figure 12, extracted from Table 14, Chapter 7. Finally we present two decision trees of the J48 classifier, and seek to find empiric effects of the different optional preprocessing techniques.

8.2.1 E3: Comparison of Preprocessing Techniques

To start with, performing the simple operation of turning all letters into *lowercase*, greatly influences the result when considering a low number of attributes, but performs almost identical to no preprocessing when considering a high number of attributes. It is reason to believe that this is caused by the fact that turning all letters into lowercase decreases the set of attributes, and hence the dimensionality. This again may give a higher degree of attribute recognition when there is only room for a low number of attributes.

Stemming has to a high degree the same characteristics as the transformation of all letters to lowercase. It reduces the attribute space, and hence the result is considerably improved when considering a low number of attributes. However, considering many attributes the trend is somewhat negative, probably not giving the same degree of discrimination as when considering the raw dataset.

Considering the stemming, it is worth noticing that when applying NORKOMPLEKS, we are not able to perform stemming in cases where a conjugated word has two meanings, because we are not able to know what stem the word is built on without performing considerably more analysis of the text. An example of such a word may be the Norwegian word “hatt”, which in English may mean both “had” and “hat”. It is likely that correcting this ambiguity would emphasize the result of the stemming, producing an even higher result when considering the little attribute set, and vice versa. Pomikálek and Rehurek (2007) have shown that stemming does not influence the result dramatically, while other reports have shown that stemming both tends to worsen and make results better when applied in text classification applications (Sebastiani, 2002).

The removal of *stopwords* is the preprocessing technique that gives the highest boost to the smaller attribute sets, removing many common words. These are words that in reality have very little discriminating effect. However, like stemming, this technique probably also removes some of the discriminating features of the greater attribute set and performs slightly poorer when taking this dataset into consideration.

The weighting technique of *tf-idf* gives heavier weights to words that occur more frequently in an instance (i.e., sentence) to be classified. However, in our experiment, this technique influences the result to a very small extent, probably due to the nature of the instances to be classified, that very few sentences contain the same word twice.

Of the preprocessing techniques considered, only the technique of adding *n-grams* expands the set of attributes. Hence, we extend the experiment to include sets of attributes per class up to 2,500. Adding the different n-grams increases the specialty of each class, making it more difficult to classify with only a few attributes to help. However, when considering the bigger set of attributes, this technique improves the result considerably, obtaining the best accuracy of all preprocessing techniques considered.

Lewis (1992) also tries text classification with phrases, but reports discouraging results. This is suggested to happen because phrases do have superior semantic qualities, but inferior statistical qualities, than classification based on single terms. Lewis (1992) further states that a classification task based on phrases will have more terms and synonymous terms, lower document frequency for terms, and lower consistency of assignment because synonymous terms are not assigned to the same documents. However, the curves presented in Lewis (1992) only show the result before the curves cross, hence not presenting the equivalent of the right part of Figure 12(e), where the highest obtained accuracy may be found. Other experiments have reported both positive and negative influence on the result when using n-grams (Pomikálek and Rehurek, 2007). To shrink the number of dimensions in the attribute set, Tzeras and Hartmann (1993) remove terms that occur less than three times in the corpus. This is indirectly the very same as we have done in this experiment, but the other way, only considering the most frequent terms.

This section exemplifies how some preprocessing techniques work on a given set of data. It is, however, worth noticing that previous experiments have shown that the characteristics of the results are dependant on both the corpus and algorithms used (Pomikálek and Rehurek, 2007).

8.2.2 A Classifier Built on Raw versus Preprocessed Data

In this section we take advantage of one of the most important features of the J48 classifier: Its intuitive way of presenting the classifier. More information about decision tree classifiers may be found in Section 4.3.3. Due to the limited space available in this report, we here only present decision trees where the number of attributes per class is equal to 10. Taking 10 terms per class into consideration means that the ten most frequent words for each class are selected. Sometimes, words from the different classes are the same, and hence the total number of terms may be less than the number of attributes selected multiplied with the number of classes.

In Figures 15 and 16 we have presented the decision tree built on raw data of D1, where the first figure presents the outline of the tree, and the other figure presents the pruned top. This tree consists of 16 different terms. However, several of the terms are used at multiple locations in the tree, thus giving the impression that the tree contains more terms than what is the case. The numbers at the leaf nodes indicate the result of the training data. The first number indicates the number of instances that suit the selection, while the last number indicates the number of erroneous training instances that are left there.

If we take a closer look at the words in the tree, we note that many of the words may be found in the list of the stopwords presented in Table 25 in the Appendix. In fact, only two of the terms presented in Figure 16 are not considered stopwords: “us”[11] and “vurdering”[12].

As seen in Figure 12(c) and explained in Section 8.2.1, the removal of stopwords increases the performance of the classifier especially when the number of attributes to help is low. However, of the different terms in Figure 16, we assume that terms like “om”[13] or “over” are words that do not appear indifferent to its inherent class. The first word, often appearing in a causal setting, may be overrepresented in the P class, while the other word “over”, which is a preposition, may be overrepresented in either of the classes S or O. However, finding the relevance of each of the words of the class is outside of the scope of this project.

The decision tree of the preprocessed data, presented in Figure 17, consists of 20 different terms. Here we see that the stopwords are removed, the stemming has taken place, all words are in lowercase, and n-grams are added to the corpus, as presented in Chapter 6.

The removal of the stopwords makes the classifier capable of including other more specific words such as “fremmedlyder”[14] or “CRP”[15]. Concerning n-grams, we see that the classifier utilizes “hun_ha”[16] or “han_ha”[17], and defines sentences containing these words as a part of the S class. These n-grams also exemplifies the effect of the stemming; the “ha” term in the n-grams may appear in other forms, both “har”[18] and “hadde”[19]. Hence, the n-gram “han_ha” both account for “han_har“ and “han_hadde”. However, as stated

[11]Norwegian abbreviation for “undersøkelse”. Eng: examination
[12]Eng: assessment
[13]Eng: if, in, about
[14]Eng: Unfamiliar sound
[15]C-reactive protein
[16]Eng: she_to_have
[17]Eng: he_to_have
[18]Eng: has
[19]Eng: had

in Section 8.2.1, the “hatt” conjugation of the verb ”ha“ is not included by the table lookup stemming due to its ambiguous meaning, where it may mean both “has had” and “hat”.

The two n-grams considered above also call attention in another manner. Both “han_ha” and “hun_ha” are n-grams that suggest that the inherent sentence belongs to the *S* class. However, “han” and “hun” could have been swapped to a more common term, like “pp” (personal pronoun), or something similar. This may probably be done for several types of terms in the corpus, applying domain knowledge to combine multiple features together reducing the set of attributes. This technique is known as *feature extraction* (Wilcox and Hripcsak, 2003). This work may also be considered beyond the scope of this project.

At the far lower left end of the tree presented in Figure 17, we see that the sentences not consisting of any of the words mentioned, are tagged according to the major class of the sentences left to be classified. In trees consisting of more terms than the trees showed in Figure 16 and 17, this dramatic majority voting at the far lower left end of the tree does not take place. Instead, the majority voting is spread around the tree. These trees are of sizes not adequate for reproduction in this report.

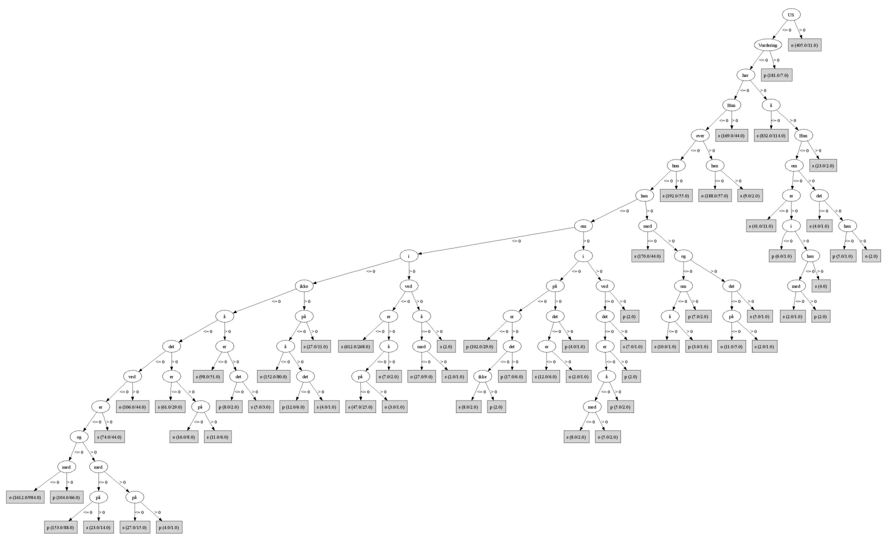


Figure 15: Outline of a J48 decision tree with up to 10 attributes per class drawn in *Graphviz* on the raw dataset.

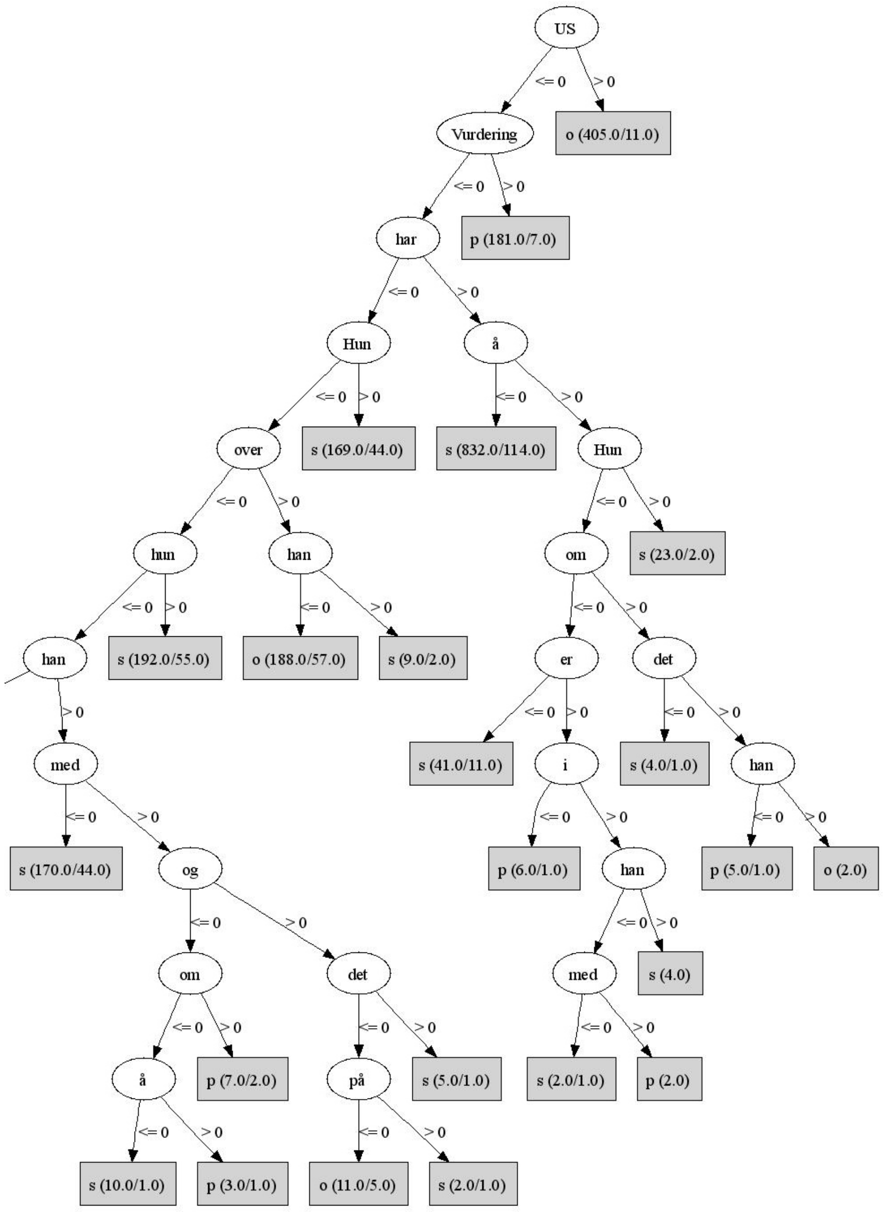


Figure 16: Pruned J48 decision tree with up to 10 attributes per class drawn in *Graphviz* on the raw dataset.

