Aalto University
School of Science
Life Science Technologies

Eetu Pursiainen

General Purpose Text Classifier for Finnish Medical Texts

Master's Thesis
Espoo, October 23rd, 2017

Supervisor: Professor Juho Rousu, Aalto University
Advisors: D.Sc. (Tech.) Hanna Heikkinen, HUS
M.Sc. (Tech) Teijo Konttila, HUS

Aalto University
School of Science
Life Science Technologies

ABSTRACT OF
MASTER'S THESIS

Author:	Eetu Pursiainen		
Title:	General Purpose Text Classifier for Finnish Medical Texts		
Date:	October 23rd, 2017	**Pages:**	56
Major:	Biomedical Engineering	**Code:**	T-110
Supervisor:	Professor Juho Rousu, Aalto University		
Advisors:	D.Sc. (Tech.) Hanna Heikkinen M.Sc. (Tech.) Teijo Konttila		

Medical texts are an underused source of data in clinical analytics. Extracting the relevant information from unstructured texts is difficult and while there are some tools available, they are often targeted for English texts. The situation is worse for smaller languages, such as Finnish.

In this work, we reviewed literature in text mining and natural language processing fields in the scope of analyzing medical texts. Using the results of our literature review, we created an algorithm for information extraction from patient record texts.

During this thesis work we created a decent text mining tool that works through text classification. This algorithm can be used detect medical conditions solely from medical texts. The usage of the algorithm is limited through the availability of large training data.

Keywords:	text mining, natural language processing, electronic health records, patient records
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Biotieteiden teknologioiden maisteriohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Eetu Pursiainen		
Työn nimi:	Yleiskäyttöinen tekstinluokittelija suomenkielisille potilaskertomusteksteille		
Päiväys:	23. lokakuuta 2017	**Sivumäärä:**	56
Pääine:	Lääketieteellinen tekniikka	**Koodi:**	T-110
Valvoja:	Professori Juho Rousu, Aalto-yliopisto		
Ohjaajat:	TkT Hanna Heikkinen DI Teijo Konttila		

Potilaskertomustekstejä käytetään kliinisessä analytiikassa huomattavan vähäisessä määrin. Olennaisen tiedon poimiminen tekstin joukosta on vaikeaa, ja vaikka siihen on työkaluja saatavilla, ovat ne useimmiten tehty englanninkielisille teksteille. Pienempien kielten, kuten suomen kohdalla tilanne on heikompi.

Tässä työssä tehtiin kirjallisuuskatsaus tekstinlouhintaan ja luonnollisen kielen käsittelyyn liittyvään kirjallisuuteen, keskittyen erityisesti menetelmiin jotka soveltuvat lääketieteellisten tekstien analysointiin. Kirjallisuuskatsauksen pohjalta loimme algoritmin, joka soveltuu yleisesti lääketieteellisten tekstien luokitteluun.

Tämän diplomityön osana luotiin tekstinlouhintatyökalu suomenkielisille lääketieteellisille teksteille. Kehitettyä algoritmia voidaan käyttää erilaisten tilojen tunnistamiseen potilaskertomusteksteistä. Algoritmin käyttöä kuitenkin rajoittaa tarve suurehkolle määrälle opetusdataa.

Asiasanat:	tekstinlouhinta, luonnollisen kielen käsittely, potilaskertomus, potilastiedot
Kieli:	Englanti

Acknowledgements

This thesis is a product of a long lasting process. I would like to thank everyone who has been a part of this journey.

Especially I want to thank my supervisor, professor Juho Rousu for guiding me to the right direction when I was looking for a master's thesis topic. I also want to thank my advisors, Teijo Konttila and Hanna Heikkinen for guidance during this thesis work, and for a great deal of other interesting information of health care data in general.

I also want to thank my friends and family for all the support and great adventures during my studies. I think this thesis could have been written years ago, if it weren't for some awesome distractions I encountered. For these, I want to thank the Guild of Physics and AYY, and particularly everyone in PaRaati and AYYH12. Finally, thank you Rissittely, for always having the answers to any and all questions one might have.

Helsinki, October 23rd, 2017

Eetu Pursiainen

Abbreviations and Acronyms

We are trying to use abbreviations and acronyms sparingly in this thesis. In some cases, however, it is impractical to do so. Therefore, we provide a list of all the abbreviation used in this work, extended with common abbreviations and acronyms used in text mining literature.

API	Application Programming Interface
AUC	Area Under Curve
BOW	Bag of Words
CART	Classification and Regression Tree
CBOW	Continuous Bag of Words
CDS	Clinical Decision Support
DM	Data Mining
EHR	Electronic Health (also Hospital) Record
FPR	False positive rate, also known as fall-out
HIT	Health Information Technology
HUS	The Hospital District of Helsinki and Uusimaa
ICD	International Statistical Classification of Diseases and Related Health Problems
IDF	Inverse Document Frequency
IE	Information Extraction
IR	Information Retrieval
JVM	Java Virtual Machine
kNN	K Nearest Neighbours
LDA	Latent Dirichlet Allocation
LSA	Latent Semantic Analysis
MeSH	Medical Subject Headings
ML	Machine Learning
MLP	Medical Language Processing
NEN	Named Entity Normalization
NER	Named Entity Recognition
NLP	Natural Language Processing
PLSA	Probabilistic Latent Semantic Analysis

POS	Part-of-Speech
POST	Part-of-Speech Tagging
RDR	Ripple Down Rule
RF	Random Forest
ROC	Receiver Operating Characteristics
SOV	Subject-Object-Verb
SVO	Subject-Verb-Object
TF	Term Frequency
TF-IDF	Term Frequency - Inverse Document Frequency
TM	Text Mining
TPR	True positive rate, also known as sensitivity
VSO	Verb-Subject-Object

Contents

Chapter 1

Introduction

Clinical texts are the most abundant type of health care data. They come in different shapes and forms, such as admission notes, treatment plans and patient summaries. Patient case books date back to at least the early 18th century.[11] Some medical professionals wrote these case books strictly for their own use, while some wrote them for teaching purposes. Modern, electronic clinical records are used for both patient care and research.

Finnish hospitals have kept a detailed record of patients' care, and they have been stored electronically since 2007. Each and every visit is recorded at some level. There are large quantities of information about and for patients' care, such as patient record texts. Additionally, data has been collected for billing purposes, without restricting it's use for medical research. A good example are ICD-10-codes, that are used to label different conditions that a patient might have.

Clinical texts are a flexible and fast way to save information about the patients. All it requires from the underlying patient record management system, is an ability to save free text. Free text is often unstructured, giving doctors the possibility to describe patients' conditions with as much or little detail as is needed. However, there may also be structural information within the free text as well.

It has been shown that using information extracted from clinical texts, one can achieve better data-analysis results than by using structured data only. For example, LePendu et al.[21] showed that extraction of adverse drug effects from clinical notes showed increased risk of myocardial infraction with specific medication, when analyzing the structured ICD-data did not.

1.1 Problem statement

Today we are used to analyzing large quantities of data with computers. Growth of computational power has enabled us to do very extensive analysis of practically anything, as long as our data is somewhat well structured. This, unfortunately, is not the case with text data, not in general and not in the field of clinical texts.

There exists a lot of structured clinical data, but it doesn't contain all the information that a doctor has collected about a patient. A large portion of this information is only stored as free text. When a researcher needs some information about a patient or a patient group, they will first check if it is available in structural form. When it isn't, often the only possible way to get the information is to read all those patient record texts. A single patient can have hundreds of texts, meaning that studying larger patient groups quickly becomes infeasible, if not entirely impossible.

The information encoded in medical texts is mainly unstructured. Sentences are generally short, do not exactly follow grammar, have a high number of abbreviations and contain relatively high amount of spelling mistakes.[13] Combining this with the fact that practically every language has a wide variety of ways to encode the same information in text format, raises many challenges in automated information extraction.

In this work, we will develop tools for information extraction and patient classification purposes, to ease the workload of doctors and researchers. The aim is to create an algorithm that could be used as a baseline for multiple different text mining scenarios, instead of creating a single purpose algorithm that performs well in a specific task and is unusable in others. An example of the latter would be rule-based systems, that are strictly tailored for a single purpose, that do not work well in other tasks.

1.2 Structure of the thesis

First, we will cover background of text mining approaches, with main focus in algorithms and techniques that are suitable for Finnish medical texts. The literature review will go through some state-of-the art methods, as well as the most used methods in this field. Main findings of this literature review are discussed in chapter 2.

In chapter 3 we will briefly discuss the environment in our project. This includes general knowledge of Hospital District of Helsinki and Uusimaa, our datasets, and software used.

After conducting the literature review, we selected some methods to be

used in our text mining algorithm. These methods are explained in chapter 4. The implementation and reasoning behind some design choices are also discussed in this chapter.

You will find evaluation and validation of the developed algorithm in chapter 5 and finally discussion and conclusions in chapters 6 and 7.

Chapter 2

Background

2.1 Text mining in general

Text mining is a broad subject, as it can mean extracting information, comparing texts, classifying texts, using efficient search algorithms for large text masses, summarizing texts and more. Text mining is closely related to natural language processing, but the latter may also mean producing new texts, translating texts, and also analysis of spoken language. In this thesis, we will focus on information extraction from clinical data that exists in electronic text formats, mainly patient and care reports.

Text can be analyzed and mined in many ways, and more often than not, any analysis requires many different text mining tools and techniques, instead of using one single tool to do everything. In the following sections we will go through some techniques that are often used in the field of text mining and natural language processing. Some of these tasks are general preprocessing tasks that have many alternative approaches such as sentence splitting. Other tasks are more specific ones, that usually are performed on already preprocessed data.

Text mining and natural language processing algorithms can be domain-specific or domain-independent. Generally, it is easier to create domain specific methods, as the vocabulary and variety of different expressions to handle remain smaller. The same applies, in an even larger scale, for different languages.

2.2 Terminology

In this chapter we will go through some terminology that is commonly used in linguistics and therefore also in text mining. It is important to under-

stand different terms, as they are often used to describe what a text mining algorithm does or tries to do.

Corpus (plural: *corpora*) is a collection of texts, usually somehow related to each other. For example a collection of patient records form a corpus, that could be used for text mining purposes, as well as other research. There are different kinds of corpora, but in text mining especially *annotated corpora* are useful. A person has read these texts and any interesting properties defined prior to the annotation are marked down.

Lexicon (plural: *lexica*) is the vocabulary of a particular language, field, social class, person, etc. There can be different types of lexica, not only describing the meaning of the word, but also other possible features, like frequency in a specific corpus, word's possible part-of-speech tags, etc.

Named entities are entities that have been given a specific name. These include names of people, places, companies, etc. In Finnish language named entities are written with a capital letter, but they can also be referenced to in more ambiguous ways.

