Applying machine learning to automatic incident detection from software log output

Aapo Rantala

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 22.5.2019

Supervisor

Asst.Prof. Arno Solin

Advisors

D.Sc. (Tech.) Tuomas Viitanen

Lic.Sc. (Tech.) Tapio Pitkäranta

Aalto University
School of Electrical Engineering

Aalto University
School of Electrical Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Aapo Rantala

Title Applying machine learning to automatic incident detection from software log output

Degree programme Automation and electrical engineering

Major Translational engineering **Code of major** ELEC3023

Supervisor Asst.Prof. Arno Solin

Advisors D.Sc. (Tech.) Tuomas Viitanen, Lic.Sc. (Tech.) Tapio Pitkäranta

Date 22.5.2019 **Number of pages** 54+4 **Language** English

Abstract

In order to secure a profitable business, stores must have enough products on shelves to offer to customers but shelf life of the products as well as available inventory space should also be considered. Optimizing the product flow from stores to customers is a critical part in supply chain management and is necessary to retain a competitive edge. RELEX Solutions offers a software platform for supply chain management.

The software offered by RELEX allows store managers to calculate demand forecasts and order proposals for each product. Calculations in the software produce log files that contain information about the current calculation. Occasionally there are errors in these calculations which are also logged in the log file. Currently a designated support team at RELEX goes through the error messages and decides what actions need to be taken according to the criticality of each error. This thesis investigates the possibility of using a machine learning system to separate critical and non-critical issues.

The raw text data in the form of error messages is first preprocessed so it is suitable for a machine learning system. The preprocessing stage includes separating the text data into individual words and filtering out irrelevant terms. Two different feature representations were studied. Selected algorithms for the text classification were support vector machines and a naive Bayes classifier. A systematic testing approach was constructed in order to find the best performing classifier.

It was found that the original data set had a strong class imbalance that deteriorated the results. A balanced data set was constructed and the models were able to obtain a better classification performance with this set. However even with a balanced data set the classifier accuracy was only around 60% with both algorithms.

This study concluded that the structure of the error messages makes the classification challenging. The error messages do not have enough separating features, they are too messy and the error messages can look similar even if they stem from a different issue. Future research should focus on improving the error messages. In this thesis the numerical data was filtered out but it could be interesting to study the effect of numbers on log classification. Different algorithms should also be researched in the future.

Keywords Text classification, machine learning, support vector machines, naive Bayes classifier, Apache Spark, data mining

Aalto University
School of Electrical Engineering

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Aapo Rantala

Työn nimi Koneoppimisen käyttäminen automaattisessa virheentunnistuksessa ohjelmiston lokitulosteesta

Koulutusohjelma Automaatio ja sähkötekniikka

Pääaine Translationaalinen tekniikka **Pääaineen koodi** ELEC3023

Työn valvoja Apulaisprofessori Arno Solin

Työn ohjaaja TkT Tuomas Viitanen, TkL Tapio Pitkäranta

Päivämäärä 22.5.2019 **Sivumäärä** 54+4 **Kieli** Englanti

Tiivistelmä

Kannattavan liiketoiminnan varmistamiseksi kaupoissa pitää olla tarpeeksi tuotteita hyllyillä asiakkaille, mutta myös tuotteiden säilyvyysaika sekä käytettävissä oleva varaston tila pitää ottaa huomioon. Tuotevirran optimoiminen kaupoista asiakkaille on olennainen osa toimitusketjun hallintaa ja se on välttämätöntä kilpailuedun säilyttämiseksi. RELEX Solutions tarjoaa ohjelmistoalustan toimitusketjun hallinnalle.

RELEXin tarjoamalla ohjelmistolla myymäläpäälliköt voivat laskea kysyntäennusteita sekä tilausehdotuksia jokaiselle tuotteelle. Laskutoimitukset ohjelmistossa tuottavat lokitiedostoja, joissa on tietoa kyseisestä laskutoimituksesta. Toisinaan näissä laskutoimituksissa tapahtuu virheitä, jotka kirjataan myös lokitiedostoon. Tällä hetkellä asiakastukitiimi RELEXillä käy läpi virheviestejä ja päättää minkälaisia toimia pitää tehdä, riippuen jokaisen virheen kriittisyydestä. Tämä diplomityö tutkii mahdollisuutta käyttää koneoppivaa järjestelmää kriittisten ja ei-kriittisten ongelmien erotteluun.

Raaka tekstidata virheviestien muodossa esikäsitellään ensin niin, että se on sopivaa koneoppivalle järjestelmälle. Esikäsittelyvaihe sisältää tekstidatan erottelun yksittäisiin sanoihin sekä merkityksettömien termien poissuodattamisen. Kahta erilaista tapaa piirteiden esitysmuodoksi tutkittiin. Valitut algoritmit tekstiluokittelulle olivat tukivektorikoneet sekä naiivi Bayes-luokittelija. Testaukseen muodostettiin systemaattinen menettelytapa, jotta parhaiten suoriutuva luokittelija löydettäisiin.

Tutkimuksessa saatiin selville, että alkuperäisessä tiedostossa oli vahva luokkaepätasapaino joka heikensi tuloksia. Tasapainotettu joukko muodostettiin ja mallit pystyivät saavuttamaan paremman luokittelutehokkuuden tällä joukolla. Kuitenkin jopa tasapainotetulla joukolla luokittelutarkkuus oli vain noin 60% molemmilla algoritmeilla.

Tutkimuksessa havaittiin, että virheviestien rakenne tekee luokittelusta haastavaa. Virheviesteillä ei ole tarpeeksi erottavia piirteitä, ne ovat liian sotkuisia ja virheviestit voivat näyttää samalta vaikka ne johtuisivatkin eri ongelmasta. Jatkotutkimuksen pitäisi keskittyä virheviestien parantamiseen. Tässä diplomityössä numeerinen data suodatettiin pois ja voisi olla mielenkiintoista tutkia numeroiden vaikutusta tekstiluokitteluun. Muita algoritmeja pitäisi myös tutkia tulevaisuudessa.