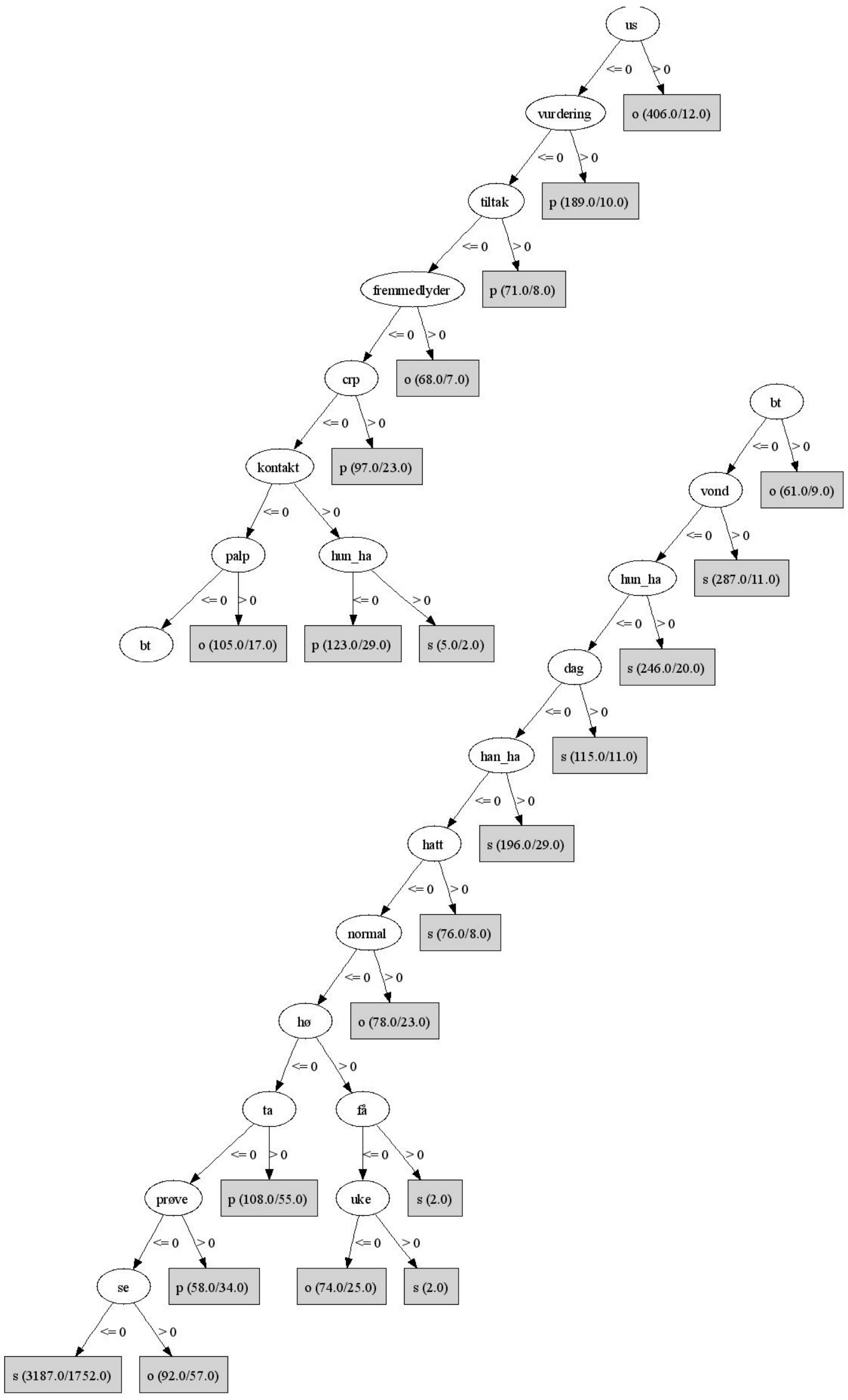


Figure 17: J48 decision tree with up to 10 attributes per class drawn in *Graphviz* on the preprocessed dataset. Note that the figure has been re-touched to fit in the report. Hence, the apparent leaf node "bt" continues at the top of the branch to the right.

8.3 E4: Varying the Amount of Data

In this section we evaluate the results of E4, presented in Figure 13 in the previous chapter. We first look into the classification of different sets of instances when the number of attributes is set to 100, before we look at the same sets of instances when the number of attributes is set to 1,000.

8.3.1 The Small Dataset of 100 Attributes

When it comes to the amount of data needed for the creation of a good classifier when the number of attributes is set to 100, the result is somewhat ambiguous. First, looking at the accuracy of classifying datasets with only 100 attributes in Figure 13(a), J48 is the only algorithm that actually shows a positive trend when adding more instances to the classifier. However, both SVM and CNB perform slightly worse when adding more instances to the dataset. One reason for this negative trend may be that 100 attributes may be "sufficient" for the classification of 1,000 instances, but does not manage to discern between as many as 5,000 instances, and hence producing a slightly worse accuracy.

8.3.2 The Bigger Dataset of 1,000 Attributes

When varying the amount of instances with the number of attributes set to 1,000, as presented in Figure 13(b), the trend is the same for J48, but different for SVM and CNB. J48, in fact, performs almost identical to the experiment considering only 100 attributes, yielding slightly better results when considering more attributes. This result confirms the result of E1 and E2, that J48 seems to be incapable of taking advantage of the addition of more attributes per class.

CNB, and particularly SVM, on the other hand, perform remarkably better when adding more instances to the classification. The accuracy of CNB increases moderately when applying more instances, while the accuracy of SVM increases dramatically. The accuracy of the algorithms SVM, J48 and CNB is respectively 80.14%, 67.19%, and 80.90% for the biggest set of instances.

Hence, we conclude that in the first experiment, taking 100 attributes into consideration is not enough to manage the classification of the decreasing number of instances, while when taking 1,000 attributes into consideration, the classifiers are capable of taking advantage of the extra instances added.

8.4 Other Issues

The previous sections explain the results of the work on RQ1. In this section we consider some other issues concerning the work.

8.4.1 Heuristic Rules for Encounter Note Anatomy

When classifying the sentences, we do not pay any attention to the structure of the encounter note. Other experiments (Nilsson et al., 2003) and manual inspection performed by us, validate that encounter notes are often written in a *SOAP* sequence. This is a feature of the encounter notes that could be advantageous to exploit. First, we would have to classify the sentences according to their belonging encounter notes. For example, we may picture a sequence of classified sentences $\mathrm{s}_n = \{S, S, P, S, S, O, O, P, P, P\}$. Then we assume that by the use of heuristic rules, like letting the classified sentence fit into its surroundings, the result will improve. In the example mentioned above one may assume that sentence number three, the P, is misclassified. Hence one would get a new sequence of sentences $\mathrm{s}_{n_{new}} = \{S, S, S, S, S, O, O, P, P, P\}$.

8.4.2 Domain Knowledge

Applying domain knowledge may give a boost to medical text report classifiers (Wilcox and Hripcsak, 2003). As suggested in Section 8.2.1, some of the stopwords that are relevant, may in reality have a discriminating value when it comes to classification. A solution is to let domain experts decide for each of the classes what terms may be considered most relevant to include, performing a manual feature selection. Another technique applied by domain experts is *feature extraction* (Wilcox and Hripcsak, 2003), where terms whose meaning is indifferent is substituted by common terms, like when suggesting that *she* and *he* are terms that both describe the same semantics in the classifier context. This is also exemplified in Section 8.2.2.

Wilcox and Hripcsak (2003) conclude that the use of domain knowledge is a very important factor concerning the performance of classifiers. On the other hand, Sebastiani (2002) states that the straight forward approach of applying a set of previously classified documents yield an accuracy comparable to that achieved by human experts. The latter does not concern medical data in particular. However, when applying techniques like feature selection, the use of human expert manpower is often needed, making the classification task more expensive (Sebastiani, 2002; Wilcox and Hripcsak, 2003).

8.4.3 Computational Infeasibility

It is expected that especially the LibSVM algorithm could have performed a little better with parameter tweaking (Hsu et al., 2003). LibSVM does provide a package, *easy.py*, to find the best parameter settings for LibSVM on a given dataset. This package checks different pairs of parameters, to find the best performance possible of LibSVM. Weka, on the other hand, has its own *GridSearch* class, shortly described in Table 6 in Chapter 5. However, an experiment with LibSVM on our computer, specified in Table 8 in Section 5.4, uses approximately seven hours to complete on the D1 dataset, taking 1,000 attributes into consideration, making the experiments of RQ1 take days to complete. Hence, carrying out experiments for the set of parameter pairs on this dataset would not have been computationally possible within the scope of the project. A solution, however, could have been scaling down the data obtaining best parameters for the reduced dataset, possibly at the expense of obtaining inaccurate parameters. Variation of the algorithmic variables comes at a cost of computational complexity. In many cases, the variation will be computationally infeasible (Pomikálek and Rehurek, 2007).

When parameter tweaking takes place, some argue, that in addition to a training and a test set, one needs a validation set (Salzberg, 1997; Sebastiani, 2002). The problem is, when using a training and a test set, the parameters will be tweaked to perform as best as possible on the test set, and hence possibly bias the result. To deal with this problem, Salzberg (1997); Sebastiani (2002) propose the addition of an extra set when performing k-fold cross validation. More specifically, they propose to first use a training and a test set to optimize the variables of the algorithms, and then run the classifier on a *validation* set, to get an unbiased result.

However, finding the absolute best result possible is not the scope of this thesis, but rather to find and compare some adequate algorithms to see *if* and *how* applicable algorithms may be used on the available data. Hence, we did not find the possible gain of performing these tests to justify the effort.

8.5 Conclusion

From the evaluation in this chapter we may conclude that CNB was the overall best classifier. It performed best on both the raw and the preprocessed dataset, achieving accuracies of 81.03% and 81.95%, respectively. SVM also performed satisfactorily, and yielded accuracies of 80.15% and 78.16% on the same datasets. On the other hand, J48 failed to take advantage of large attribute sets and achieved accuracies of around 67% for both datasets.

The overall best result was achieved by CNB on the preprocessed dataset when considering a specific number of attributes. The other two classifiers did not perform best applying the preprocessed dataset. In addition, finding an adequate ratio between the number of attributes and the size of the dataset is not straightforward. This suggests that the use of preprocessing techniques should be applied with caution.

Several observations are made by comparing the different preprocessing techniques on the CNB classifier. Preprocessing techniques that decrease the set of attributes gives boost to the performance of the small attribute set, but slightly decreases the performance when considering the larger attribute set. Oppositely, the preprocessing technique of adding n-grams decreases the achieved accuracy when applying the smaller attribute set, and increases the achieved accuracy when considering the bigger set of attributes. However, from the obtained results, it is not evident which techniques that improve classification results. It still may be suggested, as above, that the ratio between the number of attributes taken into account and the size of the dataset may indicate which preprocessing technique that is most likely to improve the classification results.

When it comes to the amount of data needed for creating a good classifier, the results suggest firstly that the number of attributes should reflect the number of instances in the dataset. E.g., when applying only 100 attributes, the result does not improve when adding more dataset instances. This is probably due to the fact that a larger set of instances also requires a larger set of attributes to discriminate within. However, when the number of attributes is more in proportion to the number of instances considered, the trend is that adding more dataset instances improves the classification.

It is evident from the results, that even the best results are not sufficient for unsupervised automatic classification of sentences between the EHR and the PHR: Slightly more than four out of five sentences are correctly classified, at best. This result will most probably not be sufficient for neither a GP using an EHR nor for a patient using a PHR. However, the accuracy of 81.95% achieved by the CNB classifier may be considered a marked improvement over the majority class baseline classification of 42.9%.

9 History Extraction Experiment Plan (RQ2)

In this part of the project we approach research question two (RQ2), where the goal is to discover whether the developed classifiers may be used to get an overview of the patient history. In particular, we seek to give an overview of how the distribution of sentences of subjective, objective, and plan character changes over time in patient histories. This we intend to do both for individual patient, and for groups of patients, presenting how the average distribution changes over time.

In the following sections we present an overview of preliminary decisions made and work done related to RQ2. Then we seek to explain how we may generate a classifier to use on the D2a dataset, and how this classifier may be evaluated against a classifier built on the D1 dataset. Afterwards we present how we plan to find the distribution of subjective, objective, and plan sentences for different diabetes patient histories from D2a and D2b. An overview of the datasets referred to in this chapter may be found in Chapter 3, while Weka classes referred to are shortly described in Table 6 in Chapter 5.

9.1 Preliminary Decisions and Work

In this section we present some of the decisions made, and work done, prior to being able to execute the experiments in the work of RQ2. First we explain why we choose to focus on the disease of diabetes. Then we present shortly what text classifying techniques and training data we seek to use. Finally we explain how we processed the data from the database, and how the manual classification of some of this data took place.

9.1.1 Selection of Disease

In the work of RQ2 we focus on patient histories from patients suffering from diabetes, as stated in Chapter 2. There are particularly two reasons to focus on the disease of diabetes. Firstly, this disease has a rather chronic nature, giving us the chance to see how the structure of the encounter notes for a given disease changes over time. Secondly, the disease is a relatively frequent disease in the overall population, hence increasing the statistical validity of our findings. This was both deduced by our general impression of this common disease, and the findings in the EHR. In Figure 18 these findings are presented. The graphs present the relation between the number of visits a patient has after the diagnosis of diabetes has been set for the first time and the number of patients that this applies to. The lower graph is created from the D2a dataset, taking only encounter notes related to diabetes into account, while the upper graph is created from the D2b dataset, taking all encounter notes into account after the diagnosis has been set. The graph shows that a total of 429 patients in the EHR have at least one encounter note related to diabetes. This equals 3.6% of the total number of patients in the EHR dataset. Further, we see that 80 patients have at least 30 encounter notes related to this disease. These observations confirm both the frequent occurrence and chronic nature of diabetes.