Noun phrases are phrases that have a noun as their head word. Most common examples include a noun that is described with some adjectives. Noun phrases are often parsed from the text so that they can be handled as a single element instead of multiple words.

Ontology is a study of what something is. It aims at describing the features and composition of things. Ontological classification is more complex than taxonomic (see below), and it is usually more of a network than hierarchical structure.

Semantics is the study of meaning of the words. The purpose is to extract meanings of the text. It is not rare that a literal phrase has many meanings, but through semantic context analysis it should become clear which interpretation was meant.

Syntax defines the rules of language, and is closely related to grammar. It depicts how we can correctly use different words within a sentence or a document. Syntactic analysis tries to find these syntactic rules that can be further used to analyze texts.

Taxonomy depicts the relations of different words. For example, words 'dog' and 'cat' could be taxonomically classified below 'domestic animals', and related to each other through that taxonomical tree. Taxonomies are not generally trees, but networks with directed edges.

2.3 Finnish language

Finnish is a language mainly spoken in Finland. There are around 4.9 million people who speak Finnish as their first language, and over 0.5 million people who speak Finnish as a second language in Finland.[49] Additionally, there are hundreds of thousands of Finnish speakers outside of Finland.[16]

Finnish is a highly inflected and agglutinative language, meaning that Finnish language uses case endings to express plurality, grammatical cases, verb tenses and other aspects of the language. In addition to the case endings, agglutination means that the meaning of the words can be altered through morphological affixes.

In Finnish language there is no fixed word order, but the syntax often follows subject-verb-object (SVO) structure. There are 15 grammatical cases for substantives. Finnish is written with extended Latin alphabet that consists of 29 letters, some of which are only used in loanwords, and share the same phonemes with other letters.

The distribution of Finnish words follows the power law (also known as Zipf's law in this context). In other words, the n^{th} most frequent word has a frequency

$$f(n) \propto \frac{1}{n^{\alpha}}, \tag{2.1}$$

where it has been found out that constant $\alpha \approx 1$. Zipf's law applies to all known human languages.[34] This behaviour in languages was first found by Estoup in 1916 and it was studied in depth and confirmed by Zipf in 1949[54]. Word frequencies of Finnish words in PAROLE corpus have been plotted in figure 2.1. PAROLE project's aim was to create large general written language resources for all EU languages.[8]

Word distributions rarely follow Zipf's law exactly, and a more accurate relation was depicted by Mandelbrot who later got famous from his fractal studies. The more accurate distribution was achieved by shifting the word rank in the power law so that

$$f(n) = \frac{1}{(n+\beta)^{\alpha}}. \tag{2.2}$$

It has been argued, that both these power laws are too general, and that they do not accurately depict the underlying mechanisms of a language.[39]

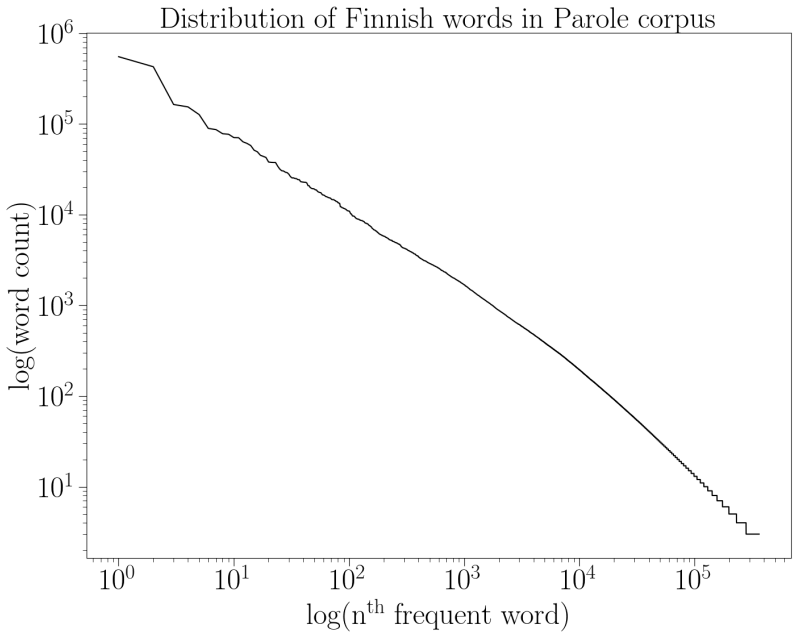

Figure 2.1: This log-log plot shows how the distribution of Finnish words follow Zipf's law. This distribution has been calculated from Finnish PAROLE corpus. The word densities have been truncated to four decimals before taking the logarithm, causing the 'stepping' in the image.

When considering text mining tasks, this does not really matter. It can be useful to know the approximate word distribution when considering word frequency normalization, for example.

2.4 Common preprosessing tasks

2.4.1 Sentence segmenting

Sentence segmenting or sentence splitting, as the names suggest, splits a text into sentences. The beginning and the end of the sentences usually have well defined characteristics that can be used for segmenting. There are, however, exceptions for this. In more informal texts the grammar may not be followed and capital letters and punctuation may not be present. It is also not rare for texts to have list structures, or some other type of formatting that omits usual sentence boundary rules.

In practice, there are two ways to do sentence segmenting. First one is by creating a set of rules to determine sentence boundaries. A very simple rule set searches for a sentence ending character such as a period, exclamation mark or question mark, which need to be followed by a space and a capital letter. A simple tokenizer like this will get approximately 95% of the sentences correct[35]. Aberdeen et al.[1] describe an advanced rule-based sentence splitter, the Alembic information extraction system in their work. Alembic's sentence boundary recognition module achieved an error rate of 0.9%. The alternative method for rule-based sentence splitting is using some machine learning algorithm with an annotated corpus to learn the sentence boundaries. Riley reported to have achieved an impressive accuracy of 99.8% with his tree-based model in 1989.[41]

From the accuracies of the selected algorithms, we can conclude that sentence segmenting is a text mining sub-task that can be done with very high accuracy. There are many programming libraries that provide sentence segmenting, and we will be using one in this work as well.

2.4.2 Tokenization

Tokenization, or lexical analysis is a process where a text is split into smaller sequences. In theory, they can be phrases, words, single letters, or any meaningful entity depending on the algorithms used. In practice, tokenization is often used to split the text into single words, numbers or values. Even though there are many ways to do tokenization, there isn't much scientific literature

available on the subject. In this section we will briefly discuss some methods and other considerations.

There are practically unlimited number of different tokenization techniques, because any text can be split into arbitrarily many smaller pieces. While most human readers would consider splitting a sentence into single words as the most natural way of tokenization, it does not mean that tokenization should always be done to the word level. For example, Kamps et al. compared some n-gram (see n-gram definition in section 2.6.3) based tokenization techniques in the context of multilingual information retrieval.[17], They showed that the choice of tokenization can have a large impact on results. For example, including the last 3 characters of the words in addition to the 4 character n-gram gave better results than having 4 character n-grams and the complete words as tokens.

The simplest approach for tokenization is to use white space as a delimiter. This works reasonably well for many texts, but it has its limitations.[28] Such tokenizer is not very good for handling compound words or named entities (”New York”, ”General Electric” etc.), for example. Whitespace tokenizers do not handle hyphenation too well either.

The choice of tokenization should depend greatly on the use case. In many tasks, one needs to output words, instead of seemingly random character combinations. If that is the case, a simple whitespace tokenizer might be all that is needed. There are many other considerations as well, such as how to handle punctuation marks. They can be considered as their own tokens, for example, or omitted entirely.

2.4.3 Stemming

Stemming is a process where different inflicted versions of a single word are reduced to a shorter form by cutting them from a selected location. This shorter form does not need to be the base or dictionary form of the word, but it can be. Stemming is done so that algorithms would be able to handle all the inflicted forms of a word as a single word, as they often have similar or exactly the same meaning. There are exceptions for this, however. For example, Finnish words ”tauti” (disease) and ”tauditta” (without disease) are derived from the same base word, but have the opposite meanings.

In 1968, Lovins described a development of stemming algorithm, pointing out various pitfalls of stemming in general and discussing different use cases.[26] One of these considerations is whether to use a strong stemming algorithm or a weaker one. A strong algorithm removes more suffixes than a weaker one. This causes more words to fall in the same category (the same stem), but it can be harmful in some cases, such as information extraction.

This is because etymologically related words that have very different meanings, can still share the same stem.[36] Stronger algorithms can still be very useful as search assistant, for example.[26].

2.4.4 Lemmatization

Lemmatization is very closely related to stemming, as both of them aim at reducing the word space that the algorithms need to handle. Lemmatization could be understood as a more sophisticated form of stemming or context-aware form of stemming.[28] It usually produces the base forms of words, and ambiguities such as homographs (see an example in section 2.5) can be resolved, meaning that it is possible to distinct what was meant with a word that can have two different meanings.

Lemmatization can be done with rule-based and machine learning systems. An example of rule-based lemmatization is an RDR-lemmatizer (ripple down rule) by Plisson et al.[40], which is practically a huge if-else-tree for detecting different inflicted forms of Slovenian words. Accuracy of rule-based systems depend on how complete set of rules have been created and also on the language in question. Some languages have irregularities in the inflicted forms, that may cause the rule-based systems to lemmatize words wrong. In Finnish language, there is particularly consonant gradation, which affects the root of the word with different inflections. There is a myriad of different consonant gradation cases, and instead of going through them all, we will only give a few examples below:

1. 'lammas' (sheep) → 'lampaat' (plural of sheep), the root of the word gaining an extra 'p'.

2. 'puku' (suit) → 'puvun' (suit's), 'k' in the word root is transformed into a 'v' in the genitive case

Korenius et al.[19] compared stemming and lemmatization techniques for Finnish language, and concluded that lemmatization performs better than stemming for Finnish language.

2.5 Statistical approach for text mining

Statistical approach for text mining means that numerical analysis of text and it's components is used to extract information from the text. Statistical approach usually needs large data sets and corpora. Simplest versions often reduce a sentence to their mere components, sets of words or their stems, discarding all the information about word order and relations between

words. These kind of algorithms are called 'bag-of-words' methods.[28] Often used statistical methods include word counting, sentence length counting, co-occurrence checking and word distribution analysis.

2.5.1 Bag-of-words

In bag-of-words approach, the context and relations between words are ignored. Instead, the words are considered as they are. This often causes problems, as a large portion of information is lost. For example, one type of problem arises with homographs, words that are typed exactly the same but have a different meaning. Example below.