Avainsanat Tekstiluokittelu, koneoppiminen, tukivektorikoneet, naiivi Bayes-luokittelija, Apache Spark, tiedonlouhinta

Preface

This thesis couldn't have been completed by myself alone. First I would like to thank my supervisor Arno Solin and my advisors Tuomas Viitanen and Tapio Pitkäranta for their extraordinary support during this thesis. They gave me wonderful insight into my thesis and their helpful comments always guided me in the right direction. I also want to thank my colleagues at RELEX for supporting me during this project. Finally I want to thank Aalto University and especially Annika Salama without whom none of us would ever graduate.

Kahdeksaan vuoteen yliopistoelämää mahtuu lukematon määrä kokemuksia ja ennen kaikkea tärkeitä ihmisiä. Opiskelu on ollut niin mukavaa, että välillä tahtia on pitänyt vähän hidastaa. Nyt valmistuminen alkaa kuitenkin jo häämöttää, mutta otan sen avosylin vastaan. Kaikki ASkissa lasketut laskarit, pitkälle aamuyöhön venyneet tapahtumajärkkäilyt sekä vaihto Japanissa ovat johtaneet tähän pisteeseen. Matka ei olisi kuitenkaan ollut mitään ilman muita ihmisiä. Kaikkia en tässä varmastikaan pysty kiittämään, mutta ne joita tässä ei erikseen mainita ovat myös kiitoksensa ansainneet.

Kiitos Joutomiehet. Opetitte, että koti on aina lähellä, perhe ei ole pelkkiä sukulaisia ja että kaljaa kannattaa muistaa juoda aina silloin tällöin.

Kiitos Kuokkijat. Opetitte palstaviljelyn saloja sekä kokoustekniikkaa. Palataan jatkossakin kohtaan kolme.

Kiitos AS ja Inkubio. Opetitte, että yhteistyössä on voimaa ja että kiltarajat on tehty rikottaviksi. Kiitos YJ'12, ASh'13, !111111, IE'13, JTM3, RATTO'15, LuTku'16, Tempaustoimikunta sekä lukemattomat kiltojen toimikunnat.

Kiitos Jo ja J.R.R. Lupasin ensimmäisen teille.

Kiitos äiti, isä ja veljet kaikesta tuesta ja olemassaolosta.

Ennen kaikkea suuri kiitos kaikille rakkaille ystäville. Teette arjestani juhlaa, ilahdutatte minua päivästä toiseen ja teette minusta paremman ihmisen.

Otaniemi, 22.5.2019

Aapo Rantala

Contents

Abstract **3**

Abstract (in Finnish) **4**

Preface **5**

Contents **6**

Symbols and abbreviations **8**

1 Introduction **9**
1.1 RELEX 10
1.2 Background and motivation 10
1.3 Research question and objectives 11
1.4 Scope of the research 12
1.5 Research methods 12
1.6 Structure of the thesis 13

2 Background **14**
2.1 Performance evaluation 14
2.2 Data representation 17
2.3 Supervised learning 19
2.4 Unsupervised learning 24
2.5 Text classification 25

3 Materials **28**
3.1 Logging in the RELEX Planning cloud 28
3.2 Kayako 30
3.3 Log data 31

4 Methods **33**
4.1 Apache Spark 33
4.2 Support Vector Machines 34
4.3 Preprocessing 35
4.4 Performance comparison 36

5 Results **39**
5.1 Results with Support Vector Machines 39
5.2 Comparison with a naive Bayes classifier 43

6 Discussion **45**
6.1 Challenges and how to improve 47

7 Summary **49**
7.1 Suggestions for future research 51

References **52**

A Ruby log **55**

B Java log **56**

C Error example **57**

Symbols and abbreviations

Abbreviations

AUC	Area under the receiver operating characteristics curve
FN	False negative
FP	False positive
FPR	False positive rate
KNN	K-nearest neighbour
NLP	Natural language processing
PCA	Principal component analysis
ROC	Receiver operating characteristics
SMOTE	Synthetic minority oversampling technique
SVM	Support vector machine
TF	Term frequency
TF-IDF	Term frequency - inverse document frequence
TN	True negative
TP	True positive
TPR	True positive rate
WSD	Word sense disambiguation

1 Introduction

It is essential for stores and warehouses to keep track of their inventories so that enough products are available to the customers. Similarly it is important to keep track of the shelf life of products so that the stores don't stock up unnecessary amounts of fresh produce that can go to waste. When deciding what products to order from the suppliers the store managers need to consider several factors including current inventory levels, the forecasted demand for the products and the shelf life of the products. The flow of products from the suppliers to the customers through the stores is called the supply chain.

In today's competitive market it is important that stores optimize the supply chain in order to retain their competitive edge. To secure a profitable business model and to maximize profits the stores need a constant flow of products through the supply chain. In order to stay ahead of competition efficient supply chain management is required.

Supply chain management seeks to achieve linkage and coordination between suppliers, customers and the organization itself. In his book Christopher [1] states that for example one goal of supply chain management might be to reduce or eliminate the buffers of inventory that exist between organisations through the sharing of information on demand and current stock levels. Christopher defines supply chain management as "the management of upstream and downstream relationships with suppliers and customers in order to deliver superior customer value at less cost to the supply chain as a whole." Thus successful supply chain management leads to a more profitable outcome for all the parties in the chain with the management of the relationships. [1]

RELEX Solutions offers a platform for supply chain management. The solution provides clients advanced forecasting and replenishment computing. Each customer has their own environment where they can calculate for example order proposals and forecast future sales. These calculations can either be set to be run automatically for example during the night or manually whenever the user wishes to. It is typical that most of the calculations are run during the night so that new order proposals are available the next day. The software produces logs for each action run in the software and some log files can contain hundreds of thousands of lines depending on the type of action.