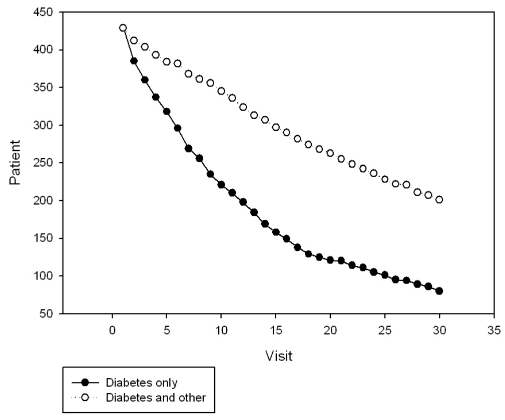


Figure 18: The lower graph presents the number of patients that contribute to the classification averages in D2a, while the upper graph presents the similar for D2b. Hence, the lower graph is relevant for Figure 21(a), and the upper graph is relevant for Figure 21(b).

Through inspection of the ICPC coding system (The Norwegian Directorate for Health and Social Affairs, 2004), we find two ICPC codes that are directly related to the disease of diabetes: T89 (diabetes insulin dependent) and T90 (diabetes non-insulin dependent).

9.1.2 Selection of Text Classifying Techniques

The previous chapter concludes that Complement Naive Bayes (CNB) is the classifier that achieves best results of the classifiers taken into account. It also concludes that preprocessing techniques should be used with caution, due to the problems of finding the correct relation between the number of attributes and the size of the dataset taken into account. Due to the observations in Chapter 8, we apply the CNB classifier without any preprocessing in this latter part of the project.

9.1.3 Selection of Training Data

The classifier created in the previous chapter is constructed with training data from the D1 dataset. Even though the D1 dataset shares many of the characteristics of the datasets D2a and D2b, there are several differences. On the one hand, the D1 dataset contains encounter notes written by *one* specific GP about *all* kinds of diseases. On the other hand, the diabetes data contain encounter notes written by *several* GPs, where the majority of the data is about *one* disease. Although many of the diabetes encounter notes also contain information about other diseases, only notes that contain any of the diabetes ICPC codes T89 and T90 are included in D2a. Hence, the nature of the data differ in two dimensions: The semantic content and the lexical and syntactic way of presenting the data material. Due to this, we find it appropriate to first validate the CNB classifier created from D1 on preclassified data from D2a, and then compare it to a CNB classifier built on manually classified training data from D2a.

Even though D2b consists of other diseases than diabetes to a larger extent than D2a, D2a also consists of a wide range of other diseases, as we will see in Chapter 11. Hence, we do not find that it is worth the effort to classify training data from this dataset in addition to the D2a dataset.

9.1.4 Data Extraction

We extract all encounter notes from patients that suffer from diabetes, as defined by the diabetes ICPC codes presented above to create the D2a and D2b datasets.[20] For D2a, the encounter notes of these patient histories that are not related to diabetes are removed. This is not done for the D2b dataset.

The D2a and D2b datasets are then processed, as in Chapter 6: All punctuation, sentences that consist only of numbers, and "windows newline" are removed. In addition we replace each number in each sentence of the notes by the capital letter "N". This latter transition was not applied in the previous experiments, but is applied now due to the manual observation of a a large amount of numbers in the corpus. We assume that by replacing each of the numbers by this letter, numbers of the same format, e.g., "NNNN" or "NN", that are overrepresented in a class, may be used as a way to discern the notes. These techniques are carried out to ease both further classification and processing. The source code of these operations is shown in Figure 27 in the Appendix. The retrieved data is then stored in the *arff* format, with the class field empty, for further manual classification to create training and test data, that later may be applied by Weka.

[20]Necessary database queries are found in Figure 25 in the Appendix.

9.1.5 Manual Classification

To evaluate both how the classifier created from D1 and how a newly trained classifier may perform on D2a, we need to manually classify a number of instances. In the *E4* experiment, we conducted an experiment where we varied the number of training data instances. When 1,000 attributes were taken into account, the performance of the accuracy seemed to stabilize around use of 3,000 training instances, as seen in Figure 13 in Chapter 7. Hence, due to the human effort of manual classification, and what we have observed in this figure, we choose to use a set of 3,000 instances from D2a to both test the CNB classifier created from D1 and for the training and evaluation with cross validation of a new CNB classifier.

From D2a we select 3,000 random sentences and then classify them manually. For each sentences we pick the most probable class of S, O, and P, and assign it to the sentence. Sentences that mainly express beliefs and thoughts of the patient are classified as subjective (S), while sentences containing measurements confirmed by the GP, or discoveries performed by the GP are classified as objective (O). Equally, sentences expressing assessments about the patient or plans about the further treatment of the patient, such as the number of tablets of a certain medicine a patient should take each day, are classified as plan (P) sentences.

In cases where a sentence consists of several subordinate clauses that may belong to different classes, we split the sentence into two sentences. E.g., some sentences contain first an *observation* of the problem before the *plan* is described. In such cases we split the sentence in two, and then each part of the sentence is assigned to its most probable class.

It may also occur that sentences consist of only one word, or other reasons that prevents us from being able to classify its content. In such cases the sentences are deleted from the training corpus. We expect the sentence splitting and deleting to balance the inclusion or exclusion of sentences, and no further actions are applied to keep the number of sentences in the training corpus at precisely 3,000.

Class	Number	Percentage
S	1094	38.6 %
O	697	24.4 %
P	1061	37.2 %
Total	2852	100.0 %

Table 18: Distribution of the classes subjective (S), objective (O), and plan (P) of the manually classified subset of the D2a dataset.

The manually classified sentences are excluded from D2a, which later is to be used for finding the S, O, and P distribution over time. Experiences from this process are further explained in Chapter 11. In Table 18, the distribution of the manually classified subset of D2a is presented. We now present each of the experiments we seek to carry out in association with RQ2.

9.2 E5: Classifier for Diabetes Data

In the fifth experiment, *E5*, we intend to build a classifier that may classify diabetes specific EHR data. First, we carry out an experiment where we use the best classifier derived from D1, the CNB classifier, to classify the set of manually classified data. We take both the classifier trained on the manually classified subset of D2a through cross validation, and the classifier built on the D1 dataset, into consideration.

There are several aspects that we need to take into consideration when we apply a classifier that is trained on another dataset. First, it is important that the indexed file (explained in Section 4.1) that the classifier is to be created on and the indexed file to be classified, contain the same attributes. I.e., the classifier created from D1 must contain the same attributes as the manually classified data from D2a. Second, to avoid a result biased towards the test set, the selection of the attributes that these sets should have in common should be based on the most frequent terms, from the *training* set. Selecting the most frequent attributes from the *test* set will make the classifier biased towards the test set and make the training set contain many attributes that are not present in any *instances* of the training set.

Because of the importance of letting both the indexed training file and the indexed test file have the same number of attributes, a new classifier has to be built for each attribute set that the classifier is to be tested on. A script we created to test a classifier on the preclassified data is presented in Figure 33. First, *StringToWordVector* is applied to index both the training and test data. Then a classifier is built on the indexed data from D1, and then this classifier is applied on the manually classified data from D2a. The latter operations are carried out by the use of the *ComplementNaiveBayes* class from Weka.

To evaluate a new classifier, we apply the same scripts as in Chapter 6, using cross validation. However, now we apply the manually classified diabetes data from D2a, and not the automatically tagged data from D1. As in Chapter 6, we seek to evaluate the classifiers for the following number of attributes per class: 10, 20, 50, 100, 250, 750, 1,000, 1,250, and 1,500. Finally we compare the accuracy achieved by the classifier trained on the manually classified subset of D2a through cross validation and the classifier built on the D1 dataset.

9.3 E6: Extraction of Graphs from Patient Histories

In the sixth experiment, *E6*, we intend to apply the best classifier created from the classifier evaluation in E5 on data that was not used for manual classification from both D2a and D2b. As above, we retrieve sentences from encounter notes, ignoring the sentences used for manual classification. The stored instances are not assigned any class; this assignment of classes is the task of the classifier.

The retrieved dataset consists of a series of sentences, where no other information is given. To later be able to recreate the structure of each note, other information for each of the sentences in the dataset is stored as well.

After having retrieved the dataset that the classifier is to classify, we apply the Weka *AddClassification* class to add the outcome of the classification and the sentences to a new dataset. The script is reproduced in Figure 34 in the Appendix. However, unlike before, we are now not able to vary the number of attributes to take into consideration, because there is no way to evaluate which number is the best. Taking both the results presented in Chapter 7 and the fact that we have trained 3,000 instances into account, we choose to use 1,000 attributes per class.

The final step is to extract the distribution of the *S*, *O*, and *P* classes for the different patients from the created data files. We would like to be able to extract both the distribution of these classes of the individual patients, and the average distribution of all patients, as specified in Chapter 2. This we seek to do by creating an overview of the distribution of these classes for each of the encounter notes of each of the patients, as presented in Figure 19. Based on such a list, we would be able to calculate[21] both the distribution of these classes of individual patient histories, and the average distribution of all patient histories in addition to its standard deviations.

Patient ID	Note ID	ICPC	S	O	P
00001 ...	01	T90	4	2	4

Table 19: Vectors that show sentence distribution of each relevant encounter note.

From the standard deviations (s) calculated from the data, we may create confidence intervals by

$$\bar{Y} \pm t_{\alpha/2}\frac{s}{\sqrt{n}}, \tag{13}$$

[21]Source code of calculation may be found in Figure 28 in the Appendix.

where n is the number observations made. Calculating the 95% confidence interval, we get $\alpha = 0.05$, and hence $t_{\alpha/2} = 1.96$ (Wadsworth, 1990).

As well as being able to retrieve the distribution of the classes S, O, and P, we would also like to retrieve the distribution of the ICPC codes of the encounter notes, for use in the evaluation of the distribution of the classes. For increased readability of the ICPC figures, we only retrieve ICPC codes that appear in at least 2% of the total number of ICPC codes for a visit number.[22]

9.4 Summary

In this chapter we have presented how we plan to use the Complement Naive Bayes (CNB) classifier to classify sentences from EHR encounter notes and see how the distribution of sentences changes over time, both individually and on average. Table 20 presents an overview of the steps that we seek to carry out to meet these goals. In the next chapter we present the experiments' results.

Exp.	**Sentences**	**Attributes**	**Algorithms**	**Prep.**
E5	From T89/T90 notes	10, 20, 50, 100, 250, 500, 750, 1000, 1250, 1500	CNB old and CNB new	no
E6	From T89/T90 patient histories	1000	CNB	no

Table 20: Overview of the history extraction plan. In E5 we evaluate how different classifiers behave on the diabetes data, while in E6 we use the best classifier to seek to find the distribution of the classes S, O, and P over time.

[22]Source code of calculation reproduced in Figure 29 in the Appendix.

10 History Extraction Results

In this chapter we present the results of the experiments E5 and E6, conducted in the work of RQ2. In comparison with the results presented, the *majority class baseline* accuracy, which is the accuracy achieved when classifying all instances as the majority class, S, achieves an accuracy of 38.6%, as stated in Table 18 in Chapter 9.

First, we look at how the different classifiers behave on the manually classified subset of the D2a dataset. Then we present the result of applying the most appropriate classifier of this experiment on patient histories of patients suffering from diabetes. An overview of the datasets referred to in this chapter may be found in Chapter 3.

10.1 E5: Classifier for Diabetes Data

In Figure 19, extracted from Table 30 in the Appendix, we present the result of the classification of the manually classified diabetes data from the subset of D2a. The lower graph presents the accuracy of the CNB classifier built on the prestructured data from the GP (D1), while the upper graph presents the accuracy of the CNB classifier trained on the manually classified subset of D2a, and evaluated by cross validation.

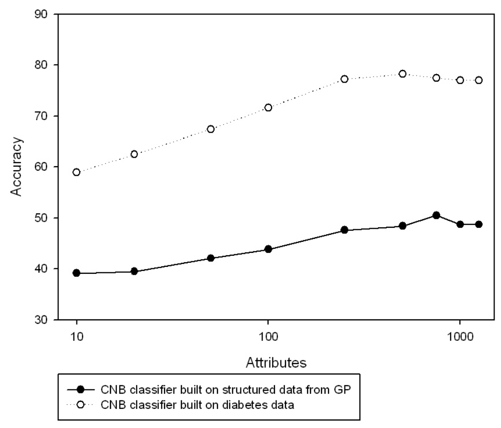


Figure 19: Comparison of the CNB classifier built on D1, and the CNB classifier built on the manually classified subset of D2a.

10.2 E6: Extraction of Graphs from Patient Histories

Based on the results of E5, we choose to apply the CNB classifier trained on the manually classified subset of D2a in E6. A more detailed comparison of the classifiers of E5 is presented in the next chapter.

In Figure 20, extracted from Table 21, we have manually selected a patient whose history of encounter notes contain a typical distribution of the classes *S*, *O*, and *P* the first ten visits after the diabetes diagnosis has been set. The data is extracted from the D2a dataset, hence only containing encounter notes associated with a diabetes diagnosis. The x-axis in the figure presents the number of visits after the diagnosis has been set in addition to the ICPC codes associated with the given encounter note.