1. Peter saw a deer in the forest.
2. Julia bought a saw from the hardware store.

Here we have two sentences where a word 'saw' is present. For a human reader, it is obvious that these two instances of 'saw' have nothing to do with each other, as the first one is the past tense of the verb 'see', and the latter refers to a tool. However, a simple statistical algorithm will consider them as the same word, as it will not be able to resolve the homography. By context analysis it would be possible to distinguish these two words.[53]

Another type of problem with this approach is the information loss for not considering the word order. Sentences 'Mary is quicker than John' and 'John is quicker than Mary' seem the same for a bag-of-words algorithm.[28] With more advanced natural language processing algorithms it is possible to determine word dependencies from the word order and part-of-speech information.

It is possible to construct sentences that have ambiguities that can not be resolved. Helsingin Sanomat, Finland's largest news paper, publishes a daily comic strip Fingerpori, in which the ambiguities of Finnish language are often used to make different sorts of puns. Luckily, these sorts of ambiguities do not appear naturally so often, especially in clinical narratives. It is still important to note that ambiguities exist, and that they may have some effect on text mining results.

Bag-of-words approach is well suited for language detection or text classification, and it has been used to detect offensive language from social media, for example.[6] For more accurate information extraction purposes this method alone is not very good, as the relations between words carry a lot of information that is lost when processing texts just as a bag of words.

2.5.2 Co-occurrence

In co-occurrence search, one tries to find two or more concepts or words from a predetermined text block. This text block can be a sentence or a page, for example. This is done to see if these two elements often are present at the same time. If that is the case, we can then generalize that these two terms are linked somehow. For example, Morinaga et al. used co-occurrence search to mine product reputations from the internet with decent results.[33]

2.6 Natural Language Processing

Natural language processing is a variety of techniques that are used to analyze text and/or speech in a way that the relations of words and the correct context are preserved. This is crucial in many text mining tasks, as most languages have many ambiguities like homography, synonymy, uncertain word order, and such.

It is common, that the use of NLP-algorithm needs user input in form of parameter tuning, preprocessing tasks and/or additional pre-created resources.[38]. There are, however, some fully automatic algorithms and methods as well.

2.6.1 Rule-based systems

Rule-based systems utilize knowledge of the structure and vocabulary of the text. Based on this knowledge, a set of rules is created to determine the wanted output. Rule-based systems can give excellent results, but creating a comprehensive set of rules is time consuming work, and needs to be done for each text mining task separately.

There have been, however, some approaches where a small set of rules have been found to provide decent, if not good results. Kilicoglu et al.[18] provided an approach with only 27 grammar rules, derived from syntactic dependency trees, that achieved good precision in event extraction tasks. Their algorithm had F-scores[1] of over 0.4, which can already be used for preliminary filtering of documents, even though recall[2] was not very good.

[1]See the definition from chapter 4.8.1
[2]See the definition from chapter 4.8.1

2.6.2 Word vectorization

The idea of representing words as vectors was presented already in 1971 by Salton.[43] Since then there have been multiple approaches to vectorize words. The basic idea behind it is to depict a word using multidimensional vectors. The goal is that each word has their own unique vector, and semantically similar words have similar vectors, allowing the comparison of the meaning of those words.

One of the latest word vectorization techniques, word2vec was introduced by T. Mikolov et al. in 2013.[31] They demonstrated how a relatively simple back propagating neural network can learn state-of-the-art level vectorization for words if the learning data set is large enough, hundreds of millions words or more. They demonstrated that the vectors created have interesting arithmetic properties, so that, for example for the vectors following is true: 'King' - 'Man' + 'Woman' = 'Queen'. The algorithm was able to learn such properties from the large corpora, without specifically providing these relations that seem intuitive for a human reader.

2.6.3 N-gram

In computational linguistics n-gram means the contiguous segment of n items from a given sequence of items. These items can be letters, words, syllables or any meaningful component of the text. For example, in a sentence 'John ate a pizza.', all the word 1-grams are 'John', 'ate', 'a', 'pizza', and all the 2-grams are 'John ate', 'ate a', 'a pizza'. Similarily, first 5 character 1-grams would be 'John ', 'ohn a', 'hn at', 'n ate' and ' ate '. In natural language processing the items are most commonly words, and the item sequences can be sentences or entire documents.

N-grams can be used to derive relations between different words, that can be further used for text analysis purposes. They can also be used as features for many machine learning algorithms. N-grams also approximately follow Zipf's law that we discussed earlier.[3]

Considering character n-grams, it has been suggested that the mean length of words in a language is a good indicator of morphological complexity.[30] This also means that morphologically more complex languages, such as Finnish, would require using longer character n-grams to extract information.

2.6.4 Kernel methods

There are some kernel methods for text mining purposes as well. The main idea in kernel methods is to create a kernel that can map a low-dimendional feature into a higher-dimensional feature space with low computational cost. This new feature space can give a better separation between features, especially with linear classifiers. Lodhi et al.[24] proposed a novel method for combining N-grams with kernel method to be used with support vector machines for text classification. The developed method made it possible to handle much larger data sets by increasing the computational efficiency through approximations. However, the string-kernel developed did not achieve very good performance when compared to word and n-gram level kernels.

2.7 Keyword extraction

Keywords are words that try to describe a document or a set of documents briefly. For example, many academic journals ask the writers to provide a list of keywords for their article.[51] Keyword extraction can be seen as a type of text summarization, a task that has been computerized as early as 1958 by Luhn[27].

There are various techniques for keyword extraction. It has been done using word co-occurrences[29], with linguistic features like POS-tags[15], by comparing term frequencies to a larger corpus[23], with rule-based genetic algorithm[51] and many other methods. Frank et al.[10] tested multiple different metrics with their Naive Bayes classifier that tried to classify words into keywords and non-keywords. It turned out that the term frequency based attributes and the position within the document were the only relevant attributes for the task in their domain.

2.8 Classification

Classification is a task that aims to assign a label (also known as categories) to some instance for which we do not already know the correct label. This is achieved by using a training set of instances, for which the correct labels are known. Based on the features (also known as observations, variables or predictors) of the training data, a classifier can learn to classify new instances with correct labels. Many text mining tasks can be reduced to binary or multinomial classification. In this section we will briefly introduce relevant classification techniques for our work.

2.8.1 Decision trees

Decision trees are classifiers that predict the label of an instance based on multiple features. They can also be used to predict ordered variables (numerical variables, for example).[32] Decision trees work by splitting the given dataset in multiple parts by simple rules. Each location where such split is made is called a node and the resulting groups are called leaves. For classification and regression tasks, the rules for each split need to be learned in a way that minimizes the final classification or regression error.

There are various different algorithms with many different metrics for creating decision trees, and going through them all is not feasible. For those interested, Wei-Yin Loh has written a good article on the history of regression and classification trees.[25]

2.8.2 Ensemble learning

Ensemble learning methods rely in combining multiple learning methods to provide better predictions than any of it constituents. If each individual classifier performs better than a random guess, it follows that these classifiers can be combined to work as a single stronger classifier. This was proven by Robert Schapire in 1990.[45]

2.8.3 Random forests

Random forests are classifiers first proposed by Ho[14] and further developed by Breiman[2] and Cutler. Random forest is an ensemble learning method where the constituent learning methods are all randomly generated decision trees. These trees branch based on the features that were selected during the training of the random forest model. Each decision tree gives their own prediction based on it's own branching rules. These trees then vote for the final result.

Random forests have been proven to be successful in many different tasks, making them a good general purpose machine learning algorithm. Random forests are also well parallelizable in both training and classification, and classification is very quick after the model has been trained. For these reasons, and for the ease of "out-of-box"-implementation we selected random forests as our machine learning algorithm.

A single decision tree is trained by taking a sample from all the training data, then taking a sample from all training features and training the tree minimizing the classification (or regression) error. Sampling of training data is done to create differences in trees, to avoid local minima caused by always

selecting the same strong features. Sampling of training features is done to prevent strong features being used in all the trees, because it would make these trees correlated, reducing the accuracy of the ensemble. [2]

2.9 Text mining of Finnish texts

Text mining from Finnish texts has been quite limited. Many natural language processing tools and text mining resources that have been available for English do not exist for Finnish. The same is true for many larger languages, such as German [48]. Basic approaches like n-grams have been studied to some extent with other languages than English, but at the time of writing, English text mining research is well ahead of other languages.

The situation regarding Finnish natural language processing tools was improved greatly in 2014 as Katri Haverinen created useful resources and tools for Finnish text mining as a part of her PhD-thesis. [13] These tools include syntactically annotated corpora, dependency annotated corpora and two Finnish treebanks, one for general Finnish and another for clinical Finnish. Also an online dependency parser was made available. These tools can be used to relate the context, grammar and relation of words to each other, allowing more advanced natural language processing methods.

2.10 Text mining of medical and clinical texts

Clinical, medical and biomedical text mining have been around for some time. Even though these fields are closely related, there are major differences between them when regarding text mining. The texts in medical and biomedical journals differ greatly from those that are found in clinical records. [44] Major differences are found in length, vocabulary, usage of acronyms and amount of spelling mistakes. Furthermore, journal entries usually have more structured format with subtitles, pictures, diagrams and tables.

Information extraction from medical reports is a challenging task. A good algorithm should be able to recognize prespecified conditions within the text. These conditions may be family sicknesses, smoking status, previous diseases, or any other thing that affects the patient's health in some way.

There are some widely used algorithms that have showed good results in extracting information from English clinical texts.

Laippala et al. [20] have tested how the training data set for clinical Finnish text parsers affect the parsing outcome. They found that a large training set of general Finnish performs worse than a smaller training set of clinical texts.

This was true even when the clinical texts were not from the exactly same medical domain.

Uzuner et al.[52] have held a competition to identify the smoking status of patients from free texts. Many of the submitted algorithms scored a microaveraged F-score above 0.8, when classifying the patients' smoking status into five categories: past-smoker, current-smoker, smoker, non-smoker and unknown, whereas the human annotators agreed on over 80% of the time.

Cogley et al.[7] tested several machine learning approaches for extracting if and when a patient experienced a medical condition. This was done by first extracting 16 binary variables, using trigger words, contextual features and linguistic features, and then using Naive Bayes, kNN and Random Forest algorithms. The latter two outperformed the ConText algorithm discussed in the next section. They also found random forests to perform slightly better than kNN.

2.10.1 Negation Extraction

Information extraction from clinical texts is one thing. In addition for that, it needs to be known if the condition is negated, because clinical texts have a very large number of negated conditions. In their research data Chapman et al.[4] report that most frequent clinical observations were negated more than half of the time. For example, "patient smokes" and "patient does not smoke" should be distinguished from each other.

In addition to direct negations, it is also good to know if the condition appeared in the past or if it was someone else who was reported to have that condition instead of the patient themselves. These all can mean that the patient is not currently (in the time of writing the medical report) experiencing the given condition.