The log file contains miscellaneous information about the action. Most of the lines contain information mostly useful only to the developers of the software for debugging purposes. Sometimes an error occurs during the run and this is also logged into the log file. These errors can vary in severity from mainly a warning to a production critical error. As RELEX is committed to delivering measurable value to the customers through efficient supply chain management it is important that these error messages are handled properly and that the customers can rely on the software. If the software is not working properly it may cause the customers to lose income. Similarly RELEX also aims to minimise the use of resources so therefore parsing the log files shouldn't take too much time. It is desirable to streamline the log parsing process as much as possible and having an efficient tool to go through the logs brings measurable value also to RELEX.

1.1 RELEX

RELEX was founded in 2005 and currently employs over 500 employees based in offices around Europe and the US. The company has over 100 customers in more than 17 countries. Thousands of stores, hundreds of warehouses and billions of euros are managed with RELEX's solutions. RELEX Solutions offers a platform for supply chain management and optimization. The solution provides clients advanced forecasting and replenishment computing. RELEX also offers space and assortment management as well as workforce optimization. RELEX is a growing company and the customer base is growing with each customer having their own specific needs.

RELEX uses a columnar database and fast in-memory calculations. Together with parallelization and aggressive compression of data even very large amounts of data can be handled with enough efficiency. This allows RELEX to reach very good results. Inventory value is decreased, waste is reduced, shelf availability nearly doubles, out-of-stocks are reduced, staff time spent on replenishment is cut and transportation and handling costs are reduced significantly. All of these in turn increase sales and income for the customers.

The supply chain management software offered by RELEX is called the Planning cloud. The Planning cloud is used to calculate the desired optimizations according to data inputted by the customer. The customer first uploads the necessary data including for example sales data, information about campaigns and inventory levels. The user can then specify the parameters for sales forecast and order proposal calculations including the desired time periods. These calculations are run with tools called actions. The actions can be run manually or scheduled to be run at a desired time. It is customary to run performance intensive actions during the night as so called nightly runs so that the order proposals are ready the next morning and also because the actions may slow the environment down it is best to run them after working hours.

1.2 Background and motivation

Currently a designated support team in the company goes through the errors reported in the log files produced by the software and decides whether or not the incidents are critical to the operation of the system and if so, inform the necessary personnel. This is naturally quite laborious as the process of handling each error message depends on several variables such as the action run that produced this log, the customer this environment belongs to and the software version. The support team uses several hours each day to go through error messages produced during the previous night.

Having an automatic incident detection system would allow the employees to focus on more pressing matters. The software currently produces a lot of unnecessary error messages. Some of these might be production critical whereas others could simply be warnings. As things stand now it is the support team's responsibility to decide the severity of each error message and whether or not some actions need to be taken to counter that error. A system that pre-labels the error messages as being either critical or needing immediate attention would allow the support team to spend less time on investigating the severity of each issue. Thus such a system would bring measurable value to the company.

Machine learning is currently a popular research field and it has been studied extensively with various approaches. Handling big data and log mining are important topics as companies produce massive amounts of log data daily. Efficient text classification is important to keep the system operational and to respond to error messages in time. Thus this topic has a great scientific significance as well. As log data continues to expand even more powerful methods are needed.

1.3 Research question and objectives

The objective of this thesis is to investigate the possibility of using machine learning in automatic incident detection from error messages in software log output. Having a computer identify possible issues is naturally much faster than having humans to go through the logs manually. Thus the aim of using such a system would basically be to save time. In other words, the objective then is that such a machine learner should be able to correctly predict the class of each error message reliably to be considered useful. If the system fails to correctly recognize critical issues the software usability is compromised. Similarly if the system predicts non-critical issues to be critical this leads to excessive workload for the human personnel who have to go through the critical issues.

We want to deepen the knowledge on text classification by finding out what is the most efficient method in classifying error messages in this kind of an application. The machine learner would allow the employees to spend their time elsewhere and thus it would bring measurable value to the company. Based on these objectives we can form the research questions as follows:

Question 1 *Can we use machine learning to reliably classify error messages found in log files?*

We want to find out if it's possible to configure a machine learner so that it classifies error messages in log files according to their criticality with decent performance. The resulting classification predictions should be usable by the support team personnel so that the classification is correct and issue-criticality can be handled accordingly. In order for the machine learner to be considered efficient, the performance of the classifier should be evaluated with different metrics that can be compared with human performance.

Question 2 *What kind of a machine learning system has the best classification performance?*

Classification performance is dependent on several variables including the data set, the characteristics of the input data and the chosen algorithm. We should form different data sets to compare how class balance and instances with different features in the training data affect the classification. To find out how the model reacts to different input data the error messages should be preprocessed to a varying degree before inputting them to the model. We want to find out if the data is usable as is or if we need to preprocess it to achieve reasonable classifier performance. Lastly we compare two algorithms to find which one has better performance for this certain application. Different approaches and conditions should be considered to find the classifier with the best performance.

Question 3 *Does the system produce measurable value, and if so, how much and how is it measured?*

Lastly we want to find out if the machine learning system is actually usable and if the predictions made by the system are comparable to human assessment. One type of measurable value is time saved. Would the predictions made by the machine learner be reliable enough so that time is actually saved or if humans would still need to double check the predictions thus not saving that much time after all?

1.4 Scope of the research

This thesis handles the log data of various RELEX customers. The scope includes log parsing and text classification applying machine learning. More specifically the classification problem concerns classifying the error messages from customer environments. The log data was gathered over a period of three months. Two different machine learning algorithms are compared to find the optimal one for this application. The models are systematically tested with different data sets to find a solution that has the best performance. Real-time log analysis is not in the scope of this thesis but instead the log data is handled in batches for every test setup.

1.5 Research methods

The machine learner is implemented with Apache Spark. Spark is an open source analytics engine for large-scale data processing that can be used for example to analyze log data. Spark also has an extensive machine learning library that is used. When it comes to text classification, Support vector machines (SVMs) [2] are nowadays widely used and considered efficient. In this thesis SVM is used to classify error messages into two classes. We can find the model with the best classification performance by varying the parameters of the SVM when training the model or by preprocessing the input data differently before training the model. Preprocessing includes for example removing stop words from the input data, sampling the data set to balance the class division or using different techniques to build the feature vector. A systematic testing approach is used to determine the best possible classifier for this application.