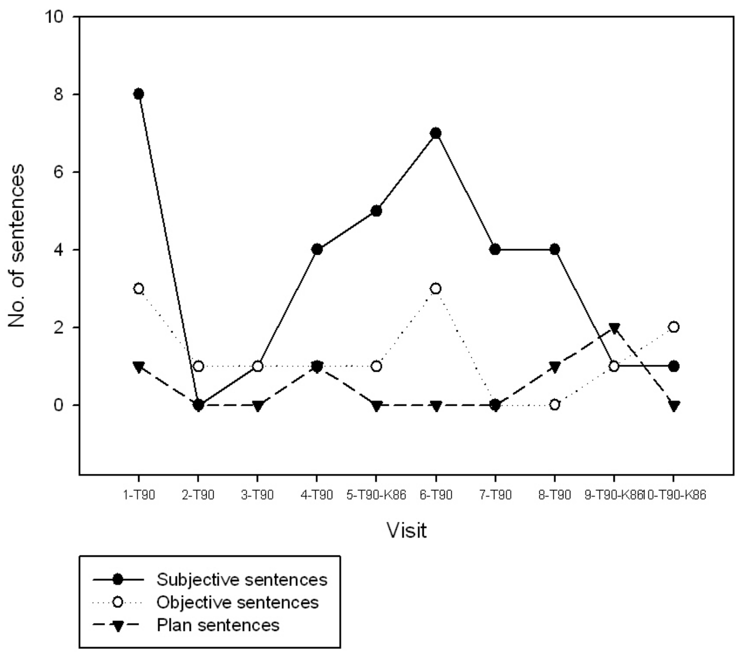


Figure 20: Class distribution for a random patient, from the D2a dataset, for his first ten visits after the diagnosis of diabetes has been set. The x-axis presents the number of visits after the diagnosis has been set in addition to the ICPC codes associated with the given encounter note.

In Figure 21, extracted from Table 31 in the Appendix, we present the average distribution of the classes *S*, *O*, and *P* for the D2a and D2b datasets. In this figure we also present the pertaining 95% confidence intervals. Figure 21(a) is created from D2a and Figure 21(b) is created from D2b. Hence, in the first figure only encounter notes that are associated with a diabetes

Patient ID	Note ID	ICPC	S	O	P
xxxxx	01	T90	8	3	1
	02	T90	0	1	0
	03	T90	1	1	0
	04	T90	4	1	1
	05	T90, K86	5	1	0
	06	T90	7	3	0
	07	T90	4	0	0
	08	T90	4	0	1
	09	T90, K86	1	1	2
	10	T90, K86	1	2	0

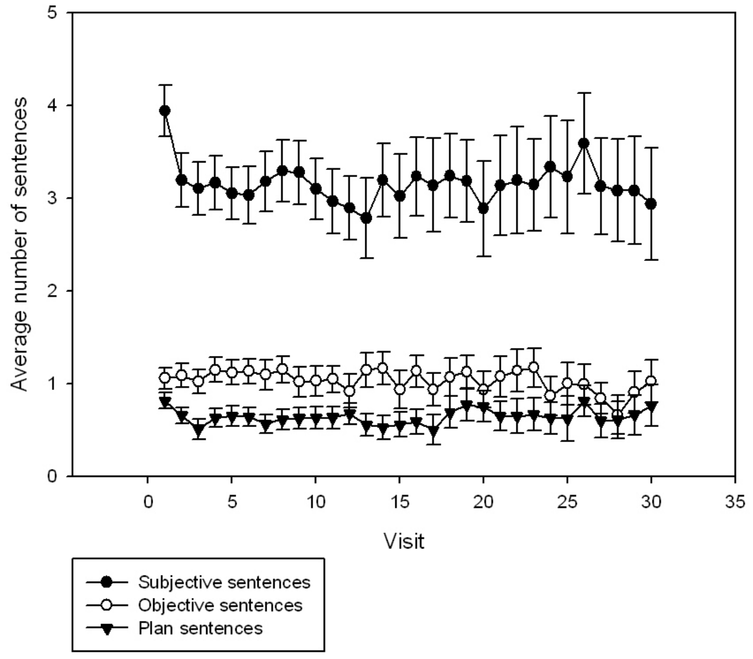
Table 21: Vectors that show sentence distribution of each relevant encounter note, presented in Figure 20. In addition, the ICPC diagnosis codes pertaining to the given encounter notes are presented.

diagnosis code is presented, while in the latter figure all encounter notes after the patient was diagnosed with diabetes are shown. These figures are extracted based on each of the tables for the individual patients, like Table 21.

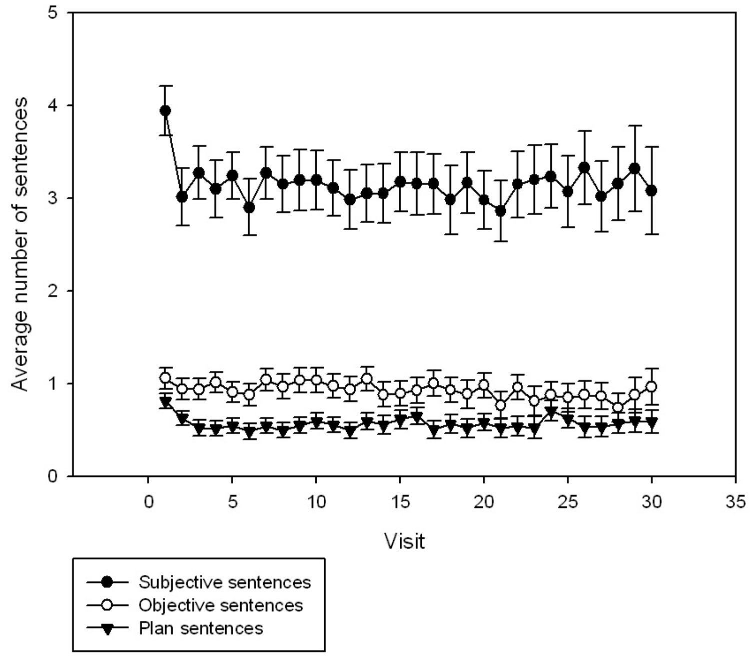
Since Figure 21(b) does not include encounter notes that do not have a pertaining diabetes diagnosis code, there are more patients making up the average scores of the S, O, and P classes for this figure than for Figure 21(a). An overview of the number of patients that contribute to the average values is found in Figure 18 in Chapter 9, where the lower graph shows the number of patients for D2a, and the upper graph shows the number of patients for D2b. One may also note that since the requirements for the first visit is that one of the diagnosis codes of diabetes appear, the averages of the first visits are the same.

10.3 Summary

In this chapter we have presented the results of the experiments presented in Chapter 9. First we have seen how the classifier built on manually classified diabetes encounter notes behave in comparison to the classifier built on general encounter notes on diabetes data. Then we have seen how the distribution of subjective, objective, and plan sentences appear for different kinds of diabetes patient histories using the most accurate classifier for diabetes data. In the next chapter we seek to interpret these results, and suggest some probable reasons for the given outcome.



(a) Diabetes encounter notes only (from D2a).



(b) Diabetes and other encounter notes (from D2b).

Figure 21: Average distribution of the classes *S*, *O*, and *P* given a number of visits.

11 History Extraction Evaluation and Discussion

In this chapter we evaluate the results of the history extraction performed in the work of RQ2. At the end of this chapter we also present some other issues concerning the procedures applied, and a conclusion of the findings.

11.1 E5: Classifier for Diabetes Data

Manual inspection of Figure 29 suggests that the CNB classifier built on the manually classified subset of the D2a dataset produces a higher accuracy than the CNB classifier built on the D1 dataset. In Chapter 8 we found that the CNB algorithm performed *significantly* better than the SVM algorithm. This we even found when the average accuracy achieved by CNB was less than 1% higher than what achieved by SVM. In E5, however, we observe that the smallest difference in achieved accuracy between the two compared algorithms is about 13.25%. Hence, in spite of different sizes of the training datasets, we assume it is safe to conclude that the CNB classifier built on the manually classified subset of D2a performs significantly better than the other, without further investigation of the folds.

11.2 E6: Extraction of Graphs from Patient Histories

Figure 20, in the previous chapter, presents how the distribution of the classes *S*, *O*, and *P* evolve in the history of encounter notes of a manually selected patient. We believe that this presentation of a medical history of a patient may to some degree be helpful both in EHR and PHR settings, where for instance the subjective graph may give an indication of both time and encounter note when the patient went through many changes even though the diagnosis code given was the same. An example may be manual inspection of note "6-T90" in Figure 20, which reveals that at this visit the patient had many patient related subjective experiences about the development of his or her disease. Through graph inspection this point may be revealed in a short matter of time, without having to browse through numerous encounter notes. However, as stated by Tang et al. (2005), there are not only differences in how GPs and patients abstract information, but also between different patients. Hence, information should be presented in a way adequate for the individual.

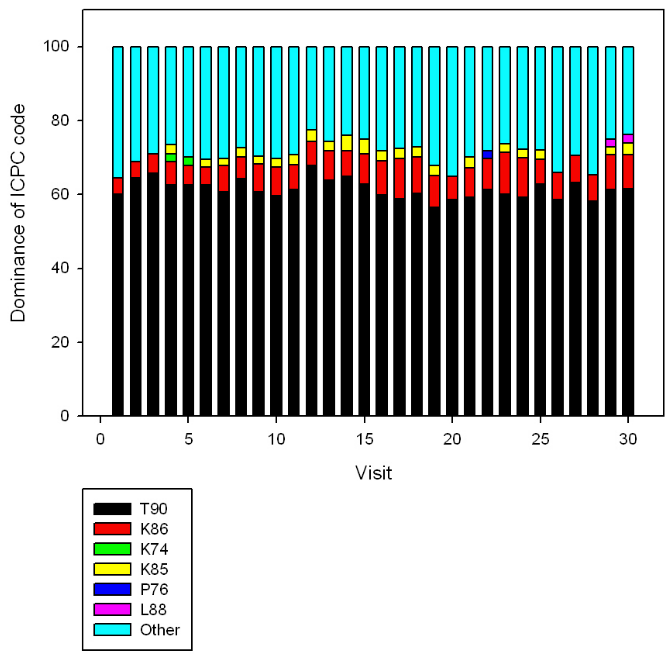
Figure 21, in the previous chapter, presents the average distribution of the classes *S*, *O*, and *P* for the D2a and D2b datasets, and the pertaining 95% confidence intervals. Figure 21(a), extracted from D2a, and Figure 21(b), extracted from D2b, show rather similar distributions for the classes *S*, *O*, and *P*, confidence intervals taken into account. However, all curves are relatively flat. Note that the confidence intervals tend to be larger the higher number of visit one takes into account, due to the decreasing number of patients along the x-axis (see Figure 18). When it comes to Figure 21, both sub figures show that there is in average a significantly higher number of S sentences the first time that the diagnosis of diabetes is set, compared to the rest of the patient history. Afterwards there are no significant changes when it comes to sentences of class S. Secondly, when it comes to objective sentences, no significant difference is shown, since the curves are rather flat. Thirdly, when we consider the plan sentences, the distribution is also quite flat, except from an initial decrease the first three visits. These observations account for both of the graphs presented in Figure 21.

The distribution of ICPC codes for the same encounter notes as reflected upon above is presented in Figure 22, where Figure 22(a) is extracted from the same data as Figure 21(a), and Figure 22(b) is extracted from the same data as Figure 21(b). In Figure 22, we have only included the ICPC codes that appear in more than 2% of the diagnosis codes. Hence, "other" diagnosis codes are ICPC codes that appear in less than 2% of the encounter notes. In Table 22, the different diagnosis codes are shortly described.

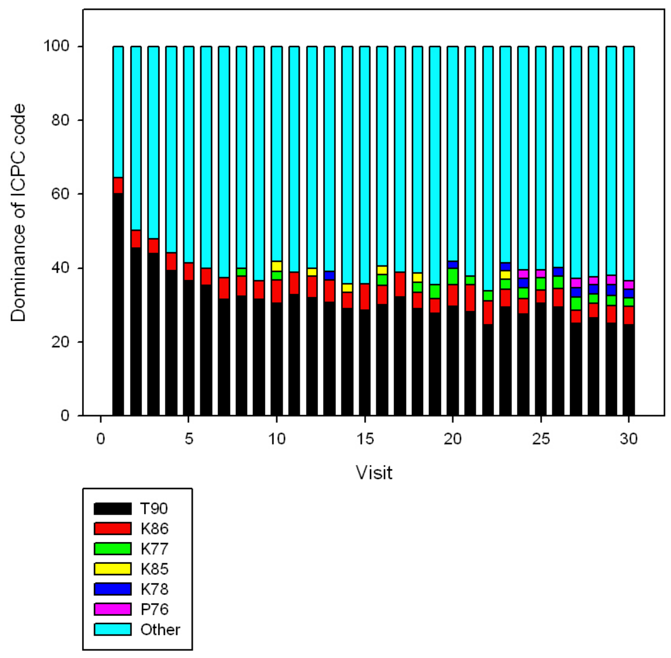
From the overview of the distribution of ICPC codes for the same data, few new conclusions about the distribution of sentences may be drawn. Firstly, in Figure 22(a), that considers the D2a dataset, no change in distribution of any ICPC codes may be seen. Other interesting observations from this figure, however, may be that only 60% of diagnosis codes set in encounter notes dealing with diabetes, are actually related to diabetes, and that many of the encounter notes dealing with diabetes also deal with elevated blood pressure. Secondly, Figure 22(b), that considers the D2b dataset, and hence all encounter notes after the diagnosis code of diabetes is first set, shows that there is a decrease in the percentage of encounter notes dealing with diabetes the six first visits after the first diabetes visit. The same decline may be seen for the number of P sentences in Figure 21(b), without being able to state any connection other than what is stated. We also note that there seems to be an increase in depressive disorders for patients suffering from diabetes. We may suspect that such ICPC codes were supported by an increase in the number of subjective sentences. However, this is not supported by our results. This is probably due to the fact that the graphs are dominated by "other" diagnosis codes, less than 2% of the total, and hence evening out the results.