NexEx-algorithm was published by Chapman et al.[5] in 2001. NexEx is a relatively simple algorithm that tries to match a list of negation phrases that co-occur with known ICD codes and their names. NegEx has also been ported for Swedish[47], French[9] and German languages, showing good results.

Key figures of NegEx for English, Swedish, French and German can be found from table 2.10.1. The figures reported are for detecting the negation correctly from the sentences. German version of NegEx is slightly more complex, as the negation phrases were allowed to be in two parts.

	en (%)	sv (%)	fr (%)	de (%)
Recall	82.4	81.9	84.6	97
Specificity	82.5	74.7	NaN	95
Precision	84.5	75.2	88.8	95
Negative predictive value	80.2	81.4	NaN	NaN
Negative predictive value (no positives)	97.0	96.5	NaN	NaN

NegEx-algorithm has been developed further to extract more information than simple negations. Harkema et al.[12] developed Context-algorithm which is based on NegEx, extracting not only the negation but temporal and experiencer information as well.

2.11 Resources and tools for Finnish clinical text mining

We have listed some Finnish language and text mining resources in this section and some resources that may be used on a variety of languages. Most of the resources are available for free use under open source licenses, but some are restricted. We have also listed the licences at the time of the writing.

FinMeSH

Medical Subject Headings (MeSH) is a structured thesaurus maintained by the US National Library of Medicine. A translated version of MeSH, FinMeSH has been made available[3] and maintained by Finnish Medical Society Duodecim.

Turku Dependency Treebank

Turku Dependency Treebank (TDT) was created by Turku BioNLP Group. It was first released as a stand-alone version, but it has been integrated into Universal Dependencies[4] project in 2015. Turku Dependency Treebank is shared with CC BY-SA 4.0 license.

[3] `https://finto.fi/mesh/fi/`
[4] `http://universaldependencies.org/`

Turku Clinical TreeBank and PropBank

Turku Clinical TreeBank and PropBank[5] are text corpuses parsed and annotated by Turku BioNLP group. They are freely available for download, and have been released with Creative Commons Attribution Share-alike licence. The underlying text corpus consists of manually anonymized nursing narratives of 8 patients, total of about 2800 sentences.

FinnPos

FinnPos[6] is an open-source morphological tagging and lemmatization toolkit for Finnish. Silferberg et al.[46] evaluated FinnPos against other available morphological taggers and lemmatizers, showing that FinnPos performance is at the state-of-the-art level in tagging and lemmatization accuracy, while still maintaining a very good speed. FinnPos is released under Apache Software License v2.0.

Word list of modern Finnish

The institute for the languages of Finland, Kotus, has created a word list of modern Finnish words in 2007. The list[7] contains 94110 Finnish words in their basic forms along with the inflection types encoded in UTF-8 and stored in XML-form. The list is released under GNU LGPL (Lesser General Public License), EUPL v.1.1 and CC BY 3.0 ND. Kotus has also published a wide variety of other Finnish resources[8], that may be helpful when doing text mining in other research areas.

Finnish stopwords

There are several collections of Finnish stopwords available on the internet. One extensive list is provided by University of Neuchâtel on their website[9]. Stopword lists are usually used for removing regularly used words. This makes searching for keywords computationally less expensive, as words with less importance are omitted. It is questionable if stopwords should be removed in text mining, as those frequent words can have crucial information about negations, time or doers. The algorithm developed during this thesis work does not remove stopwords.

[5] `http://bionlp.utu.fi/clinicalcorpus.html`

[6] `https://github.com/mpsilfve/FinnPos`

[7] `http://kaino.kotus.fi/sanat/nykysuomi/`

[8] `http://www.kotus.fi/aineistot/tietoa_aineistoista/sahkoiset_aineistot_kootusti` (In Finnish only)

[9] `http://members.unine.ch/jacques.savoy/clef/`

FinTWOL

FinTWOL is a morphological parser for Finnish. It is a product developed by Lingsoft[10]. They provide a limited use FinTWOL online demo on their web-page.

OMorFi

OMorFi is an open-source morphological analyzer for Finnish language. It is freely available for download[11] and is released under GNU GPL v3 licence.

FinnWordNet

FinnWordNet[12] is a lexical database for Finnish. It is a semantic network, linked through relations such as synonymy and antonymy. The first version was a direct translation of Princeton's English WordNet. Current FinnWordNet version 2.0 was released in 2012 under CC-BY 3.0 licence and is also a subject to original Princeton WordNet licence.

OpenNLP

OpenNLP[13] is an open-source toolkit for natural language processing. It provides tools for many generally used natural language processing tasks, such as tokenization, lemmatization and part-of-speech tagging. OpenNLP is a Java library, and it can be run on Java Virtual Machine. At the time of writing, the latest OpenNLP version is 1.7.2 (released on 4th of February, 2017), and is licensed under Apache Licence 2.0.

[10] `www.lingsoft.fi`

[11] `https://github.com/flammie/omorfi`

[12] `http://www.ling.helsinki.fi/en/lt/research/finnwordnet/index.shtml`

[13] `http://opennlp.apache.org/`

Chapter 3

Environment

Hospital District of Helsinki and Uusimaa

Hospital District of Helsinki and Uusimaa is the largest hospital district in Finland, providing specialized medical care services for over 1.6 million residents in 24 municipalities. HUS treats around 500 000 patients each year, creating vast quantities of specialized health care data. It should be emphasized that all this data is specialized health care data, and does not contain any primary health care data. Because of this, one should not try draw any conclusions of general population based on numbers presented in this thesis.

Data lake

The electronic health records of the Hospital District of Helsinki and Uusimaa (HUS) are located in multiple different databases. To provide researchers an access to some of these records, HUS has created a data lake ('tietoallas' in Finnish), the pre-production version of which was first opened in the first quarter of 2017 after development period and data migration from source databases.

The data lake is built in HDInsight. It runs many services, but in the scope of this work we mainly use its Spark cluster with Hive integration. Our main method for running Spark jobs on the cluster is through Livy, which provides a REST API for running batch jobs on Spark.

At the time of writing, HUS data lake Spark cluster consists of 2 host nodes and 3 worker nodes. These worker nodes have 4 cores and 13.69 GB RAM each. At this early stage the cluster is not very large, but can relatively easily be scaled up when needed.

Apache Spark

During the development of our algorithm, we used Apache Spark 2.0.2, which was later updated to version 2.1. Apache Spark is a general-purpose cluster computing platform. It provides application programming interfaces for Scala, Java, Python and R. These APIs provide a wide range of functionality from simple statistical tools to machine learning algorithms.

3.1 Programming languages

Scala

To develop text mining algorithms we used Scala 2.11.11. We selected Scala, because Apache Spark is Scala native, and this quite possibly eliminates multiple bottlenecks and other compatibility issues that might be present when using Spark through some other language APIs.

Scala is also a relatively efficient language in terms of computational time, and it runs on Java Virtual Machine (JVM) so that it can easily be run on multiple platforms.

Java

Apache OpenNLP library has been written using Java, and all it's documentation was also in Java only. To use this library for lemmatization, we used mixed Java/Scala code. This was possible, as Java libraries can easily be imported in Scala, and because both languages can be run on Java Virtual Machine.

Python

At first we did some development on Python, but we quickly realized that Spark's Scala API would be a better choice for our work. It would have been possible to do all the work with Python, but Spark's Scala API provided better usability and integrating Apache OpenNLP was easier with Scala.

Other programming languages

We tested some tools created by third parties, that have been written in other languages than described above, namely C and C++. These tools could be useful, but making them work together with our Spark cluster would have been too much work, especially when similar tools were available in JVM supporting languages.

3.2 Data sets

We are using two datasets to develop and test our algorithm. The first one is a dataset consisting of all HUS's patients who have a heart attack diagnosis in their records (ICD10 codes I21 and I22) and a similar number of random patients who do not have this diagnosis. In this data set, there are 2678349 patient record texts from 36571 patients, of which 18654 have been diagnosed with myocardial infarction.

For keyword extraction purposes, we require a baseline for word frequencies in patient record texts. For this purpose, we used a random sample from all HUS's patient record texts. This random sample consisted of 1048576 patient record texts.

To evaluate the generalizability of our pipeline, we additionally tested it on a customer feedback dataset. This dataset consisted of 1165 feedback texts, labelled with positive/neutral/negative sentiment tags.

Chapter 4

Methods and Implementation

4.1 Text mining pipeline

In this chapter, we will go through the methods and implementation of our text mining pipeline. We will also explain the reasoning behind our design choices, where applicable. Parts of the process are explained in the same order that the data goes through in our algorithm, so that each step should be easy to follow.

When we began building our text mining pipeline, we had a few requirement for the final system. The requirements were:

- built on HUS's new Spark cluster
- general purpose
- modular
- interpretable results (no black box models)
- fast (in our case equals well parallelizable)

A text mining process naturally consists of multiple smaller subroutines. It was quite easy to link these tools together while maintaining modularity of the system. These processes and subroutines make up the text mining pipeline. The final pipeline used in this work is shown in figure 4.1.

4.2 XML Parser

Most patient records in HUS are stored in XML format. However, there are also patient records that are in plain text. Our first part of the XML-parser