To estimate classifier performance accuracy, balanced accurary, precision, recall and the area under the receiver operating characteristics curve (AUC) are used. By using the different methods listed above to optimize the classification result we can find the best combination of algorithm parameters and preprocessing. Once the optimal SVM model has been discovered we can build a naive Bayes classifier using the same data preprocessing to compare if using a different algorithm affects the classifier performance.

1.6 Structure of the thesis

This Master's thesis is organized as follows. First in Chapter 2 the background of this work is explained regarding basic principles of machine learning, performance evaluation, data representation, supervised learning and unsupervised learning. Support vector machines and text classification will be explained in more detail as they are key elements in this thesis. In Chapter 3 the materials used in this thesis are presented. After this in Chapter 4 the methods used in this thesis will be discussed with regard to the materials presented in the previous chapter. In Chapter 5 the results of the thesis are presented and in Chapter 6 there is some discussion based on these results. Chapter 7 concludes the thesis where a summary of the thesis is presented along with suggestions for future research.

2 Background

The field of machine learning investigates the question of how to construct computer programs that automatically improve with experience. Mitchell [3] explains how already in the 1990's there were several achievements in the field: recognizing spoken words, detecting fraudulent use of credit cards and driving autonomous vehicles on public highways. Nowadays these are becoming even more commonplace as shown by for example Tesla's commercially available self-driving cars.

Mitchell [3] defines learning as performing some class of tasks T with performance measure P while gaining experience E. A computer program then learns if its performance P at tasks in T improves with experience E. When designing a learning system we first need to consider the training experience. For example it should be noted how well the training experience represents the final test data. The system needs to be quantified in such a way that the performance of the system improves based on certain rules.

The task T can be anything from speech recognition or natural language processing to driving autonomous vehicles or pattern recognition in images. In this thesis the task is to identify critical incidents in the software by interpreting the log files. In essence the task is a classification problem because we want to classify the error messages in the log files to critical and non-critical classes.

The experience E is the set of rules upon which the algorithm relies on. In supervised learning experience comes from humans in the form of pre-labeled data points. In unsupervised learning the algorithm learns from the features of the data set alone without any human interference.

The performance P is a measure for evaluating how well the learner is executing it's task T. By evaluating the performance of the model we can find the optimal result. To find the optimal result we need to consider several different factors and different performance metrics.

2.1 Performance evaluation

According to Batista et al. [4] "The most straightforward way to evaluate the performance of classifiers is based on the confusion matrix analysis." Table 1 shows a confusion matrix for a two-class problem with class 1 and class 2. In this table we find the following terms:

- True positive: The number of class 1 instances correctly classified as belonging to class 1.
- False negative: The number of class 1 instances misclassified as belonging to class 2.
- False positive: The number of class 2 instances misclassified as belonging to class 2.
- True negative: The number of class 2 instances correctly classified as belonging to class 1.

Table 1: Confusion matrix for a two-class problem. Modified from Batista et al. [4].

	Predicted class 1	*Predicted class 2*
Actual class 1	True positive (TP)	False negative (FN)
Actual class 2	False positive (FP)	True negative (TN)

These values can be used to build different performance evaluating metrics. In this case the performance of classification can be measured for example with accuracy, precision and recall. From these metrics we can also deduce the area under the receiver operating characteristics (ROC) curve (AUC) which is also used to evaluate model performance. Because this thesis considers the classification problem these metrics are presented here in terms of classification.

Accuracy

Accuracy is the ratio of the number of correctly classified items to the total number of items as shown in Equation (1). It's a widespread effectiveness measure in machine learning to evaluate a classifier's performance. It's important to note that accuracy is not reliable if the data set is imbalanced. For example if the data set contains 95 class 1 samples and 5 class 2 samples, simply classifying all items as class 1 gives 0.95 accuracy. The error rate which is defined as $(1 - \mathit{Accuracy})$ is also a common evaluator used in machine learning [5]

$$\mathit{Accuracy} = \frac{\sum \mathrm{TP} + \sum \mathrm{TN}}{\sum \text{Total population}}. \tag{1}$$

In case the data set is skewed the accuracy metric is not reliable. In this case we can use balanced accuracy which aims to take into account the class imbalance in the data set. We can achieve a high accuracy easily with a skewed data set by simply predicting the overrepresented class more often but this doesn't explain how effective the model is in reality. Balanced accuracy seeks to correct this by taking into account the class division. The formula for balanced accuracy can be derived from accuracy in Equation (1) and we get

$$\mathit{Balanced\,accuracy} = \left(\frac{\sum \mathrm{TP}}{\sum \mathrm{TP} + \sum \mathrm{FP}} + \frac{\sum \mathrm{TN}}{\sum \mathrm{TN} + \sum \mathrm{FN}} \right) / 2. \tag{2}$$

Precision

Precision describes how many of the selected items are actually relevant. In a classification task precision is the number of items correctly classified to class X divided by the number of items classified to class X altogether as shown in Equation (3). Simply put, precision tells how useful the results are. Precision is also known as the