11.3 Other Issues

Both during manual and automatic classification, we discovered some issues that should be further stressed. We here present the most important ones.



(a) Diabetes encounter notes only (for D2a).



(b) Diabetes and other encounter notes (for D2b).

Figure 22: Average distribution of ICPC codes given a number of visits.

ICPC	Description
L88	Rheumatoid/seropositive arthritis
K74	Ischaemic heart disease w. angina
K77	Heart failure
K78	Atrial fibrillation/flutter
K85	Elevated blood pressure
K86	Hypertension uncomplicated
P76	Depressive disorder
T90	Diabetes non-insulin dependent

Table 22: ICPC codes that appear in the results.

11.3.1 Comparison of the Manually Classified Datasets of D1 and D2a

In Chapter 9 we explained how we carried out the manual classification of the subset of D2a. A difference between the classifiers evaluated above is concerning *how* the data has been classified. While the GP data is structured, and hence indirectly classified by a GP, the diabetes data is classified by us. Hence, some of the sentences that we have classified may have another medical meaning than the one perceived by us, who have limited knowledge about medical terms.

When it comes to the distribution of the classes S, O, and P for the manually classified datasets of D1 and D2a, there are also some differences. In Table 2 in Chapter 3 the distribution of these classes is presented for D1, and in Table 18 in Chapter 9, this distribution is presented for the D2a dataset. The P class, for instance, is some 12 percentage points larger in the D2a dataset than in D1. In addition, the fact that the classifiers built on D1 and D2a behave so differently on the same dataset (as shown in Section 11.1), confirm that there is a considerable difference between the D1 and D2a datasets.

11.3.2 Distribution of Classes in the Classified Dataset

When it comes to the classes S, O, and P, we learn from Figure 21 that the distribution of the classes is rather skewed: A majority of the sentences are classified as S. This is in remarkable contrast to the distribution of the manually classified sentences, presented in Table 18 in Chapter 9. There are many plausible reasons that may explain some of this overrepresentation of the class S. Here we may consider many of the same reasons as described in Section 8.1.1, where we state that the fact that the S class is the largest, makes it even dominate more. First, a classifier that is built from a training set with an overrepresentation of the S class, is assumed to be accordingly better adequate at classifying *S* sentences. Second, in cases where the words of a sentence are not present in the classifier, the sentence is always classified as *S*, since this is the most probable class. Third, through manual inspection we get the impression that the *S* class has a more condensed vocabulary. Words in sentences of this class are hence more likely to get a "match" with words present in the classifier. These considerations apply during the classification of CNB are explained in more detail in Section 8.1.1. However, during manual classification of the training data, no sentences are classified falsely due to mismatch between the words present in the classifier and the sentence or simply because the sentence is classified as the major class because there are no words to help. This may explain the fact that the distribution of the classes of the training data is far more balanced than for the datasets classified by CNB.

11.3.3 Class Features in Diabetes data

In Figure 23, a J48 decision tree is created from the manually classified diabetes data from D2a. We have not preprocessed the data before creating this tree, but to present more relevant attributes we have built the tree containing up to 20 attributes per class. This is carried out by including the 20 most frequent terms from each class *S*, *O*, and *P*. However, to increase readability, only the pruned top section of the figure has been included in this report.

This tree may be compared with both the preprocessed tree presented in Figure 16 and the non-preprocessed tree in Figure 17 in Chapter 11. Taking the attributes of these figures into consideration, we may confirm what we stated earlier: The GP data and the diabetes data are very different. Of the attributes presented in these figures, the only attributes that reappear are "prøve"[23] and "bt"[24].

We also note that the terms "NNN" and "NNNNN" are included in this tree. These terms are representations of sequences of digits of the sizes three and five, respectively, as described in Chapter 9. Hence, we see that the decision tree suggests that the sentence "BT NNNNN" is an objective sentence, which may be considered true, since this sentence probably considers a blood pressure measurement of five digits.

An example of a word that is typically found in plan sentences is "Øker"[25], which is probably an indication of that the GP is suggesting to increase the dosages. A typical word denoting a subjective sentence is "form"[26], which

[23]Eng: test

[24]Norwegian abbreviation: Blodtrykk, Eng: blood pressure

[25]Eng: Increases

[26]Eng: shape

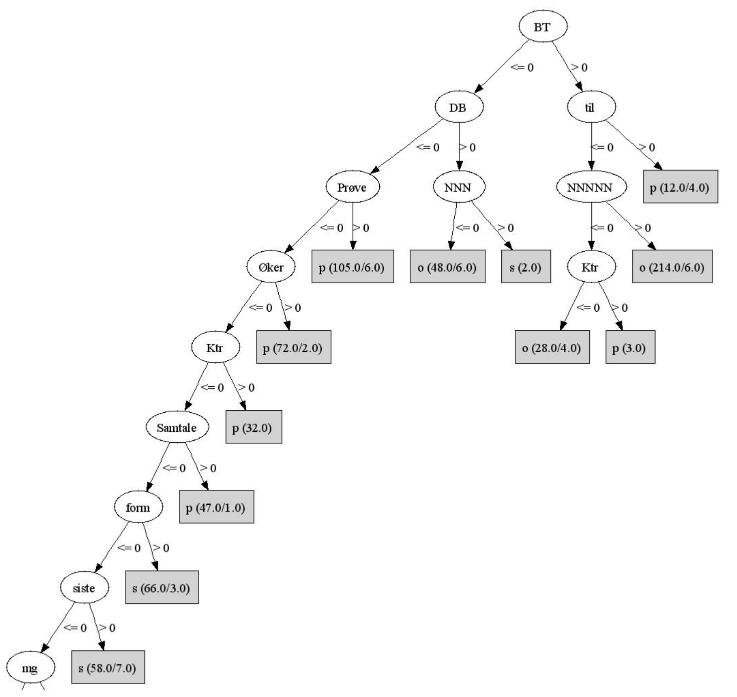


Figure 23: A J48 decision tree on the raw random diabetes dataset with up to 20 attributes per class extracted from the retrieved .dot file in *Graphviz*. Please note that the decision tree has been pruned to increase readability.

probably informally indicates whether a patient feels fine or not.

When describing the incentives for carrying out RQ2, we stated that making overviews of the distributions of different classes would be interesting in a PHR setting as well. We assume, however, that sentences in the EHR are rather different from sentences that may appear in a PHR. For instance, in a PHR setting, one may suspect that the subjective notes are often written in a personal way, like “Tonight I had fever”. In an EHR setting, however, the GP tends to use more formal words like ”febrile“ or “nocturnal”, even though the sentence is in nature describing a subjective experience from the patient. We assume the GP uses other terms than the patient to describe what he or she is expressing. Hence, there is reason to believe that a classifier built for EHR data is rather different than a classifier built for PHR data, but that most text mining techniques used in this project are applicable in both systems.

11.4 Conclusion

From the evaluation of the classifiers we may conclude that there is a big difference between classifiers built on different kinds of EHR data. This encourages the creation of new classifiers when two datasets are written by different authors or the diseases in focus are of different character, rather than reusing classifiers built on slightly different data. This process may be automated, and hence one may picture that individual classifiers may be applied for each author and/or topic to classify the information in the best way possible.

Presenting the distribution of sentences of the classes *S*, *O*, and *P*, may disclose information about the treatment of individual patients, however, no empiric investigation is carried out. When it comes to average distribution of such sentences for the disease of diabetes, we were only able to find very limited characteristics about diabetes, the disease in focus. However, it would be interesting to see what this average distribution is like for other diseases, to discover possible trends in groups of diseases.

12 Conclusion

This project has focused on two main research questions (RQs). In RQ1 we investigated how classification of sentences in encounter notes of EHRs may facilitate a more structured exchange of information between EHRs and PHRs. In the encounter notes we considered the sentences of the character subjective (S), objective (O), and plan (P). In our work we experimented with several preprocessing techniques, classifiers, and amounts of data. Concerning this RQ, we found that the classifying algorithm of Complement Naive Bayes (CNB) produced the best result, both when preprocessing of the data had taken place and not. On the raw dataset, CNB yielded an accuracy of 81.03%, while on the preprocessed dataset, CNB yielded an accuracy of 81.95%. The Support Vector Machine classifier achieved results comparable to these, but the J48 decision tree algorithm performed considerably poorer.

In general, we found that the more data that was applied as training data, the more accurate results the classifiers achieved. However, when it comes to the use of preprocessing techniques, the results do not speak for themselves. Techniques reducing the dimensionality of the datasets seemed to improve the results for smaller attribute sets, but worsen the results for larger attribute sets. The trend was opposite for preprocessing techniques that expanded the set of attributes. However, finding an adequate ratio between the number of attributes and the size of the dataset is difficult. Hence, in the work of RQ2 we applied the classifiers without preprocessing.

We conclude that even the best results of the classification are not good enough for unsupervised automatic classification of sentences between the EHR and the PHR. However, taking the majority class baseline classification of 42.9% into account, the accuracy of 81.95% achieved by the CNB classifier may be considered a marked improvement.

When evaluating different classifiers for use in RQ2, we discovered that classifiers built on different portions of the EHR data behave very differently. We assume that new classifiers should be created for portions of EHR data that are either written by different authors or that focus on different diseases.

In RQ2 we further used the best classifier derived from the work of RQ1, CNB, to classify encounter notes in sickness histories of patients suffering from diabetes. First, we managed to extract graphs that show how the distribution of sentences of the classes S, O, and P evolve over time for individual patients. We believe further work in this field may facilitate the identification of special points of interests in individual patient histories.

We then found how the average distribution of the classes S, O, and P evolve over time for patients suffering from diabetes. From the derived graphs little may be concluded, except that there is a high number of subjective sentences when the diagnosis of diabetes is first made. However, we believe that similar experiments for several diseases may uncover patterns or trends concerning the diseases in focus.

13 Further Work

Through the work presented in this report, we have discovered several areas that require further research:

PHR data classification: In this project we have presented ways that information in the EHR and the PHR may be structured to ease information flow between these systems. To make the flow of information possible it is important that classifiers taking PHR data into account are developed as well.

Handling spelling errors: We noted that there are quite a few spelling errors in the corpus. These errors may degrade the result. First, techniques that handle such errors should be implemented, and second, techniques that discern between such errors and common abbreviations (typically not present in a dictionary) should also be implemented.

Groups of preprocessing techniques: In this project we have applied several preprocessing techniques. The techniques have either been applied solo, or all together. A topic of further research would be to evaluate how different preprocessing techniques may influence each other, and thus how they appear in tandem, groups of three and so on.

Stopword validation: We suspect that several of the words that are part of the stopwords in reality have a discriminating effect. An example of such a word may be “om”[27], which we assume may be overrepresented in the plan class. Hence, each stopword should be validated to confirm its class indifference.

Feature extraction: In the report we showed that the terms “hun_ha”[28] and “han_ha”[29] are typical for the subjective class. As a mean of dimensionality reduction, we suggest to replace “han” and “hun” in this setting with for example “pp” (personal pronoun). Applying domain knowledge to combine multiple features together may probably be done for several types of terms in the corpus, reducing the set of attributes.

Encounter note anatomy: When we classify the sentences of subjective, objective or plan character, we do not take into account the position of the sentence in the encounter note. We assume, however, that sentences of the different classes are to a large extent written among other sentences belonging to the same class in the encounter note. Hence, we assume that the surroundings of a sentence may be utilized to decide the class of a sentence, as described in Section 8.4.1.

[27]Eng: if, in (time)
[28]Eng: She_to_have
[29]Eng: He_to_have

Further classification: It would be interesting to further classify the result of the classification of the sentences *S*, *O*, and *P*. E.g., observations may be further classified into what are observations made by the GP, and what are actual measurements. Plan sentences may equally be classified into planned medical treatment, and planned treatment based on change in behaviour of the patient. These are only examples, and maybe the use of clustering techniques is most appropriate, being able to disclose groups in the data, unbiased by human beliefs.

Other diseases than diabetes: It would also be interesting to make graphs that show the average distribution of the classes *S*, *O*, and *P* for other diseases than diabetes. Equally, other semantic groupings than the *SOAP* standard may be considered, also decided by preliminary clustering techniques.