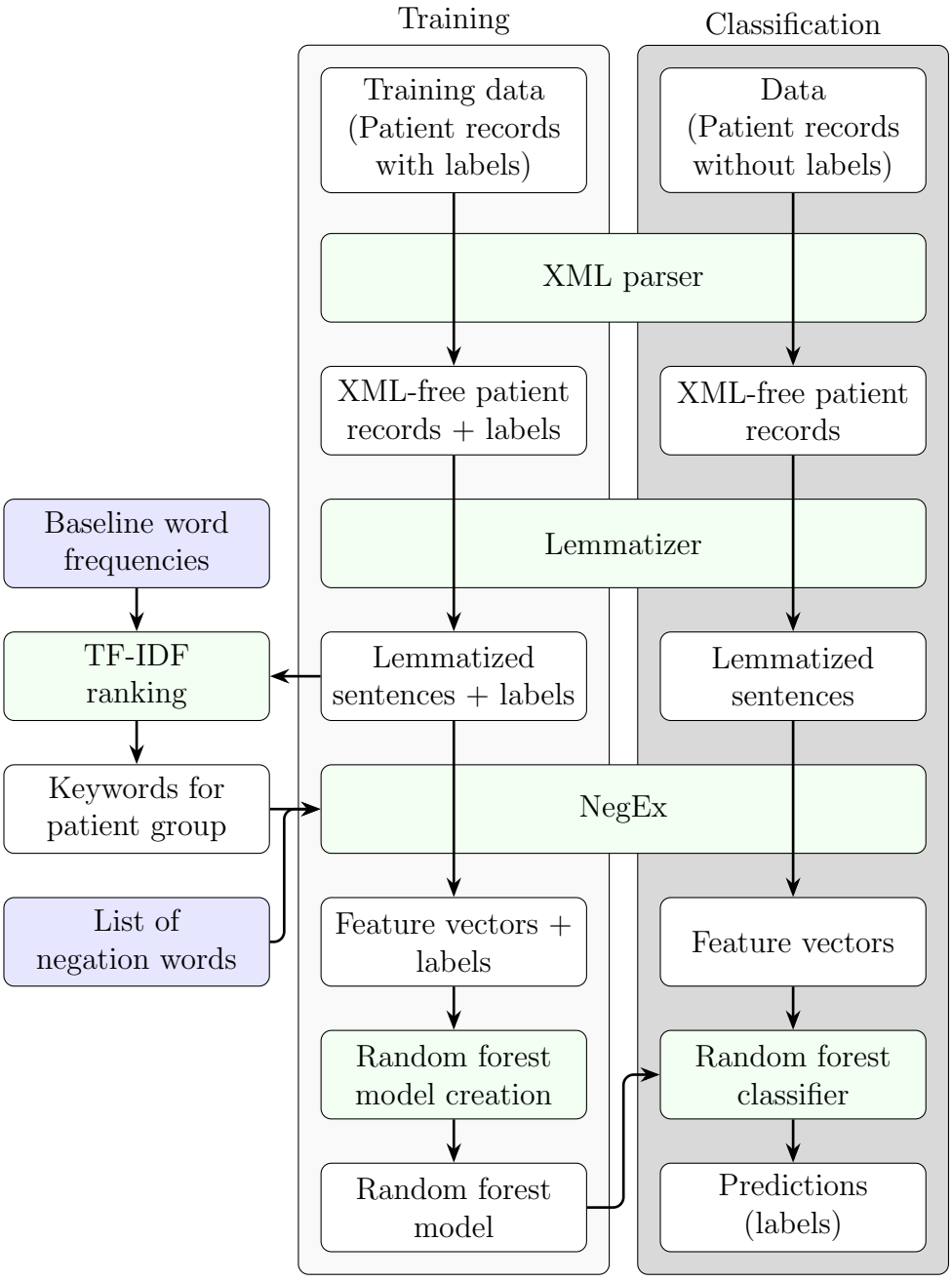

Figure 4.1: Text mining pipeline created during the thesis work. Each green box represents an individual component that can be replaced or developed further. White boxes represent the state of our data between each processing component. Blue boxes represent resources that have been created prior to running this algorithm.

is a short regex-code that looks for valid XML-tags, and feeds the input to the actual XML-parser when those tags are found.

The XML structure of patient records has many different predefined sections and subtitles that are automatically or semi-automatically created when a doctor writes the patient record. All these sections and chapters were individually stored as single XML-nodes, that were nested within each other. The count of these nodes varied, and it was not possible to know or predict how many there will be in a single document.

We used the standard Scala XML Library to parse patient record text from the XML. We used recursive parsing of the XML, because of the nested structure and unknown number of nodes. Each node's text was appended into a single string, collecting the patient record text in it's entirety.

4.3 Spell-check

Medical records are usually not spell-checked, in order to prevent the computer from guessing a wrong word to replace a misspelled one. For text mining algorithms, alternating spellings of the same word can be a problem, as they would not be considered the same word. As the texts are usually very quickly written, it is a good idea to have some form of spelling check or typo handling in place.

In this thesis work we decided not to implement a separate spell checker. This task was left for the lemmatizer, which will be explained in the next section. Implementing a good spell checker could improve results, but we expect this effect to be very small.

4.4 Lemmatization

Lemmatization is a relatively difficult task in it's own and creating a lemmatizer from a scratch was clearly out of scope of this thesis work. Therefore, we used open-source tools for lemmatization.

We used Apache OpenNLP 1.7.2[1] for lemmatization. The OpenNLP lemmatizer requires the input sentences to be tokenized and part-of-speech-tagged. Therefore, we used OpenNLP tools for sentence splitting, tokenization and part-of-speech-tagging as well. These four parts make up our lemmatizer block.

[1] `http://opennlp.apache.org/download.html`

To train the OpenNLP minimum entropy models, we used Universal Dependencies 1.4 Finnish treebank[2].

OpenNLP was updated to version 1.8.0 on 18th of May 2017. Due to time limitations we did not update this version to our algorithm. Making the required changes would make our lemmatizer code a little more straightforward and add a direct CONLL-U support, which is needed in training the lemmatizer.

Lemmatizing is fully parallelizable, but it is still very slow. It takes roughly 16 hours to lemmatize one million patient record texts on our 12-core cluster. This is roughly 1000 patient records per minute. Fortunately, lemmatization only needs to be done once for each patient record.

4.5 Keyword extraction

NegEx algorithm needs keywords that are searched from the text. Manually creating a complete list of all relevant keywords is nearly impossible, and requires a lot of time and expertise. Creating these keyword lists must also be done for each text mining task separately, which means that doing it manually would require a lot of time from doctors. This wasn't a good option for us, so we decided to implement an automatic keyword extraction method for our algorithm.

Our keyword extractor calculates word frequencies for every word that occurs in HUS's medical reports. It also calculates the fraction of all the medical records that contain that same word. By comparing the frequencies from the entire medical report database with those of our special interest (specific diagnose, or any known condition), we hope to be able to recognize the relevant keywords that are linked to a specific condition.

The method described above is known as TF-IDF. It stands for term frequency - inverse document frequency. Variants of it are widely used in the field of text mining, even though the information theoretical justification for it's use is unclear to some extent.[42] TF-IDF-ranking does, however seem like an intuitive way to rank words as keywords for a specific document set. The rationale behind TF-IDF is that infrequent words in the entire corpora that are frequently present in a subset of the corpora, are very likely to be good keywords for that subset. Thus, searching with those keywords would return documents belonging to that specific subset, and not so much other documents.

TF-IDF approach has been previously used with multiple parameters and

[2] `https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-1827`

tweaks with good results.[23] Usually it is used to extract single keywords, but a word bigram expansion has also been tested by Liu et al. achieving improvement over the traditional version.[22]

There are many ways to scale and normalize TF-IDF measures. Wikipedia has a great listing of different scalings: `https://en.wikipedia.org/wiki/Tf-idf`

Term frequency is calculated for each lemma in the patient records of our group of interest (e.g. patients that have had a heart attack). It is commonly assumed that term relevance does not linearly follow term frequency, and therefore a damping should be used for TF. Most common damping method is to use log(TF) measure instead.[37] We selected this same method with a small constant correction to avoid negative TFs. Our TF function is presented in equation 4.1.

In this work we used TF-IDF with logarithmic normalizations. TF normalization was calculated using

$$\mathrm{tf}(t, d) = 1 + \log(f_{t,d}), \tag{4.1}$$

where $f_{t,d}$ is the frequency of a term t in the subset d of patient records.

Inverse document frequency is calculated on a document level. This means that we calculate the frequency of documents that contain the lemma. This frequency is the inverted and again logarithmically scaled. Inverse document frequency is defined in equation 4.2. For IDF we used

$$\mathrm{idf}(t, D) = \log\left(1 + \frac{N}{n_t}\right), \tag{4.2}$$

where N is the number of all patient records D and n_t is the number in which the term t is present.

Finally, term frequency and inverse document frequencies for each lemma are multiplied with each other to achieve TF-IDF ranking. Words with the highest rank should be better keywords for our patient group than those with lower rank. We can then use any number of highest ranking words as NegEx keywords. Final TF-IDF ranking was then calculated with

$$\text{TF-IDF}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D) \tag{4.3}$$

4.6 NegEx

We created a simple implementation of NegEx-algorithm. This algorithm searches for given keywords, and any negation words within a specified distance range from keywords. For our work, we used following negation words:

'ei' (='no'), 'eikä' (='and not') and 'ilman' (='without'). Our NegEx-algorithm outputs feature vectors that can be directly used in Spark MLlib.

Feature vectors are created by taking a vector of zeros with a length of the given keyword list. When a keyword is found, the corresponding element in the feature vector is updated. If there is a negation word present within the scope, we deduct 1 from the element, otherwise we add 1. Feature vector is calculated for each medical report separately, and they are summed element-wise for each patient when patient level features are needed.

4.7 Random forest classifier

Once we have a feature vector for each patient record (or patient by summing the individual feature vectors), we can train a random forest model. For this we use Spark MLlib random forests. We left 30 % of the training data out for validation purposes and used 70 % for training. This training and validation dataset was entirely separate from our final validation dataset.

Instead of random forests, we could use some other machine learning methods or classifiers. Comparison of different machine learning methods was left out of scope of this work, but it is definitely something that should be done in the future.

When we have patient records without labels and a pre-trained random forest classifier model, we can classify these unknown patient records with our random forest model.

4.8 Evaluation of the algorithm

In this section, we will go through classification metrics that are commonly used.

4.8.1 Precision, recall and F-score

For binary classification, data can belong to four categories. These categories are true positive (TP), false positive (FP), true negative (TN) and false negative (FN). The category depends on the actual and predicted label of the item, as shown in figure 4.2.

Standard way of evaluating clustering, named entity recognition and other identification tasks is to use precision, recall and F-measure. Perfect algorithm returns all the interesting documents and there are no uninteresting

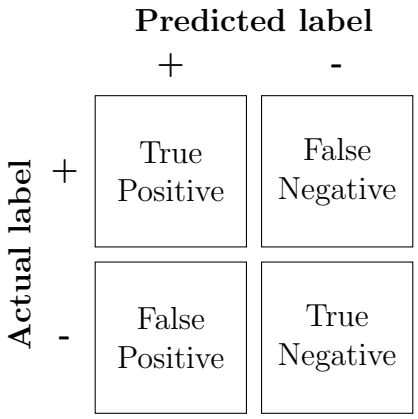

Figure 4.2: Confusion matrix of binary classification.

documents in the search results. Precision and recall are derived from true positives, true negatives, false positives and false negatives as follows.

Let us assume a task of identifying interesting documents out of a large set of documents. Precision of the algorithm tells us what proportion of those documents that were retrieved, were actually interesting. Precision P can be calculated as

$$P = \frac{N_{TP}}{N_{TP} + N_{FP}}, \tag{4.4}$$

where N_{TP} is the number of true positives and N_{FP} is the number of false positives.

Recall (also known as sensitivity) gives us the proportion of the interesting documents that were retrieved from all the interesting documents in that set. Recall R is given by

$$R = \frac{N_{TP}}{N_{TP} + N_{FN}}, \tag{4.5}$$

where N_{FN} is the number of false negatives.