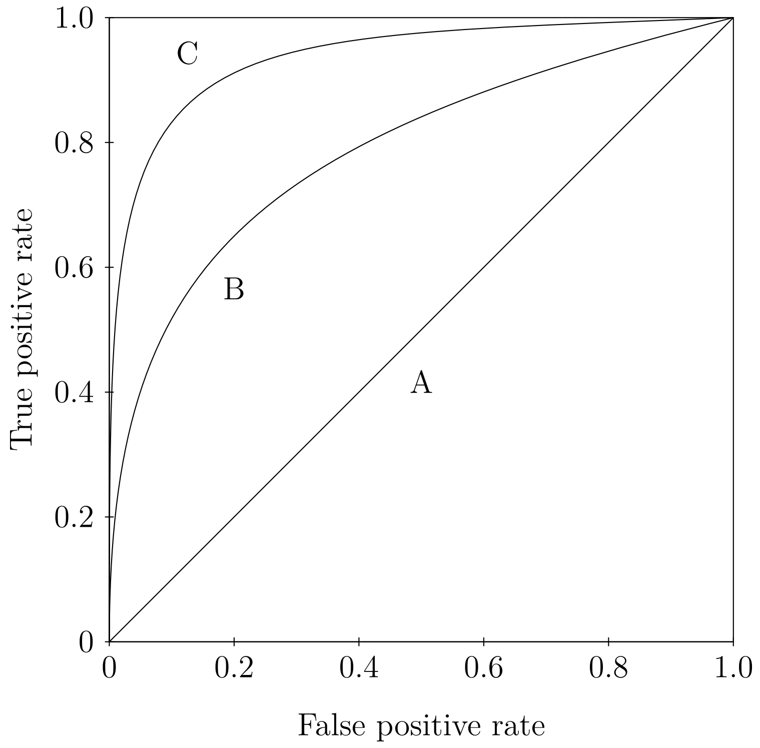


Figure 1: An example of three ROC curves. The AUC of curve A is 0.5 which means that the model does not perform particularly well with new data but instead it's performance seems quite random. Curve B has better AUC compared to curve A. Curve C has a very high AUC and this model's performance would be superior to models A and B.

positive predictive value [5]

$$Precision = \frac{\sum \mathrm{TP}}{\sum \mathrm{TP} + \sum \mathrm{FP}}. \tag{3}$$

Recall

Recall describes how many relevant items are selected. In a classification task recall is the number of items correctly classified to class X divided by the number of items belonging to class X altogether as shown in Equation (4). Simply put, recall tells how complete the results are. Recall is also known as the true positive rate (TPR) [5]

$$Recall = \frac{\sum \mathrm{TP}}{\sum \mathrm{TP} + \sum \mathrm{FN}}. \tag{4}$$

Area under the receiver operating characteristics curve

According to Huang and Ling [6] the receiver operating characteristics (ROC) curve was used to represent the trade-off between the hit rates and false alarm rates already in the 1970s. The area under the ROC curve (AUC) provides a good summary for the performance of the ROC curves. One way to think about AUC is that it describes how well the model adjusts to new unforeseen data. The bigger the value of AUC the better the model can perform with foreign data. The ROC curve is plotted with the TPR or recall against the false positive rate (FPR) where TPR is on the y-axis and FPR is on the x-axis. FPR is defined as [6]

$$FPR = \frac{\sum \mathrm{FP}}{\sum \mathrm{FP} + \sum \mathrm{TN}}. \tag{5}$$

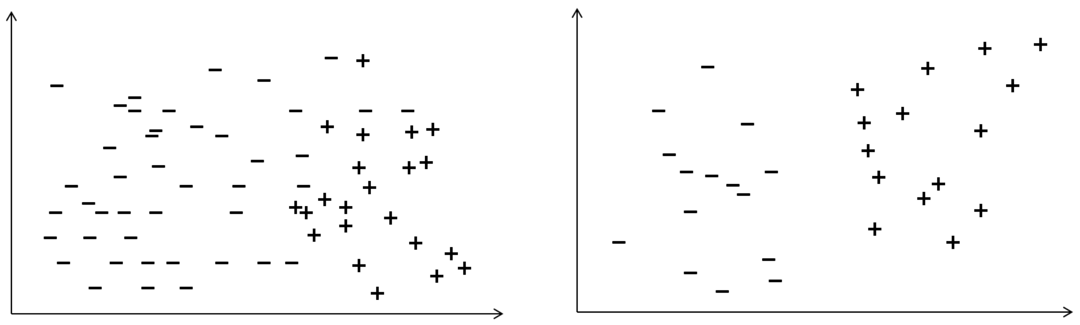
An example of three ROC curves can be seen in Figure 1. The AUC of curve A is 0.5 which means that the model does not perform particularly well with new data but instead it's performance seems quite random. In a classifier this would mean that the model doesn't have a good understanding of the classes so it would classify instances randomly. Curve B has better AUC compared to curve A and for example in a classification task this model would perform much better. Curve C has a very high AUC and this model's performance would be superior to models A and B. The most optimal model would have a ROC curve that goes right by the diagram's upper-left corner where the AUC is 1.0. ROC and AUC don't explain much about a model's results in an individual task but rather offer an important tool to estimate the overall performance of the model over different data.

2.2 Data representation

The data used in machine learning consists of instances represented using the same set of features. To find these features the data first needs to be preprocessed, this stage is also called feature extraction to produce the desired feature representation. We can form a so-called feature vector for each instance in the data set where the contents of the feature vector represent each instance's features. The features can be continuous, categorical or binary. The task of the machine learning system is then to classify each instance to corresponding correct outputs based on each instance's features. [7]

All the feature vectors of all of the instances in the data together form a feature space. The dimensions of the feature space are determined by the size of the feature vectors, in other words if the data has 100 features then the resulting feature space is 100-dimensional. This leads to a phenomenon first described by Bellman [8] as the Curse of dimensionality. As the number of features increases so does the size of the corresponding feature space resulting in a situation where the sparsity of the data actually increases despite having more features. However as features hold information the simple solution to ignore some features in favor of a smaller feature space would result in ignoring a part of the data and thus possibly resulting in a categorization error or bias.

The data that is used in machine learning is first divided into training data and test data. The purpose of the training data is to train the model and then the test data is used to independently verify the results of the model. Generally speaking a bigger training data set provides the model with a larger variety of instances with



(a) Imbalanced data set

(b) Balanced data set

Figure 2: An example of a class imbalance. (a) Many negative instances with only a few positive class instances. The data set also has some degree of class overlapping. (b) A balanced data set with well-defined clusters.

different features. On the other hand the test set should also be sufficiently large so that there is enough variation in the test data instances as well to test possible outcomes thoroughly. Typically data is then divided into 70–80% training data and 20–30% test data. Another method for data division is k-fold cross validation that produces k different divisions of test and training data. This can be used to validate the results of the model as the training and test data is different each time. Schlimmer and Fisher [9] present a concept for incremental learning where the model is continuously trained as new training data emerges. This is opposed to the idea that the model is trained once with a fixed training data set. There are advantages and drawbacks to both incremental and non-incremental learning mainly concerning the prediction reliability and the cost of updating memory. Incremental learning is useful if the characteristics of the data vary over time but even then the amount of necessary data to be stored should be examined carefully.