Other time intervals: In the graphs presented in Chapter 9 we have used "visit" as time intervals. However, it is likely that using another time interval, like actual time since last visit, could have produced another result.

Other algorithms: It is a fact that we have only applied a small subset of all algorithms capable of performing text classification. Hence, the use of other algorithms may give other results than the ones we have obtained. In addition, several classifiers may be combined to create *meta* classifiers, that *together* may decide what the most likely outcome of a classification is.

Mining in sequences: We have not used any data mining technique to evaluate how the distribution of the classes in focus evolve over time. There are techniques, like the *Apriori All* (Agrawal and Srikant, 1995), based on the association rule algorithm *Apriori* (Agrawal et al., 1993), that may make this possible. Apriori All has the opportunity of finding *sequential association rules*, i.e., rules from the data that statistically occur more often than others. One such rule may be that persons who buy a computer, then tend to buy a digital photo camera, and then a memory card to this camera. Similarly, such rules may disclose patterns in EHRs or PHRs, concerning both ICPC codes or the notes structure itself.

References

Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216. Accessible: `portal.acm.org/citation.cfm?id=170036` [28.05.2007].

Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In Yu, P. S. and Chen, A. S. P., editors, *Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan. IEEE Computer Society Press. Accessible: `citeseer.ist.psu.edu/agrawal95mining.html` [28.05.2007].

Andreassen, H., Sandaune, A.-G., Gammon, D., and Hjortdahl, P. (2002). Nordmenns bruk av helsetilbud på Internett. *Tidsskrift for Den norske lægeforening*, (17/2002):1640–1644. Accessible: `www.tidsskriftet.no/lts-pdf/pdf2002/1640-4.pdf` [28.05.2007].

Andreassen, H., Trondsen, M., Kummervold, P. E., Gammon, D., and Hjortdahl, P. (2006). Patients Who Use E-Mediated Communication With Their Doctor: New Constructions of Trust in the Patient-Doctor Relationship. *Qualitative health research*, (2/2006):238–248. Accessible: `qhr.sagepub.com/cgi/reprint/16/2/238` [28.05.2007].

Baerheim, A. (2002). The diagnostic process in general practice: has it a two-phase structure? *Family Practice*, 18(3):243–245. Accessible: `fampra.oxfordjournals.org/cgi/reprint/18/3/243` [30.05.2007].

Baeza-Yates, R. A. and Ribeiro-Netoziri, B. (1999). *Modern Information Retrieval*. Addison-Wesley, Boston, MA, USA.

Ball, M. J. and Lillis, J. (2000). E-health: transforming the physician/patient relationship. *International Journal of Medical Informatics*, (61/2001):1–10.

Bemmel, J. H. V. and Musen, M. A. (1997). *Handbook of Medical Informatics*. Springer, Heidelberg, Germany.

Brage, S., Bentsen, B. G., Bjerkedal, T., Nygård, J. F., and Tellnes, G. (1996). ICPC as a standard classification in Norway. *Family Practice*, 13(4/1996):391–396. Accessible: `fampra.oxfordjournals.org/cgi/reprint/13/4/391` [30.05.2007].

Brasethvik, T. and Kofod-Petersen, A. (2006). Eigenjournal: A personal collaborative medical journal. In *Proceedings of the 1st International Workshop on Health Pervasive Systems*, pages 52–56, Lyon, France. IEEE Computer Society Press.

Crawford, W. (2006). Personally Controlled Health Records: The US Landscape and the Indivo Project. In Kofod-Petersen, A. and Brasethvik, T., editors, *Proceedings of the International Symposium on Personal Electronic Health Records (ISePHR 2006)*, number 06/2006, pages 1–4, Trondheim, Norway. Department of Computer and Information Science, NTNU.

da Silva, C. F., Vieira, R., Osório, F. S., and Quaresma, P. (2004). Mining linguistically interpreted texts. In *Proceedings of the 20th International Conference on Computational Linguistics*, Geneva, Switzerland. COLING organizers. Accessible: `www.coli.uni-saarland.de/conf/linc-04/silva.pdf` [29.05.2007].

de Freitas Vale, R., Ribeiro-Neto, B. A., de Lima, L. R. S., Laender, A. H. F., and Freitas-Junior, H. R. (2003). Improving text retrieval in medical collections through automatic categorization. In *SPIRE*, pages 197–210. Accessible: `www.springerlink.com/content/2dmxmwtv1d3qvjbk` [14.03.2007].

Dumais, S. T. and Chen, H. (2000). Hierarchical classification of Web content. In Belkin, N. J., Ingwersen, P., and Leong, M.-K., editors, *Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval*, pages 256–263, Athens, GR. ACM Press, New York, US.

Fan, R.-E., Chen, P.-H., and Lin, C.-J. (2005). Working Set Selection Using Second Order Information for Training Support Vector Machines. *Journal of Machine Learning Research*, 6:1889–1918. Accessible: www.csie.ntu.edu.tw/ cjlin/papers/quadworkset.pdf [30.05.2007].

Fowler, M. (2003). *UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd ed.* Addison-Wesley, Boston, MA, USA.

Frakes, W. B. and Baeza-Yates, R. A. (1992). *Information Retrieval: Data Structures and Algorithms.* Prentice Hall, NJ, USA.

Han, Jiaweiand Kamber, M. (2006). *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, second edition.

Hersh, W. R., Leen, T. K., Rehfuss, P. S., and Malveau, S. (1998). Automatic Prediction of Trauma Registry Procedure Codes from Emergency Room Dictations. *Medinfo*, (9):665–9. Accessible: `citeseer.ist.psu.edu/hersh98automatic.html[14.06.2007]`.

Hsu, C.-W., Chang, C.-C., and Lin, C.-J. (2003). A practical guide to support vector classification. Department of Computer Science and Information Engineering, National Taiwan University. Accessible: `www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf` [30.05.2007].

Jain, G., Ginawala, A., and Aslandogan, Y. A. (2004). An approach to text classification using dimensionality reduction and combination of classifiers. In *Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration, IRI - 2004, November 8.10, 2004*, pages 564–569, Las Vegas, NV, USA. Accessible: `ieeexplore.ieee.org/iel5/9790/30875/01431521.pdf` [30.05.2007].

Joachims, T. (1998). Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In Nédellec, C. and Rouveirol, C., editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398, pages 137–142, Chemnitz, DE. Springer Verlag, Heidelberg, DE. Accessible: `citeseer.ist.psu.edu/joachims97text.html` [30.05.2007].

Johannes, F. (1998). A Study Using n-gram Features for Text Categorization, Technical Report OEFAI-TR-9830. Austrian Institute for Artificial Intelligence. Accessible: `citeseer.ist.psu.edu/johannes98study.html` [28.05.2007].

John, G. H. and Langley, P. (1995). Estimating Continuous Distributions in Bayesian Classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, San Mateo, CA, USA. Accessible: `citeseer.ist.psu.edu/john95estimating.html` [28.05.2007].

Kienzle, W., Wichmann, F. A., Scholkopf, B., and Franz, M. O. (2006). Learning an interest operator from human eye movements. In *CVPRW '06: Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop*, page 24, Washington, DC, USA. IEEE Computer Society. Accessible: `portal.acm.org/citation.cfm?id=1153773` [30.05.2007].

Letrilliart, L., Viboud, C., Boelle, P.-Y., and Flahault, A. (2000). Automatic Coding of Reasons for Hospital Referral from General Medicine Free-Text Reports. In *Proc AMIA Symp.*, pages 487–91. Accessible `www.amia.org/pubs/symposia/D200436.PDF` [29.05.2007].

Lewis, D. D. (1992). An Evaluation of Phrasal and Clustered Representations on a Text Categorization Task. In *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 37–50, New York, NY, USA. ACM Press.

Long, W. (2006). Extracting Diagnoses from Discharge Summaries. In *AMIA 2005 Symposium Proceedings*, pages 470–4, DC, USA.

Markle Foundation (2003). Connecting for Health, A Public-Private Collaborative. Accessible: www.markle.org/downloadable_assets/final_phwg_report1.pdf [14.03.2007].

McCallum, A. and Nigam, K. (1998). A Comparison of Event Models for Naive Bayes Text Classification. In *AAAI-98 Workshop on Learning for Text Categorization.*, WI, USA. Accessible: citeseer.ist.psu.edu/mccallum98comparison.html [30.05.2007].

Mladenic, D. and Globelnik, M. (1998). Word Sequences as Features in Text Learning. In *Proceedings of the 17th Electrotechnical and Computer Science Conference (ERK98)*, Ljubljana, Slovenia. Accessible: citeseer.ist.psu.edu/mladenic98word.html [14.06.2007].

Nilsson, G., Åhlfeldt, H., and Strender, L.-E. (2003). Textual content, health problems and diagnostic codes in electronic patient records in general practice. *Scand J Prim Health Care*, 21(1):33–36. Accessible: www.ingentaconnect.com/content/tandf/spri/2003/00000021/00000001/art00009 [28.05.2007].

Nordgård, T. (1996). NORKOMPLEKS. Some Linguistic Specifications and Applications. In Lindebjerg, Ore and Reigem, editor, *ALLC-ACH '96. Abstracts*, pages 214–216. Humanistisk Datasenter, Universitetet i Bergen. Accessible: gandalf.aksis.uib.no/allc/nordgaar.pdf [29.05.2007].

Norwegian Ministry of Social Affairs and Norwegian Ministry of Health (2004). S@mspill 2007. Accessible: www.shdir.no/samspill [30.05.2007].

Pavlov, D., Balasubramanyan, R., Dom, B., Kapur, S., and Parikh, J. (2004). Document Preprocessing for Naive Bayes Classification and Clustering with Mixture of Multinomials. In *KDD 2004*, pages 829–834. Accessible: citeseer.ist.psu.edu/pavlov04document.html [30.05.2007].

Plaisant, C., Milash, B., Rose, A., Widoff, S., and Shneiderman, B. (1996). Lifelines: Visualizing personal histories. In *CHI*, pages 221–227, Vancouver, BC, Canada. Accessible: citeseer.ist.psu.edu/plaisant96lifelines.html [28.05.2007].

Pomikálek, J. and Rehurek, R. (2007). The Influence of Preprocessing Parameters on Text Categorization. *International Journal of Applied Science, Engineering and Technology*, 4(1):430–433. Accessible: www.enformatika.org/ijaset/v4/v4-1-82.pdf [14.06.2007],.

Quinlan, J. R. (1993). *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Rennie, J., Shih, L., Teevan, J., and Karger, D. (2003). Tackling the poor assumptions of Naive Bayes text classifiers. In Fawcett, T. and N, M., editors, *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, pages 616–623, DC, USA. AAAI Press. Accessible: `citeseer.ist.psu.edu/710402.html` [30.05.2007].

Riva, A., Mandl, K. D., Oh, D. H., Nigrin, D. J., Butte, A., Szolvits, P., and Kohane, I. S. (2000). The Personal Internetworked Notary and Guardian. *International Journal of Medical Informatics*, (62/2001):27–40.

Rose, O. (2006). Modell for tilbakerapportering på behandlingsplaner i egenjournal. Project as part of Master's Degree in Computer Science at Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU).

Røst, T. B., Nytrø, O., and Grimsmo, A. (2006a). Classifying Encounter Notes in the Primary Care Patient Record. In *Workshop on Text and Information Retrieval (TIR06) at ECAI06.*, pages 1–5, Verona, Italy.

Røst, T. B., Nytrø, O., and Grimsmo, A. (2006b). Investigating the Value of Diagnosis Codes in the Primary Care Patient Record. In Peek, N. and Combi, C., editors, *Proceedings of IDAMAP 06 Intelligent Data Analysis in Biomedicine and Pharmacology; 2006*, pages 93–8, Verona, Italy.

Salzberg, S. (1997). On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach. *Data Mining and Knowledge Discovery*, 1(3):317–328. Accessible: `www.citeseer.ist.psu.edu/salzberg97comparing.html` [14.06.2007].

Sebastiani, F. (2002). Machine Learning in Automated Text Categorization. *ACM Computing Surveys*, 34(1):1–47. Accessible: `citeseer.ist.psu.edu/sebastiani02machine.html` [14.06.2007].

Sebastiani, F. (2005). Text categorization. In Zanasi, A., editor, *Text Mining and its Applications to Intelligence, CRM and Knowledge Management*, pages 109–129, Southampton, UK. WIT Press. Accessible: `citeseer.ist.psu.edu/sebastiani05text.html` [14.06.2007].

Sibanda, T., He, T., Szlolvits, P., and Uzner, O. (2006). Syntactically-Informed Semantic Category Recognizer for Discharge Summaries. In *Proceedings of the Fall Symposium of the American Medical Informatics Association*, DC, USA. Accessible: `people.csail.mit.edu/ozlem/uzuner-amia2006.pdf` [28.05.2007].

Stone, A. A., Shiffman, S., Schwartz, J. E., Broderick, J. E., and Hufford, M. R. (2002). Patient non-compliance with paper diaries. *British Medical Journal*, (324):1193–4. Accessible `bmj.bmjjournals.com/cgi/reprint/324/7347/1193` [30.05.2007].