It should be noted, that it is trivial to make an algorithm that has the recall of 1, simply by returning every document as it guarantees that all the desired documents will be returned. Precision can be 'hacked' in the similar manner, simply returning a single document that has the desired feature and nothing else. Considering this, it becomes clear that one can trade precision for recall and vice versa. Therefore, both of these measures need to be taken into account when determining the performance of an algorithm. For this purpose, F-measure can be used. There are both weighted and unweighted F-measures. Standard, unweighted F-measure F is the harmonic mean of

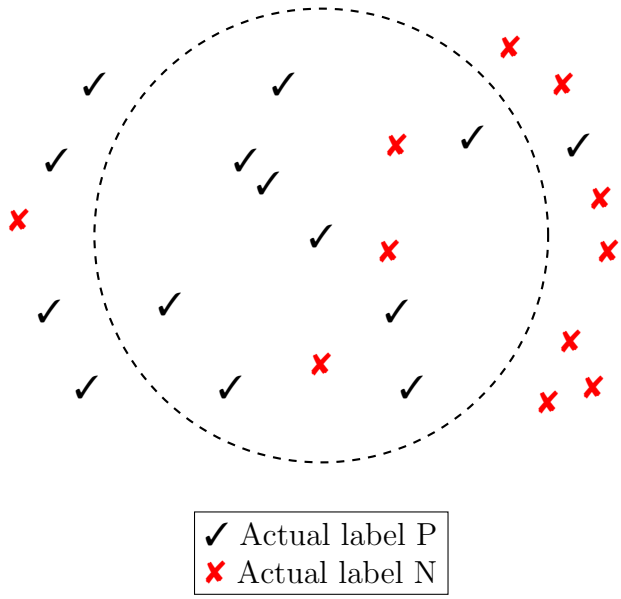

Figure 4.3: A diagram to illustrate precision and recall. An algorithm (circle) returns total of 12 documents. 9 of the returned documents have the desired feature (label P, (✓) and 3 do not (label N, ✘). Precision is therefore 9/12. There were 14 documents with the desired feature, but only 9 of them were returned, so recall is 9/14.

precision and recall,

$$F = 2 \cdot \frac{P \cdot R}{P + R}. \tag{4.6}$$

4.8.2 Receiver operating characteristics

Another common way to evaluate binary classifiers is receiver operating characteristics (ROC) curve. Binary classifiers usually produce a continuous variable that is used to determine the class together with a threshold. ROC curve is true positive rate plotted against false positive rate at any given threshold. This allows the user to tune the threshold to take different kinds of trade-offs into account. For example, when selecting patients for screening, it is likely that false positives should not be penalized very strongly, as screening is not likely to have adverse effects. Situation would be completely opposite when deciding whether a patient should be given some medicine with known side-effects.

Receiver operating characteristics space has a range [0,1] on both x-axis (false positive rate, FPR) and y-axis (true positive rate, TPR). False positive

rate and true positive rate are defined in equations 4.7 and 4.8. Next we will go through some points on that space. Point (0,0) represents a classifier which always predicts negative class and (1,1) always predicts positive class. A straight line from the origo to (1,1) is the so-called 'random-guess line', which depicts a classifier that randomly guesses the class. A perfect classifier would have a ROC curve that goes through (0,1), meaning 100% true positive rate and 0% false positive rate.

Fall-out or false positive rate (FPR) is given by

$$\text{FPR} = \frac{N_{FP}}{N}, \tag{4.7}$$

where N is the number of all classified items. Sensitivity or true positive rate (TPR) is given by

$$\text{TPR} = \frac{N_{TP}}{N_P} = \frac{N_{TP}}{N_{TP} + N_{FN}}, \tag{4.8}$$

where N_P is the number of all positive cases in the classification data.

Any reasonable classifier has a ROC curve that is clearly over the random-guess line. Being well under it is also acceptable, as it simply means that the classifier's results should be re-mapped so that all positives become negatives and vice versa. A classifier is the better the higher area it covers. Therefore, it is meaningful to use the area under the curve (AUC) as a summarizing evaluation metric for binary classifiers. This is simply the integral over the ROC-range. AUC is also known as c-statistic.

4.8.3 K-fold cross-validation

K-fold cross-validation is a technique, where the training set for a classifier is divided into K parts. Then a single part out of these K parts is left out from training, and used as validation dataset. This process is run separately for each K folds, and the chosen evaluation metrics are averaged over all of them. In this thesis work, we use K-fold cross validation (with K=5) in random forest parameter tuning.

Chapter 5

Evaluation

Myocardial infarction dataset

We used myocardial infarction data for preliminary parameter tuning. This dataset consists of 2678349 patient record texts from 36571 patients, of which 18654 who have been diagnosed with myocardial infarction (ICD10 codes I21 and I22). The task was to classify patients into two groups, patients that have not had a heart attack, and patients that have had a heart attack. We divided the dataset into two parts, 70% into parameter tuning and training and 30% into testing. The 70% training data was further divided into 5 approximately same size subsets for k-fold cross-validation. These 5 folds were kept constant for the entire validation, as calculating the word frequencies etc. again for each randomization would have slowed down the evaluation process significantly.

For the starting point of our parameter tuning, we selected following parameters:

- $N_{keywords} = 1000$
- $depth_{max} = 30$
- $N_{trees} = 250$

For the simplicity, we used F-score to evaluate the goodness of the parameters. One-by-one, we tuned different parameters, starting from the maximum depth of trees. Results for the tree depth tuning can be seen in figure 5.1. F-score stays practically constant regardless of the maximum tree depth. There is some strange behaviour with tree depths of 13 and 14, that we can not explain. Training deeper trees takes only a little more time than training shallower trees, and there was no sign of serious overfitting, so we decided to use 30 as our maximum depth of trees, which is also the maximum supported depth in Spark's ML library.

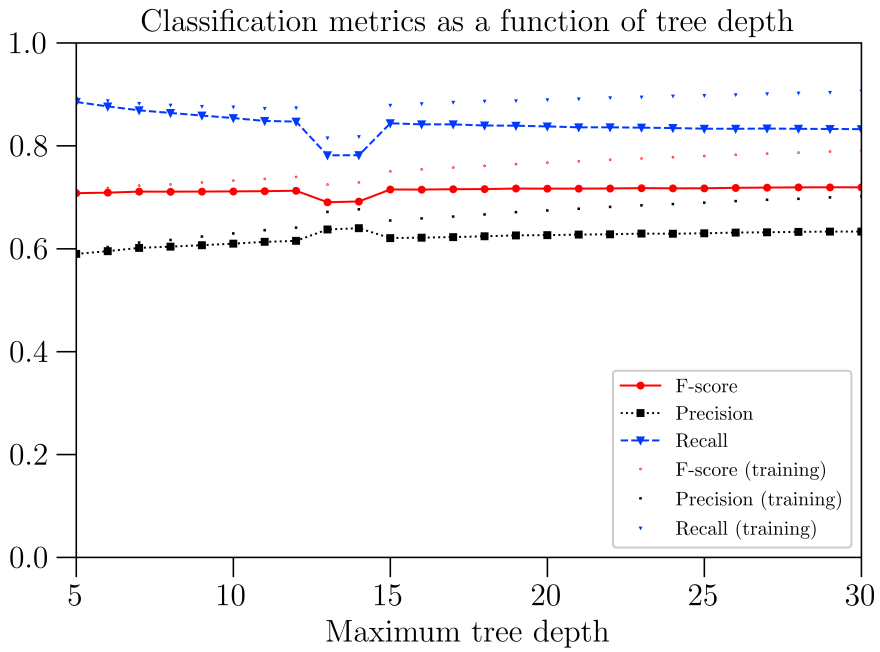

Figure 5.1: Classification metrics for parameter tuning, when tree count and number of keywords were kept constant and number of keywords was varied. The results are an average of 5-fold cross-validation. Lines were validated on the partition of data that was left out of training, separate ticks were validated on the training data consisting of 4 partitions.

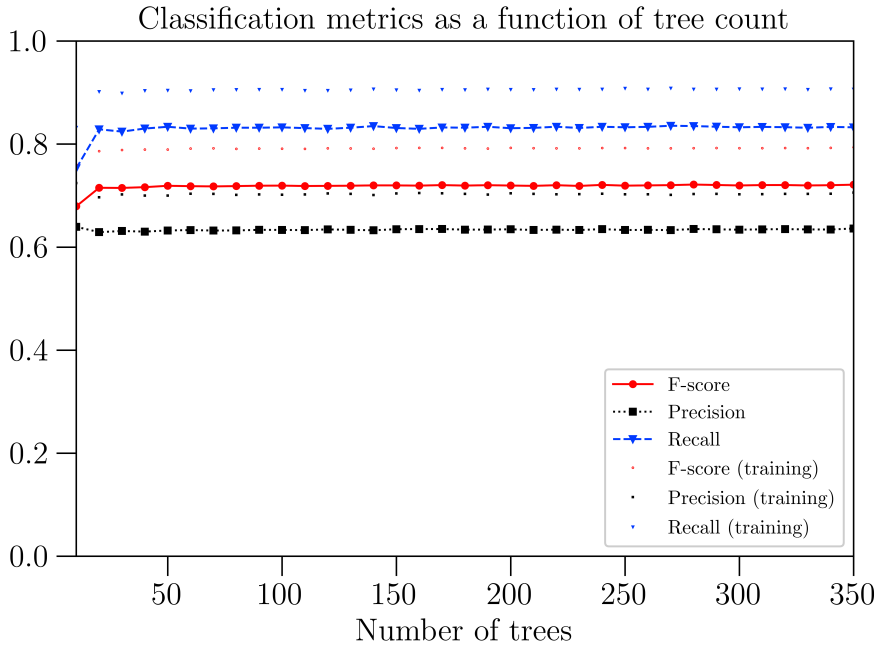

Figure 5.2: Classification metrics for parameter tuning, when tree count and number of keywords were kept constant and number of keywords was varied. The results are an average of 5-fold cross-validation. Lines were validated on the partition of data that was left out of training, separate ticks were validated on the training data consisting of 4 partitions.

After determining optimal maximum tree depth, we tuned the number of trees in our random forest. In figure 5.2 we can see that the classification metrics stay almost constant regardless of the tree count. There is, however, a slight trend upwards with respect to the tree count. In theory, there should be no harm in having higher tree count, other than longer training time, so we conservatively selected tree count of 250 to be used later.

First, we ran 5-fold cross-validation for the number of keywords ranging from 100 to 2000. We soon noticed that 2000 words was too low upper limit. It was not possible to say whether F-score would keep rising above 2000 words, so we ran another parameter validation run in the range from 2000 to 14000 keywords with a stepping of 2000 keywords. Both results can be seen in figure 5.3. From the figure we can see that the classification F-score does not get significantly better after 10000 words. Additionally, with our current cluster it was not possible to use more than 14000 words, due to memory limitations.