When choosing the data for the machine learner the uniformity and the composition of the data should be carefully considered. Japkowicz and Stephen studied the class imbalance problem extensively in [10]. According to Japkowicz and Stephen "the class imbalance problem corresponds to the problem encountered by inductive learning systems on domains for which one class is represented by a large number of examples while the other is represented by only a few." An example of a class imbalance is presented in Figure 2. If the training data set is imbalanced then the model will be biased towards a certain class and as mentioned earlier the accuracy of the system is not reliable either if the data set is imbalanced. Japkowicz and Stephen propose for example under-sampling to counter this imbalance where elements in the over-sized class are eliminated until it matches the size of the smaller one. The drawback to this method is naturally that we lose information and we eventually have a smaller usable data set. The study found that different classifiers reacted differently to varying degrees of class imbalance. Nevertheless the possibility of a skewed data set should be considered when building a machine learner. [10]

2.3 Supervised learning

Kotsiantis et al. [7] define supervised learning as “the search for algorithms that reason from externally supplied instances to produce general hypotheses, which then make predictions about future instances.” Supervised learning is the process of learning a set of rules from already labeled instances in the training data. Having already labeled data is a key to supervised learning which means that it requires human interaction. A specialist is needed to evaluate instances of data and label them based on their own judgement. Supervised learning can be divided roughly into two separate categories: regression and classification. Regression analysis means approximating a real-valued target function [3]. In other words regression aims to find a certain trend or pattern among the features of known instances. Classification on the other hand tries to find similarities in the instances based on their features and classify new instances based on the knowledge gained from known instances. In the simplest form there are two classes but multiclass classification is also possible. The problem in this thesis concerns text classification and therefore regression won’t be discussed further. With classification the machine learner or the classifier is used to assign class labels to the testing instances where the values of the features are known but the value of the class label is not. The object is to create a classifier that can be used to classify new instances based on the learned set of rules. [7]

A typical process for applying supervised learning can be seen in Figure 3. The same basic principles of supervised learning presented here apply both to regression and classification but for clarity only classification is concerned in the examples that follow. First the necessary data is identified and gathered in a form that’s usable for the system by performing required preprocessing. The data set is then divided into the training set and the test set as explained before. After the algorithm has been selected the model is then trained with the training set and the performance of the model is evaluated with the test set. If the performance is not satisfactory the previous steps can be reconsidered as shown in Figure 3 accordingly. [7]

Several different algorithms have been developed for supervised learning. Next we will go through a couple while listing their characteristics and applications. We will briefly discuss decision trees and the nearest neighbour algorithm. In addition we will cover naive Bayes classifiers that are used in this thesis. We will also go through support vector machines in greater detail as they are utilised heavily in this work.

Kotsiantis et al. [7] define decision trees as trees that classify instances by sorting them based on feature values. Each node in a decision tree represents a feature to be classified and each branch represents a value that the feature can assume. Classification starts at the root node and the instances are sorted based on their feature values. The root node represents the feature that best divides the training data. One of the most useful characteristics of decision trees is that they are comprehensible to humans. It is easy to understand why a certain instance is classified to a specific class. Decision trees are typically used when the data to be classified is discrete or categorical. [7]

K-Nearest Neighbour (kNN) algorithm is based on the idea that the feature vectors of each instance in the data set define a certain point in the feature space.

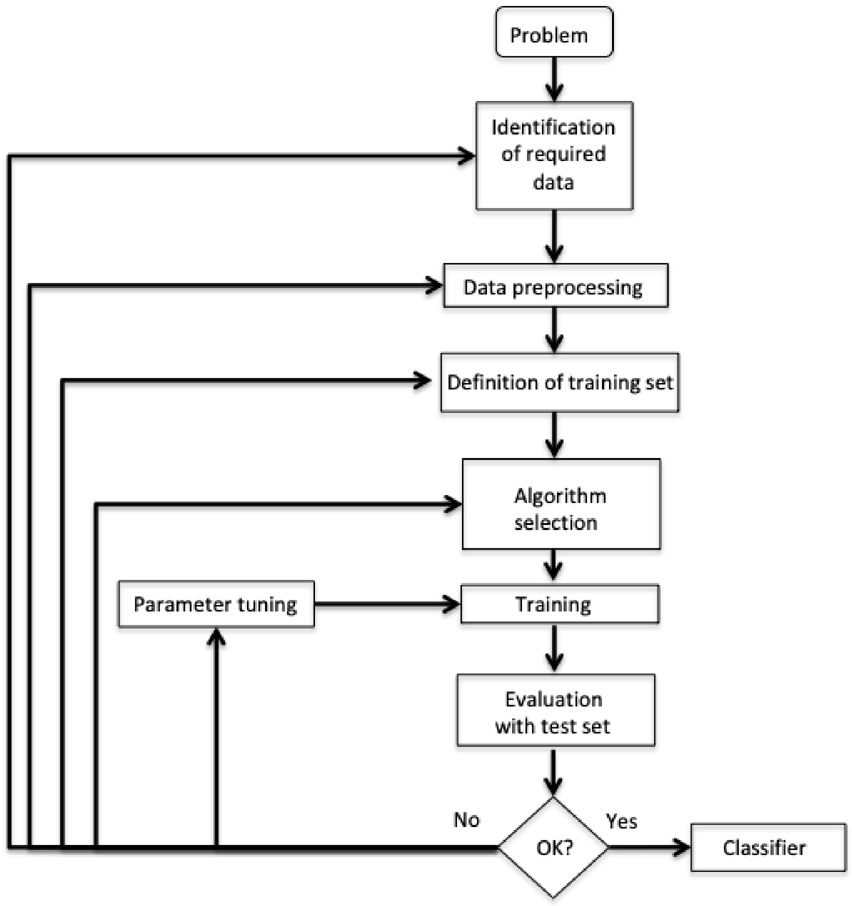


Figure 3: The process of supervised machine learning.