Szolovits, P., Doyle, J., Long, W. J., Kohane, I., and Pauker, S. G. (1994). Guardian Angel: Patient-Centered Health Information Systems. Technical report, Massachusetts Institute of Technology Laboratory for Computer Science, Boston, MA, USA. Accessible `groups.csail.mit.edu/medg/projects/ga/manifesto/GAtr.pdf` [30.05.2007].

Tang, P. C., Ash, J. S., Bates, D. W., Overhage, J. M., and Sands, D. Z. (2004). The PING Personally Controlled Electronic Medical Record System: Technical Architecture. *J Am Med Inform Assoc.*, 12(1):47–54. Accessible: `www.jamia.org/cgi/reprint/12/1/47` [30.05.2007].

Tang, P. C., Ash, J. S., Bates, D. W., Overhage, J. M., and Sands, D. Z. (2005). Personal Health Records: Definitions, Benefits, and Strategies for Overcoming Barriers to Adoption. *J Am Med Inform Assoc.*, 13(2):121–6. Accessible: `www.jamia.org/cgi/reprint/13/2/121` [30.05.2007].

The Norwegian Directorate for Health and Social Affairs (2004). *Den internasjonale klassifikasjonen for primærhelsetjenesten, ICPC-2.* Accessible: `www.kith.no/upload/1895/ICPC-2-bok_110304.pdf` [28.05.2007].

Tzeras, K. and Hartmann, S. (1993). Automatic indexing based on Bayesian inference networks. In Korfhage, R., Rasmussen, E., and Willett, P., editors, *Proceedings of SIGIR-93, 16th ACM International Conference on Research and Development in Information Retrieval*, pages 22–34, Pittsburgh, PA, USA. ACM Press, New York, NY, USA.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory.* Springer-Verlag, New York, NY, USA.

Wadsworth, H. M. (1990). *Handbook of Statistical Methods for Engineers and Scientists.* McGraw-Hill, New York, NY, USA.

Walker, J., Pan, E., Jognston, D., Adler-Milstein, J., Bates, D. W., and Middleton, B. (2005). The Value of Health Care Information Exchange and Interoperability. *Health Affairs*, (January 19):11–18. Accessible: `content.healthaffairs.org/cgi/reprint/hlthaff.w5.10v1` [30.05.2007].

Wilcox, A. B. and Hripcsak, G. (2003). The Role of Domain Knowledge in Automating Medical Text Report Classification. *J Am Med Inform Assoc.*, 10(4):330–8. Accessible: `www.jamia.org/cgi/content/full/10/4/330` [30.05.2007].

Witten, I. H. and Frank, E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Yang, Y. and Liu, X. (1999). A re-examination of text categorization methods. In *22nd Annual International SIGIR*, pages 42–9, San Francisco, CA, USA.

Appendices

A Abbreviations

A	Assessment
CNB	Complement Naive Bayes
Dn	Dataset n
EHR	Electronic Health Record
En	Experiment n
GP	General Practitioner
ICD	International Classification of Diseases
ICPC	International Classification for Primary Care
NB	Naive Bayes
NSEP	Norsk Senter for Elektronisk Pasientjournal
O	Objective
P	Plan
PHR	Personal Health Record
RQn	Research Question n
S	Subjective
SVM	Support Vector Machine
UML	Unified Modeling Language
Weka	Waikato Environment for Knowledge Analysis
WHO	World Health Organization

Table 23: List of abbreviations used in the report.

B Standards

B.1 Binomial Distribution

n	k left	P	k right
10	0	0.0010	10
	1	0.0107	9
	2	0.0547	8
	3	0.1719	7
	4	0.3770	6
	5	0.6230	5

Table 24: Binomial distribution table where $n = 10$ and $\theta = 0.5$. Table is extracted from Wadsworth (1990).

B.2 Arff format

```
% ARFF file for subjective and objective thoughts, and planned
% treatment.

@relation sop
@attribute class {s, o, p}
@attribute note string

@data
s, 'my foot hurts'
o, 'fever is 37.9'
p, 'take two tablets of paracet three times a day'
p, ...
```

Figure 24: Example of an *arff* file for subjective thoughts (S), objective observations (O) and planned treatment (P).

C Stopwords

List of stopwords presented in Table 25 is reproduced from `http://snowball.tartarus.org/index.php`.

Table 25: List of stopwords.

Norwegian	English	Norwegian	English
og	and	kan	can
i	in	hans	his
jeg	I	hvor	where
det	it/this/that	eller	or
at	to (w. inf.)	hva	what
en	a/an	skal	shall/must
et	a/an	selv	self (reflective)
den	it/this/that	sjøl	self (reflective)
til	to	her	here
er	is/am/are	alle	all
som	who/that	vil	will
på	on	bli	become
de	they / you	ble	became
med	with	blei	became *
han	he	blitt	have become
av	of	kunne	could
ikke	not	inn	in
ikkje	not *	når	when
der	there	være	be
så	so	kom	come
var	was/were	noen	some
meg	me	noe	some
seg	you	ville	would
men	but	dere	you
ett	one	som	who/which/that
har	have	deres	their/theirs
om	about	kun	only/just
vi	we	ja	yes
min	my	etter	after
mitt	my	ned	down
ha	have	skulle	should
hadde	had	denne	this
hun	she	for	for/because
nå	now	deg	you

Continued on next page

Table 25 – continued from previous page

Norwegian	English	Norwegian	English
over	over	si	hers/his
da	when/as	sine	hers/his
ved	by/know	sitt	hers/his
fra	from	mot	against
du	you	å	to
ut	out	meget	much
sin	your	hvorfor	why
dem	them	dette	this
oss	us	disse	these/those
opp	up	uten	without
man	you/one	hvordan	how
ingen	none	ho	she *
din	your	hoe	she *
ditt	your	henne	her
blir	become	hennar	her/hers
samme	same	hennes	hers
hvilken	which	hoss	how *
hvilke	which (plural)	hossen	how *
sånn	such a	ikkje	not *
inni	inside/within	ingi	noone *
mellom	between	inkje	noone *
vår	our	korleis	how *
hver	each	korso	how *
hvem	who	kva	what/which *
vors	us/ours	kvar	where *
hvis	whose	kvarhelst	where *
både	both	kven	who/whom *
bare	only/just	kvi	why *
enn	than	kvifor	why *
fordi	as/because	me	we *
før	before	medan	while *
mange	many	mi	my *
også	also	mine	my *
slik	just	mykje	much *
vært	been	no	now *
være	to be	nokon	some (masc./neut.) *
båe	both *	noka	some (fem.) *
begge	both	nokor	some *
siden	since	noko	some *
dykk	your *	nokre	some *

Continued on next page

Table 25 – continued from previous page

Norwegian	English	Norwegian	English
dykkar	yours *	si	his/hers *
dei	they *	sia	since *
deira	them *	sidan	since *
deires	theirs *	so	so *
deim	them *	somt	some *
di	your (fem.) *	somme	some *
då	as/when *	um	about*
eg	I *	upp	up *
ein	a/an *	vere	be *
eit	a/an *	vore	was *
eitt	a/an *	verte	become *
elles	or *	vort	become *
honom	he *	varte	became *
hjå	at *	vart	became *

D Code

D.1 Database calls

```
def getWriterNotes(self, writer_id):
  """
  Get all (pat_id, contact_id, note_id, block_num, block_txt)
  tuples from database
  """
  return self._executeSQL('select contact.pat_id, contact.contact_id, ' +
    'note.note_id, note_block.block_num, note_block.block_txt ' +
    'from patient, contact, note, note_block where ' +
    'contact.contact_id=note.contact_id and note_block.note_id = ' +
    'note.note_id and patient.pat_id = contact.pat_id and ' +
    'note.writer_id = %i' % writer_id)

def getPatientsWithICPCCodes(self, icpc_codes):
  """
  Get all pat_id for a set of icpc codes.
  """
  result = self._executeSQL('select distinct contact.pat_id from contact' +
    'where contact.contact_id in (select distinct diagcont.contact_id' +
    'from diagcont, diagnosis where diagcont.diag_id = diagnosis.diag_id' +
    'and diagnosis.diag_code in %s)' % ("('%s')" % "', '".join(icpc_codes)))
  return [int(x[0]) for x in result]

def getPatientData(self, patient_ids):
  """
  Get all (pat_id, contact_id, cont_ts, diag_code, note_id,
   block_num, block_txt) tuples
  for patients with the given ids.
  """
  sql = 'select contact.pat_id, contact.contact_id, contact.cont_ts, ' + \
        'diagnosis.diag_code, note.note_id, note_block.block_num, ' + \
        'note_block.block_txt from patient, contact, diagcont, diagnosis, ' + \
        'note, note_block where patient.pat_id = contact.pat_id and ' + \
        'contact.contact_id = note.contact_id and ' + \
        'contact.contact_id = diagcont.contact_id and ' + \
        'diagcont.diag_id = diagnosis.diag_id and ' + \
        'note.note_id = note_block.note_id and ' + \
        'contact.pat_id in (%s)' % str(patient_ids)[1:-1]
  result = self._executeSQL(sql)
```

Figure 25: Database calls used to retrieve the information necessary in the work of RQ1 and RQ2.

D.2 Python preprocessing code

```
# For each of the paragraphs classify the sentences
for i, paragraph in enumerate(self.paragraphs):
  sentences = self._split_sentences(paragraph)
  for sentence in sentences:
    j = i + 1
    if i > 2:
      j = 3

    if stem:
      sentence = u"%s '%s'" % (self.SENTENCE_TYPE_CLASS[j], \
       ' '.join(sentence.tokens(stem)))
    else:
      sentence = u"%s '%s'" % (self.SENTENCE_TYPE_CLASS[j], sentence.text)
    result.append(sentence)
```

Figure 26: Tagging the sentences in a paragraph.

```
def arff_sentences(note, out_filename):
  """
  Get ARFF representation of each sentence. I.e. ?, string:
    'Patient feels pain' => "? 'Patient feels pain'"
  """
  result = []
  numbers = range(10);

  #For sentence in paragraph
  for sentence in note.sentences():

    #Remove punctuation
    sentence = "".join([c for c in sentence.text if c not \
     in string.punctuation])

    #Remove sentences that are only digits
    if not (sentence.isdigit()):

      #Replace numbers with N
      for number in numbers:
        sentence=sentence.replace(str(number), "N")

      #Remove windows newline
      sentence=sentence.replace("\n", " ")
      sentence=sentence.replace("\r", " ")

      #Append result
      result.append(u"%s, '%s'" % ("?", sentence))
  return result
```

Figure 27: Sentence preprocessing.

```
# For each retrieved sentence
for i, line in enumerate(infolist):
  basedir_len = len(basedir.split('/'))

  # Do not retrieve new note info if sentence is from same note
  if not (line == oldline or not oldline) or i == infolist_len - 1:
    nofNotes+=1

    # Set variables for note and add result
    infolistwords = oldline.split('/')
    patient_id = infolistwords[basedir_len+1]
    noteinfo = infolistwords[basedir_len+2].split('_')
    note_id = noteinfo[3]
    diagnosis_code = noteinfo[1]
    result.append(u"%s, %s, %s, %s, %s, %s" % \
     (patient_id, note_id, diagnosis_code, s_tmp, o_tmp, p_tmp))

    # Set visit to 0 and no relevant ICPC code detected if new patient
    if (patient_id != old_patient_id):
      j=0
      firstDiabNote = False
      old_patient_id = patient_id
      nofPatients+=1
    if (j==0 and (diagnosis_code.find("T90")!=-1 or \
        (diagnosis_code.find("T89")!=-1):
        firstDiabNote = True

    # Add statistical information for note number
    if (firstDiabNote == True):
      if (j < historylength):
        s[j]+=s_tmp
        o[j]+=o_tmp
        p[j]+=p_tmp
        nofPat[j]+=1
        for code in diagnosis_code.split('-'):
          if code in icpc[j]:
            icpc[j][code]+=1
          else:
            icpc[j][code]=1
        j+=1

    # Set tmp counters to zero
    s_tmp = 0
    o_tmp = 0
    p_tmp = 0

# Find and add type of sentence
type = getType(datalist[i])
if(type == 's'):
  s_tmp+=1
elif (type == 'o'):
  o_tmp+=1
else:
  p_tmp+=1
oldline = line
```

Figure 28: Getting the number of sentences belonging to the classes S, O, and P for each note in a patient history.

```
# Calculate average for ICPC codes
icpc_output = [""]*historylength
for i, visit in enumerate(icpc):

  # Find tot. no. of ICPC codes per visit to calculate average
  tot_nof_icpc = 0
  for codes in icpc[i]:
    tot_nof_icpc+=icpc[i][codes]

  # Only storing the ICPC codes that appear in at least 2% of the codes
  expose_threshold = 0.02
  for codes in icpc[i]:
    if (float(icpc[i][codes])/tot_nof_icpc>expose_threshold):
      icpc_output[i]=icpc_output[i]+' '+codes+': \
       '+str(round(float(icpc[i][codes])/tot_nof_icpc, 3))+'.'
```

Figure 29: Calculating the average distribution for the ICPC codes, where we only take the ICPC codes that appear in at least 2% of the total number of codes into account.