For sanity check, we ran evaluation runs again, starting with the first

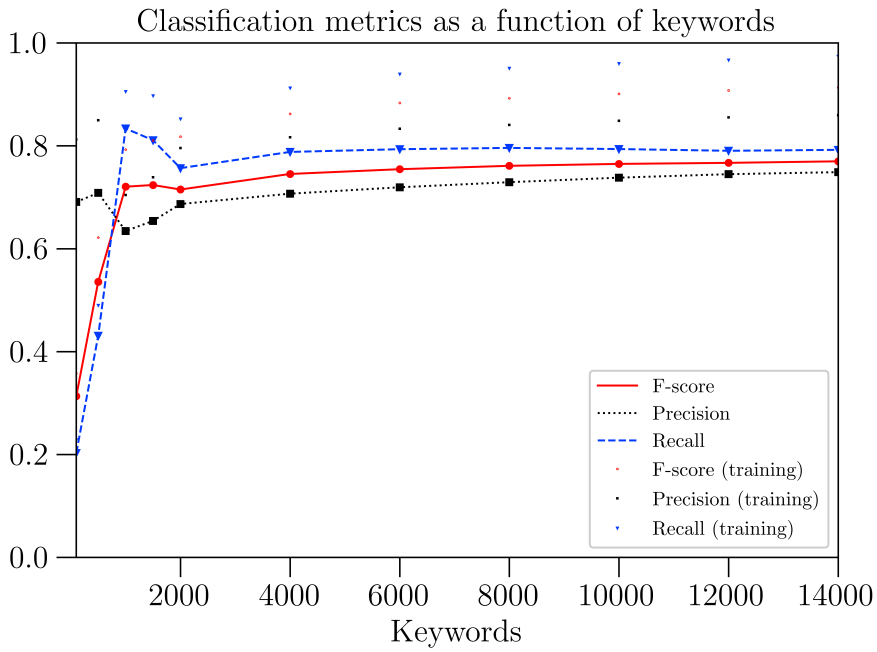

Figure 5.3: Classification metrics for parameter tuning, when tree count and depth were kept constant and number of keywords was varied. The results are an average of 5-fold cross-validation. Lines were validated on the partition of data that was left out of training, separate ticks were validated on the training data consisting of 4 partitions.

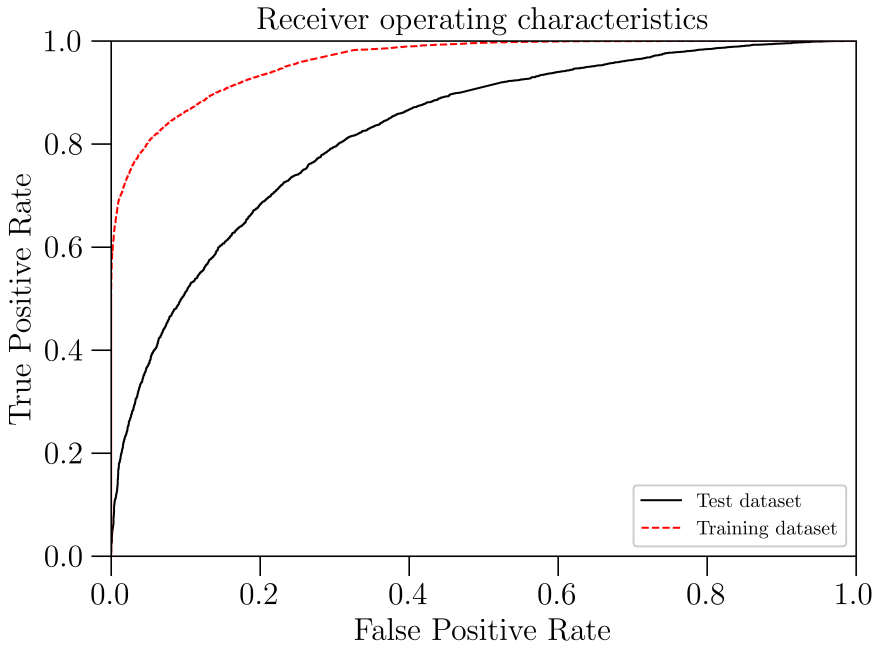

Figure 5.4: Receiver operating characteristics curve. Pure text mining task, all patient record texts were included, both before and after the diagnosis of myocardial infarction. Area under ROC = 0.825.

round's optimal parameters. There were no noticeable changes. We concluded that the following parameters are optimal, or close enough to optimal in our use:

- $N_{keywords} = 10000$
- $depth_{max} = 30$
- $N_{trees} = 250$

We tested the generalization of our algorithm with these parameters with our 30% test data. Area under receiver operating characteristics curve was 0.825 in this task. The ROC curve can be seen in figure 5.4.

After getting encouraging results in pure text mining task, we created another data set from our myocardial infarction data. This new dataset consists of the patient record texts that were recorded 28 days or more before the first myocardial infarction diagnosis. Therefore, these texts should not have any direct signs of a heart attack, but possibly features that could predict one. We found that even with this time filtering the results are good. Receiver operating characteristics curve of this prediction task can be seen in figure 5.5.

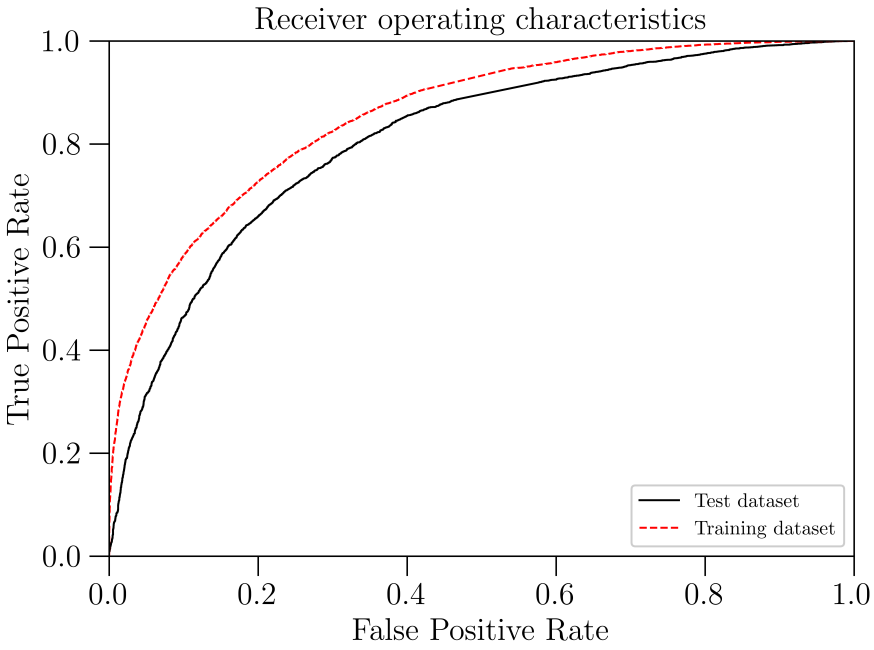

Figure 5.5: Receiver operating characteristics curve for predicting myocardial infarction. Only patient record texts ≥28 days before patients' first myocardial infarction diagnosis were included. Area under ROC = 0.807.

Customer feedback dataset

Our algorithm was additionally tested on customer feedback data. This dataset consisted of 1165 feedback texts, that an expert had classified into three categories: positive, negative and neutral. Because the dataset was small, we did not rank the keyword candidates with TF-IDF, but used all the distinct lemmas that were present in the texts. For this task we did not do separate parameter tuning, but used the parameters that were found to be best with the myocardial infarction dataset.

The dataset was randomly divided into 10 approximately same size subsets. To classify all the feedbacks, we classified each of these subsets separately. Each subset was classified using a model that was trained only with the 9 other data subsets. By classifying texts with this method, we were able to match the expert's classification in 916 feedback texts out of the total 1165. This equals to 78.6% success rate. For comparison, another human classifier agreed on the expert's classification for 87.1% of the time.

Chapter 6

Discussion

In this work we created a multipurpose text mining tool that works for Finnish texts in medical domain. Our targeted user group was HUS's data scientists, who can now use the created tool within HUS data lake environment. We showed that our algorithm works within medical domain, but as there are no domain specific parts, it could also be used outside the medical domain.

Our algorithm can classify patients into groups based on their patient record texts. The groups can be arbitrary, as long as the user provides sufficient training data for each group (or class). It is likely that our algorithm's accuracy depends greatly on the selected target grouping and the amount of training data given. However, from the 3 different use cases tested, we can assume good classification accuracy in very different tasks.

Medical text classification has been done before, but it is not easy to compare different approaches, when the data and the language are different. The closest comparison we found during our literature review was in Suominen's doctoral dissertation. They tested binary topic labeling with machine learning techniques for Finnish nursing notes, achieving classification accuracy between 0.44 to 0.69, depending on the topic in question. [50]

To our knowledge, this type of classification task has not been reported for Finnish medical texts. Therefore, our work could be considered as some type of baseline for text classification in Finnish medical domain.

Further development of our algorithm requires component-wise evaluation and separate human annotated training data for each task. If we allow black box models in our pipeline, a wider range of machine learning methods could be tested in this task. Especially neural networks (deep convolutional or recurrent) could provide interesting results in this task, but require the feature vectors or the entire pipeline to be revamped. For example, there are neural network approaches that learn text features from the character level,

instead of word (or lemma) level, like we did in this thesis work. Future work could also include testing the algorithm's predictive capabilities on primary health care data, when it is available.

6.1 Limitations

The main limitation for the usage of our algorithm is, that any problem must be reduced to a classification or regression task. For example, our tool does not provide means to search for individual laboratory results. Additionally, the algorithm benefits from very large training data, and the classification accuracy may be hindered if given too small training data sets. The availability of sufficiently large training data is often limited due to privacy concerns, but it may be solved with anonymization and data shuffling techniques.

Many of the components used in our text mining pipeline are likely not at the state-of-the-art level, and significant improvements may be available by replacing single components in the pipeline. It would not have been possible to test and evaluate each component separately during this thesis work. Many of the state-of-the-art methods were not readily available for Spark environment and implementing them from a scratch would have taken too much time. For example, instead of using a maximum entropy machine learning lemmatizer provided within OpenNLP, it would be interesting to see how well lemmatization can be done with algorithms that use dependency parsing. Likewise, our random forest classifier might be replaced with some convolutional neural network approach.