The main principle behind kNN is that these points will generally exist in close proximity to other instances that have similar properties. Thus if the instances are labeled with a class label then the class of an unclassified instance can be determined by observing the class of its nearest neighbours. The kNN finds the k nearest instances to the unclassified instance and by finding the most frequently occurring class labels among its neighbours, assigns this label to that instance as well. The dimensions of the feature space are determined by the number of features which means that this algorithm is computationally very intensive with many features and large feature vectors. For example this means that kNN is not suitable for text classification as each individual word represents one feature. Key points in improving the results of kNN is the choice of k and the approach to define the distance between two instances. [7]

One of the simplest models for classification is the naive Bayes classifier because it assumes that all the features of each instance are independent of each other in the context of the class. This is the so-called “naive Bayes assumption” and even though this assumption clearly does not hold in most real-word tasks, naive Bayes often performs very well. The classifier learns from training data the conditional probability of each feature A_i given the class label C and the classification is then done by applying Bayes rule to compute the probability of C given the particular values of $A_1, \ldots, A_n$ and then predicting the class with the highest posterior probability [11]. The Bayes rule is

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{6}$$

where $P(A|B)$ is the conditional probability of event A occurring given that B is true and vice versa for $P(B|A)$. $P(A)$ and $P(B)$ are the probabilities of observing A and B independently of each other.

Because of the independence assumption the conditional probabilities of each feature can be learned separately which simplifies learning. There are different approaches to naive Bayes classification for example the Bernoulli model where the feature vector is binary signifying whether a certain word occurs in the document or not, and the multinomial model where feature vectors contain information about word frequencies in the document. [12]

Naive Bayes classifiers have been used for text classification and for example McCallum and Nigam [12] achieved good classification performance. In the case of text classification using the naive Bayes assumption “the probability of each word event in a document is independent of the word’s context and position in the document.” Using Equation (6) we can then calculate the posterior probability of each class for test documents. In this thesis a multinomial naive Bayes model is used to classify error messages. The classification results obtained with support vector machines are then compared to the naive Bayes classifier results.

Support vector machines (SVMs) were first introduced by Cortes and Vapnik in [2]. The SVM is a binary classifier that uses a hyperplane to find the optimal solution for the classification problem. SVM has also been extended to solve regression problems to estimate an unknown real-valued function. There are several studies on the applications of SVM for regression analysis and for example Cherkassky and Ma [13] studied the parameter selection for SVM regression. SVM regression won’t be discussed further and instead the focus of this thesis is on the classification aspect of SVMs.

The feature vectors of the data are first mapped to the feature space defined by the dimensions of the feature vectors. A linear decision surface called the hyperplane is then constructed in this feature space. The optimal hyperplane is defined as the linear decision function with maximal margin between the two classes. It is enough to consider only a small amount of training data, or the so called support vectors, to determine this margin as can be seen in Figure 4. [2]

Cortes and Vapnik offer the following reasoning for finding the optimal hyperplane. The set of labeled training patterns

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l), \quad y_i \in \{-1, 1\}, \tag{7}$$

where vectors $\mathbf{x}_i$ correspond to vectors in the training data is said to be linearly

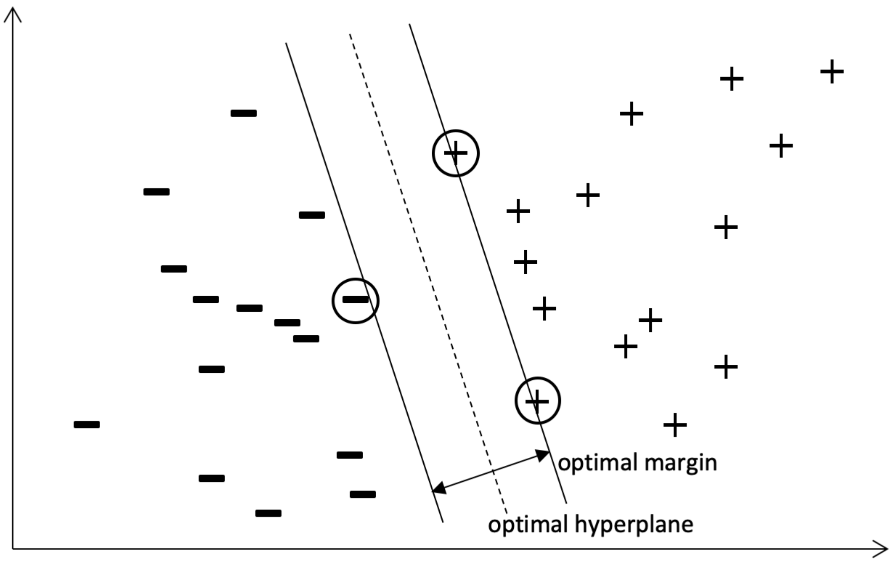


Figure 4: An example of finding the optimal hyperplane with SVMs. The support vectors of both classes are marked with circles and they define the margin of the largest separation between the two classes.

separable if there exists a vector $\mathbf{w}$ and a scalar b such that the inequalities

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i + b \geq 1 \quad \text{if} \quad y_i = 1, \\ \mathbf{w} \cdot \mathbf{x}_i + b \leq -1 \quad \text{if} \quad y_i = -1, \end{aligned} \tag{8}$$

are valid for all elements of the training set in Equation (7). We can write the inequalites in Equation (8) as:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad i = 1, ..., l. \tag{9}$$

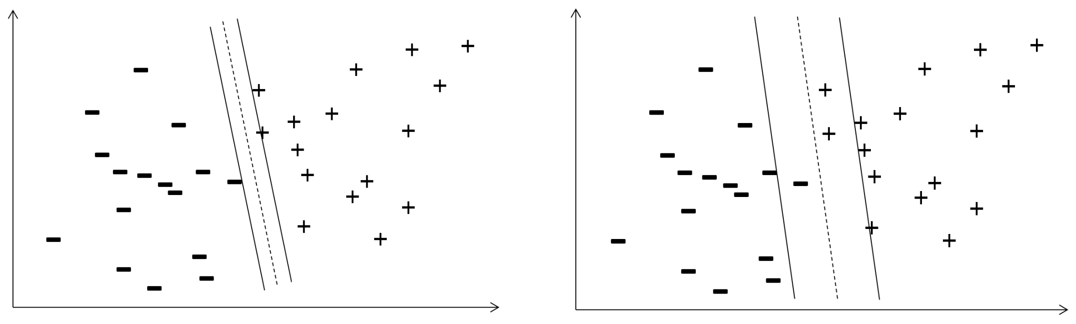
The optimal hyperplane,

$$\mathbf{w}_0 \cdot \mathbf{x} + b_0 = 0, \tag{10}$$

is the unique hyperplane that separates the training data with a maximal margin as seen in Figure 4.