D.3 Weka Addition

```
--
 * <pre> -G &lt;int&gt;       <
 *  The number of n-grams (default = 1).</pre>      <
 *       <
--
  /**       <
   * The default number n-grams       <
   */       <
  private int g_nofNGrams = 1;       <
       <
    result.addElement(new Option(
     "\tNumber of n-grams (default = 1).",      <
     "G", 1, "-G <int>"));       <
--
   *       <
   *  <pre> -G &lt;int&gt;       <
   *  The number of n-grams (default = 1).</pre>      <
   *       <
--
    value = Utils.getOption('G', options);       <
    if (value.length() != 0)       <
      setNofNGrams(Integer.valueOf(value).intValue());       <
    else       <
      setNofNGrams(g_nofNGrams);       <
--
   * Gets the number of words (per class if there is a class  <
   * assigned) to attempt to keep.       <
   *       <
   * @return the target number of words in the output vector  <
   * assigned).       <
   */       <
  public int getNofNGrams() {       <
    return g_nofNGrams;       <
  }       <
       <
  /**       <
   * Sets the n-grams       <
   *       <
   * @param nofGrams       <
   * vector (per class if assigned).       <
   */       <
  public void setNofNGrams(int nofNGrams) {       <
    g_nofNGrams = nofNGrams;       <
  }       <
       <
 --
```

Figure 30: Additions in Weka to make it possible to add n-grams from GUI or command line. The figure is extracted by use of the “diff -y” command in UNIX, and then the irrelevant data is manually removed.

```
 --
    // Tokenize all training text into an orderedMap of "word
    for (int i = 0; i < getInputFormat().numInstances(); i++)
      Instance instance = getInputFormat().instance(i);
      int vInd = 0;
      if (!m_doNotOperateOnPerClassBasis && (classInd != -1))
vInd = (int)instance.classValue();
      }

      // Iterate through all relevant string attributes of th
      Hashtable h = new Hashtable();
      for (int j = 0; j < instance.numAttributes(); j++) {
        if (m_SelectedRange.isInRange(j) && (instance.isMissi

  // Get tokenizer
          Enumeration st;
          if(this.m_onlyAlphabeticTokens==false)
              st = new StringTokenizer(instance.stringValue(j
                                                    m_Delimite
          else
              st = new AlphabeticStringTokenizer(instance.str

          int nofGrams = getNofNGrams();       <
          String[] gramList = new String[nofGrams];       <
      <
          while (st.hasMoreElements()) {
            for(int k = gramList.length-1; k>0; k <
   gramList[k] = gramLis <
   }       <
   gramList[0] = ((String)st.nex <
int k = 0;       <
word = gramList[0];       <
     // Create the n-gram for inclusion
     while(k!=gramList.length&&gramList[k]!=null)  <
     if (k!=0) {       <
word = gramList[k]+"_ <
}       <
k++;       <
      <
      <
            if(!(h.contains(word)))       <
                h.put(word, new Integer(0));       <
      <
      Count count = (Count)dictionary <
      if (count == null) {       <
        dictionaryArr[vInd].put(word, <
      } else {       <

       count.count ++;       |
      }       |
```

Figure 31: Additions in Weka to make it possible to calculate n-grams. Note that the upper part of the figure presents code that was present in *String-ToWordVector* prior to the addition of n-grams, but is added for contextual reasons. The figure is extracted by use of the "diff -y" command in UNIX, and then the irrelevant data is manually removed.

D.4 Clssification Scripts

```
#!/bin/sh
clear

#
# Classify a dataset with cross validation,
# - performing tf-idf weighting
# - creating n-grams
# - removing stopwords
#
# Parameters:
# -----------
# $1 = training data (that the classifier is to be built upon)
# $2 = indexed training data
# $3 = number of attributes to take into consideration
# $4 = n, in n-grams to include
#

echo "Running all preprocessing techniques "
java weka.filters.unsupervised.attribute.StringToWordVector \
-i $1 -o $2 -c 1 -W $3 -G $4 -T -I -S

java -Xmx2048m weka.classifiers.bayes.ComplementNaiveBayes \
-t tmp/dataset/indexed.arff -c 1 -x 10 -s 1 -i
```

Figure 32: Script that performs the preprocessing techniques of removing stopwords, adding n-grams and tf-idf weighting, before it performs Complement Naive Bayes classification with ten-fold cross validation.

```
#!/bin/sh
clear

#
# Create a classifier
#
# Parameters:
# -----------
# $1 = training data (that the classifier is to be built upon)
# $2 = indexed training data
# $3 = test data
# $4 = indexed test data
# $5 = number of n-grams to include
# $6 = number of terms to include
# $7 = classifier model to be made

echo "Indexing manually classified sentences, and sentences"
echo "to be classified..."
java weka.filters.unsupervised.attribute.StringToWordVector \
-b -i $1 -o $2 -r $3 -s $4 -W $5 -G $6 -c 1

echo "Making a ComplementNaiveBayes classifier model..."
time java -Xmx2048m weka.classifiers.bayes.ComplementNaiveBayes \
-t $2 -c 1 -x 10 -s 1 \
-d $7

echo "Using the model to classify data..."
time java -Xmx2048m weka.classifiers.bayes.ComplementNaiveBayes \
-T $4 -c 1 -x 10 -s 1 \
-l $7
```

Figure 33: Script that creates a new complement naive Bayes classifier.

```
#!/bin/sh
clear

#
# Add classified classes to arff file
#
# Parameters:
# -----------
# $1 = indexed test data arff file
# $2 = reordered test data arff file
# $3 = classifier
# $4 = classified arff file

echo "Reordering attributes, putting the classifier attribute"
echo "in the end of the file, to make it adequate for AddClassification..."
java weka.filters.unsupervised.attribute.Reorder \
  -R 2-last,1 -i $1 -o $2

echo "Classifying unclassified data, based on input model..."
java -Xmx2048m weka.filters.supervised.attribute.AddClassification \
-serialized $3 -classification \
-remove-old-class -i $2 -o $4 -c last
```

Figure 34: Script that adds a classification label to the text.

E Result Data Values

Table 26: Data values obtained in E1.

(a) F_{ma} and accuracy.

Attrib.	Accuracy			F_{ma}		
	SVM	**J48**	**CNB**	**SVM**	**J48**	**CNB**
10	59.35	58.75	54.37	0.551	0.541	0.498
20	62.70	62.22	59.19	0.598	0.592	0.566
50	70.00	66.68	66.91	0.689	0.653	0.652
100	72.92	66.95	71.82	0.715	0.655	0.706
250	75.68	67.35	77.03	0.741	0.658	0.760
500	77.13	67.32	79.40	0.757	0.657	0.785
750	78.46	67.20	80.50	0.773	0.656	0.796
1000	79.93	67.16	80.92	0.790	0.656	0.799
1250	80.15	67.25	81.03	0.793	0.657	0.801
1500	75.75	67.23	80.84	0.721	0.656	0.797

(b) Precision and recall.

Attrib.	Precision			Recall		
	SVM	**J48**	**CNB**	**SVM**	**J48**	**CNB**
10	0.596	0.591	0.564	0.556	0.549	0.497
20	0.621	0.621	0.598	0.596	0.591	0.559
50	0.694	0.652	0.676	0.640	0.654	0.644
100	0.716	0.654	0.719	0.714	0.656	0.700
250	0.758	0.659	0.765	0.734	0.657	0.759
500	0.773	0.659	0.787	0.751	0.657	0.785
750	0.777	0.658	0.797	0.771	0.665	0.798
1000	0.790	0.657	0.800	0.793	0.655	0.801
1250	0.792	0.658	0.801	0.795	0.656	0.803
1500	0.760	0.665	0.798	0.719	0.656	0.802

Table 27: Data values obtained in E2.

(a) F_{ma} and accuracy.

Attrib.	Accuracy			F_{ma}		
	SVM	**J48**	**CNB**	**SVM**	**J48**	**CNB**
10	61.65	61.41	61.51	0.567	0.565	0.572
20	64.54	64.20	64.61	0.610	0.602	0.614
50	68.77	66.22	69.59	0.665	0.632	0.676
100	72.51	66.29	72.88	0.708	0.636	0.715
250	75.22	66.26	76.71	0.737	0.635	0.756
500	77.36	66.37	78.69	0.760	0.637	0.777
750	77.58	66.17	79.86	0.763	0.634	0.790
1000	78.16	66.15	79.90	0.768	0.634	0.790
1250	77.15	66.15	80.53	0.756	0.634	0.797
1500	74.37	66.19	81.56	0.713	0.635	0.808
1750			81.56			0.808
2000			81.95			0.811
2250			76.16			0.737

(b) Precision and recall.

Attrib.	Precision			Recall		
	SVM	**J48**	**CNB**	**SVM**	**J48**	**CNB**
10	0.703	0.708	0.683	0.560	0.558	0.563
20	0.704	0.721	0.694	0.598	0.590	0.602
50	0.724	0.718	0.725	0.652	0.618	0.664
100	0.749	0.689	0.741	0.696	0.624	0.707
250	0.758	0.687	0.767	0.729	0.623	0.753
500	0.777	0.690	0.783	0.752	0.625	0.777
750	0.778	0.689	0.792	0.756	0.623	0.792
1000	0.784	0.688	0.792	0.760	0.623	0.792
1250	0.774	0.688	0.797	0.748	0.623	0.800
1500	0.754	0.689	0.807	0.704	0.623	0.811
1750			0.807			0.811
2000			0.811			0.814
2250			0.745			0.734

Table 28: Accuracy for each of the folds in E1 and E2.

(a) E1 with 1,250 attributes.

Fold	0	1	2	3	4
SVM	80.02	80.97	80.58	80.44	80.73
J48	68.04	68.25	67.66	69.19	67.50
CNB	80.48	80.85	81.45	81.15	81.06

Fold	5	6	7	8	9
SVM	80.00	79.27	79.22	79.56	80.18
J48	67.72	68.03	67.15	67.54	67.22
CNB	80.57	80.51	81.06	80.99	80.94

(b) E2 with 1,250 attributes.

Fold	0	1	2	3	4
SVM	78.61	76.96	78.17	76.52	76.90
J48	68.67	65.84	66.37	65.66	64.25
CNB	80.87	80.55	80.60	81.35	80.37

Fold	5	6	7	8	9
SVM	76.08	77.22	76.89	76.57	76.60
J48	64.42	63.83	65.96	68.44	68.09
CNB	80.09	80.41	79.89	80.87	81.21

Fold	0	1	2	3	4
CNB	79.81	79.11	79.51	80.26	80.10
CNB prep.	81.24	81.50	81.54	82.08	81.19

Fold	5	6	7	8	9
CNB	80.21	80.47	79.79	79.11	80.05
CNB prep	82.11	82.38	81.43	81.75	82.12

Table 29: The accuracy of each of the fold of CNB in E2 for the preprocessed and the raw dataset when then number of attributes is set equal to 2,000.

Attrib.	CNB on D1	CNB on D2a
10	42.00	57.50
20	33.85	60.40
50	30.10	66.15
100	36.20	68.20
250	33.85	72.85
500	34.35	73.90
750	59.30	72.55
1000	59.30	72.55

Table 30: Data values obtained in E5.

Visit	D2a			D2b		
	S	**O**	**P**	**S**	**O**	**P**
1	3.94	1.06	0.81	3.94	1.06	0.81
2	3.19	1.09	0.66	3.01	0.94	0.63
3	3.10	1.02	0.51	3.27	0.94	0.52
4	3.17	1.15	0.63	3.10	1.01	0.51
5	3.05	1.12	0.65	3.24	0.91	0.55
6	3.03	1.13	0.64	2.90	0.88	0.48
7	3.18	1.10	0.57	3.27	1.04	0.54
8	3.29	1.15	0.61	3.15	0.97	0.50
9	3.28	1.02	0.63	3.19	1.03	0.55
10	3.10	1.03	0.63	3.19	1.03	0.59
11	2.97	1.05	0.63	3.11	0.97	0.56
12	2.89	0.92	0.68	2.98	0.94	0.49
13	2.78	1.15	0.55	3.05	1.05	0.59
14	3.20	1.17	0.53	3.05	0.88	0.56
15	3.02	0.94	0.56	3.18	0.89	0.62
16	3.23	1.13	0.59	3.16	0.93	0.65
17	3.14	0.93	0.50	3.15	1.00	0.50
18	3.24	1.07	0.69	2.98	0.93	0.57
19	3.18	1.13	0.77	3.16	0.88	0.52
20	2.88	0.93	0.75	2.98	0.98	0.58
21	3.13	1.08	0.65	2.86	0.76	0.52
22	3.19	1.14	0.65	3.15	0.96	0.54
23	3.14	1.17	0.67	3.20	0.81	0.52
24	3.33	0.87	0.63	3.23	0.88	0.71
25	3.23	1.00	0.62	3.07	0.85	0.62
26	3.59	0.99	0.81	3.33	0.88	0.53
27	3.13	0.84	0.60	3.02	0.86	0.53
28	3.08	0.66	0.61	3.16	0.74	0.57
29	3.08	0.91	0.66	3.32	0.87	0.60
30	2.94	1.03	0.76	3.08	0.97	0.59

Table 31: Data values obtained in E6.