When evaluating the results in this thesis, one needs to keep in mind, that Hospital District of Helsinki and Uusimaa only offers special healthcare. We do not have access to primary health care data or social care data, and therefore we can not directly access some very important information about patients' health. In the future this may change, as the electronic health record systems are unified within Helsinki region and new counties begin their work. Our algorithm should be re-evaluated with this expanded data, if this happens.

Chapter 7

Conclusions

Our goal in this work was to create a text mining pipeline, that is modular, works on Apache Spark, is general purpose, does not have black box models and is relatively fast. In this thesis we created a multipurpose text mining tool for Finnish medical domain with the given properties. We found that the textual features do not need to be very complex for an algorithm to perform well.

We also tested the same algorithm on a prediction task. We showed that our algorithm can be used for predicting diagnoses by restricting our algorithms access to only pre-diagnosis texts. From this test it becomes also clear that patient record texts created in special health care have features that can be used for predicting future diagnoses.

Before and during the development of our algorithm we conducted a literature review, that can be used to get acquainted to text mining and tools available for that purpose.

We created a text mining pipeline that works with the tasks we tested it with. The pipeline has no domain specific information, allowing it to be tested for many different text mining tasks and used as is, or only be used as a baseline for other approaches. The created text mining algorithm is available for HUS's data scientists to use within HUS's data lake environment.

Bibliography

[1] Aberdeen, J., Burger, J., Day, D., Lynette, H., David, P., Patricia, R., and Marc, V. Mitre: Description of the alembic system used in met. In *The Proceedings of the Sixth Message Understanding Conference (MUC-6)* (1995).

[2] Breiman, L. Random forests. *Mach. Learn. 45*, 1 (Oct. 2001), 5–32.

[3] Cavnar, W. B., and Trenkle, J. M. N-gram-based text categorization. In *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval* (1994), pp. 161–175.

[4] Chapman, W. W., Bridewell, W., Hanbury, P., Cooper, G. F., and Buchanan, B. G. Evaluation of negation phrases in narrative clinical reports. *Proceedings of the AMIA Symposium* (2001), 105–109.

[5] Chapman, W. W., Bridewell, W., Hanbury, P., Cooper, G. F., and Buchanan, B. G. A simple algorithm for identifying negated findings and diseases in discharge summaries. *Journal of Biomedical Informatics 34*, 5 (2001), 301–310.

[6] Chen, Y., Zhou, Y., Zhu, S., and Xu, H. Detecting offensive language in social media to protect adolescent online safety. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing* (Sept 2012), pp. 71–80.

[7] Cogley, J., Stokes, N., Carthy, J., and Dunnion, J. Analyzing patient records to establish if and when a patient suffered from a medical condition. In *Proceedings of the 2012 Workshop on Biomedical Natural Language Processing* (Stroudsburg, PA, USA, 2012), BioNLP '12, Association for Computational Linguistics, pp. 38–46.

[8] CORDIS - Community Research and Development Information Service. Le-parole: Language engineering - preparatory action

for linquistic resources organization for language engineering. `http://cordis.europa.eu/project/rcn/34076_en.html`. [referenced: 25.7.2017].

[9] Deléger, L., and Grouin, C. Detecting negation of medical problems in french clinical notes. In *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium* (New York, NY, USA, 2012), IHI '12, ACM, pp. 697–702.

[10] Frank, E., Paynter, G. W., Witten, I. H., Gutwin, C., and Nevill-Manning, C. G. Domain-specific keyphrase extraction. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (San Francisco, CA, USA, 1999), IJCAI '99, Morgan Kaufmann Publishers Inc., pp. 668–673.

[11] Gillum, R. F. From papyrus to the electronic tablet: A brief history of the clinical medical record with lessons for the digital age. *The American Journal of Medicine 126*, 10 (2013), 853–857.

[12] Harkema, H., Dowling, J. N., Thornblade, T., and Chapman, W. W. Context: An algorithm for determining negation, experiencer, and temporal status from clinical reports. *J. of Biomedical Informatics 42*, 5 (Oct. 2009), 839–851.

[13] Haverinen, K. *Natural Language Processing Resources for Finnish.* PhD thesis, University of Turku, 8 2014.

[14] Ho, T. K. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1* (Washington, DC, USA, 1995), ICDAR '95, IEEE Computer Society, pp. 278–282.

[15] Hulth, A. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing* (Stroudsburg, PA, USA, 2003), EMNLP '03, Association for Computational Linguistics, pp. 216–223.

[16] Institute for the Languages of Finland. Institute for the languages of finland webpage. `http://www.kotus.fi/kielitieto/kielet/suomi`. [referenced: 13.12.2016].

[17] Kamps, J., Adafre, S. F., and de Rijke, M. *Effective Translation, Tokenization and Combination for Cross-Lingual Retrieval.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 123–134.

[18] Kilicoglu, H., and Bergler, S. Effective bio-event extraction using trigger words and syntactic dependencies. *Computational Intelligence 27*, 4 (2011), 583–609.

[19] Korenius, T., Laurikkala, J., Järvelin, K., and Juhola, M. Stemming and lemmatization in the clustering of finnish text documents. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management* (New York, NY, USA, 2004), CIKM '04, ACM, pp. 625–633.

[20] Laippala, V., Viljanen, T., Airola, A., Kanerva, J., Salanterä, S., Salakoski, T., and Ginter, F. Statistical parsing of varieties of clinical finnish. *Artificial Intelligence in Medicine 61*, 3 (2014), 131–136. Text Mining and Information Analysis of Health Documents.

[21] LePendu, P., Iyer, S. V., Fairon, C., and Shah, N. H. Annotation analysis for testing drug safety signals using unstructured clinical notes. *Journal of Biomedical Semantics 3*, 1 (2012), S5.

[22] Liu, F., Liu, F., and Liu, Y. Automatic keyword extraction for the meeting corpus using supervised approach and bigram expansion. In *2008 IEEE Spoken Language Technology Workshop* (Dec 2008), pp. 181–184.

[23] Liu, F., Pennell, D., Liu, F., and Liu, Y. Unsupervised approaches for automatic keyword extraction using meeting transcripts. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics* (Stroudsburg, PA, USA, 2009), NAACL '09, Association for Computational Linguistics, pp. 620–628.

[24] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. Text classification using string kernels. *J. Mach. Learn. Res. 2* (Mar. 2002), 419–444.

[25] Loh, W.-Y. Fifty years of classification and regression trees. *International Statistical Review 82*, 3 (2014), 329–348.

[26] Lovins, J. B. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics 11*, 1 and 2 (Mar and Jun 1968).

[27] Luhn, H. P. The automatic creation of literature abstracts. *IBM J. Res. Dev. 2*, 2 (Apr. 1958), 159–165.

[28] Manning, C. D., Raghavan, P., and Schütze, H. *Introduction to Information Retrieval*, online ed. Cambridge University Press, 4 2009.

[29] Matsuo, Y., and Ishizuka, M. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools 13*, 01 (2004), 157–169.

[30] McNamee, P., and Mayfield, J. Character n-gram tokenization for european language text retrieval. *Information Retrieval 7*, 1 (2004), 73–97.

[31] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[32] Morgan, J. N., and Sonquist, J. A. Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association 58*, 302 (1963), 415–434.

[33] Morinaga, S., Yamanishi, K., Tateishi, K., and Fukushima, T. Mining product reputations on the web. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2002), KDD '02, ACM, pp. 341–349.

[34] Newman, M. Power laws, pareto distributions and zipf's law. *Contemporary Physics 46*, 5 (2005), 323–351.

[35] O'Neil, J. Doing things with words, part two: Sentence boundary detection. http://archive.is/UIeD, 10 2008.

[36] Paice, C. D. An evaluation method for stemming algorithms. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 1994), SIGIR '94, Springer-Verlag New York, Inc., pp. 42–50.

[37] Paik, J. H. A novel tf-idf weighting scheme for effective ranking. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2013), SIGIR '13, ACM, pp. 343–352.

[38] Paukkeri, M.-S. *Language- and domain independent text mining*. PhD thesis, Aalto University, 11 2012.

[39] Piantadosi, S. T. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic Bulletin & Review 21*, 5 (2014), 1112–1130.

[40] Plisson, J., Lavrac, N., Mladenić, D., et al. A rule based approach to word lemmatization. In *Proceedings of the 7th International Multiconference Information Society, IS-2004* (2004), pp. 83–86.

[41] Riley, M. D. Some applications of tree-based modelling to speech and language. In *Proceedings of the Workshop on Speech and Natural Language* (Stroudsburg, PA, USA, 1989), HLT '89, Association for Computational Linguistics, pp. 339–352.

[42] Robertson, S. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation 60*, 5 (10 2004), 503–520.

[43] Salton, G. *The SMART Retrieval System—Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.

[44] Savova, G. K., Masanz, J. J., Ogren, P. V., Zheng, J., Sohn, S., Kipper-Schuler, K. C., and Chute, C. G. Mayo clinical text analysis and knowledge extraction system (ctakes): architecture, component evaluation and applications. *Journal of the American Medical Informatics Association 17*, 5 (2010), 507–513.

[45] Schapire, R. E. The strength of weak learnability. *Machine Learning 5*, 2 (Jun 1990), 197–227.

[46] Silfverberg, M., Ruokolainen, T., Lindén, K., and Kurimo, M. Finnpos: an open-source morphological tagging and lemmatization toolkit for finnish. *Language Resources and Evaluation 50*, 4 (2016), 863–878.

[47] Skeppstedt, M. Negation detection in swedish clinical text: An adaption of negex to swedish. *Journal of Biomedical Semantics 2*, 3 (2011), S3.

[48] Starlinger, J., Kittner, M., Blankenstein, O., and Leser, U. How to improve information extraction from german medical records. *it - Information Technology* (2016 published online).

[49] Statistics Finland. Official statistics of finland (osf): Population structure [e-publication]. `http://www.stat.fi/til/vaerak/2015/vaerak_2015_2016-04-01_tau_002_en.html`. [referenced: 13.12.2016].

[50] Suominen, H. *Machine Learning and Clinical Text.* PhD thesis, University of Turku, 12 2009.

[51] Turney, P. D. Learning algorithms for keyphrase extraction. *Information Retrieval 2*, 4 (May 2000), 303–336.

[52] Uzuner, Ö., Goldstein, I., Luo, Y., and Kohane, I. Identifying patient smoking status from medical discharge records. *Journal of the American Medical Informatics Association 15*, 1 (2008), 14–24.

[53] Yarowsky, D. *Homograph Disambiguation in Text-to-Speech Synthesis.* Springer New York, New York, NY, 1997, pp. 157–172.

[54] Zipf, G. *Human behavior and the principle of least effort: an introduction to human ecology.* Addison-Wesley Press, 1949.