In a case where the training data cannot be separated without misclassifying some instances the margin needs to be reconsidered. In this case we want to separate the training set with a minimal number of misclassified instances. The margin between the two classes can be thought of as either a hard margin, where misclassified instances are not allowed but the margin is small, or a soft margin, where some misclassified instances can be accepted if it leads to a larger margin between the classes. [2]

If the instances that are to be misclassified are excluded from the training set we can separate the remaining part of the set without issues. To separate the problematic part of the training data we can construct an optimal separating hyperplane. Cortes and Vapnik present the constant C or the so-called misclassification penalty [2]. Because SVM defines the plane that maximizes the distance to the support vectors



(a) A large value of C (b) A small value of C

Figure 5: An example of two classes separated by a hyperplane and the effect of C on the margin. In (a) the margin is very small because the few data points near the class boundary dictate the position of the hyperplane. In (b) we can ignore the points closest to the boundary and we get a separating hyperplane with a much bigger margin.

we can define SVM using the hinge loss function even in the case where the classes are not linearly separable. We get

$$max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x_i} + b)) \tag{11}$$

to which we can add the regularization parameter C to get

$$max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x_i} + b)) + C||\mathbf{w}||^2. \tag{12}$$

Using this we can reduce the effect of individual misclassified instances and therefore reach a better model that maximizes the margin according to general consensus of all the datapoints rather than individual instances. The effect of C can be clearly seen in Figure 5. With large values of C the resulting margin is small and misclassifications are not tolerated. A smaller value allows to ignore points close to the boundary and results in a larger margin.

Typically the data used in SVMs is not linearly separable. In this case we can apply kernel methods to the input vectors of the SVM so that they are non-linearly mapped to a very high dimensional feature space. The transformed data in this feature space then often has a shape that can be handled using simple methods such as linear algorithms. The so-called kernel trick makes it possible to perform operations in the input space rather than the new high dimensional feature space through the use of kernel functions. This enables effective computation because it is enough to consider the smaller dimension space. [14]

SVMs are able to achieve good performance in several different applications. One element that makes SVM so effective is that because of the support vectors it is enough to consider the instances close to the boundary between the classes. According to Joachims [15] “one remarkable property of SVMs is that their ability to learn can be independent of the dimensionality of the feature space.” SVMs don’t necessarily care about the number of features but instead consider the margin with

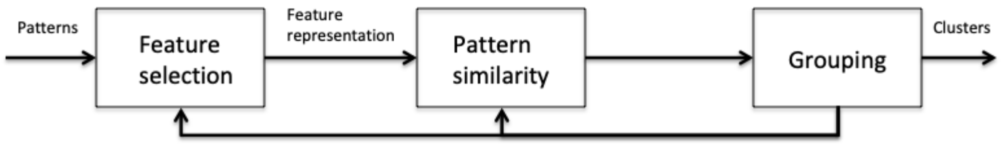


Figure 6: Typical stages in a clustering algorithm. Feature selection aims to find the most effective subset of the original features to use in clustering. The next step is to find similarities in the pattern for example by measuring pattern proximity. Finally the grouping can be either hard or fuzzy.

which they separate the data. SVMs have been used extensively in classification tasks such as spam categorization [16], image classification [17] and predicting protein secondary structure [18]. In addition SVMs have been used successfully in text classification as shown for example by Joachims [15] and Dumais et al. [19].

2.4 Unsupervised learning

As opposed to supervised learning, in unsupervised learning there is no human input that provides some sort of labeled data or a preclassified set of data points. The goal of unsupervised learning is to find regularities in the input data which is the only information that the algorithm receives. One important method of unsupervised learning is clustering where the aim is to find clusters of groupings of input. [20]

Clustering is useful in several exploratory pattern analysis, grouping, decision making and machine learning situations. Clustering is particularly appropriate for the exploration of interrelationships among the data points to make an assessment of their structure. In other words if we have no previous information about the form, classes or features of the used data then clustering is an efficient tool. [21]

The stages in a clustering algorithm are presented in Figure 6. Feature selection aims to find the most effective subset of the original features to use in clustering. The next step is to find similarities in the pattern for example by measuring pattern proximity. Finally the grouping can be either hard where each point is categorized to a group, or fuzzy where each pattern has a variable degree of membership in each of the output clusters. A feedback loop ensures that when new patterns are discovered, the corresponding data points are grouped correctly. [21]

All clustering algorithms will produce clusters regardless of whether the data contain clusters or not. Some algorithms may obtain better clusters than others if the data does contain clusters. On the other hand data which do not contain clusters should not be processed by a clustering algorithm; it's no use trying to find patterns or clusters that don't exist in the first place. Thus it should be considered carefully whether clustering is applicable for the current application or not. [21]

Unsupervised learning algorithms will not be discussed further as the nature of the data used in this thesis rules these methods out of the scope of this thesis. The data used is already labeled, so disregarding the labels and using clustering would in fact mean losing information that we already have about the characteristics of the data. Clustering can be used for finding theme specific words in certain documents but the data used in this thesis has error messages that rarely hold thematic meaning and therefore clustering is not applicable.

2.5 Text classification

Text classification with machine learning has been studied for some time now and for example the paper from Sebastiani [22] discusses the history and the current state of the subject. Sebastiani also compares different approaches to document classification. In the 80's text classification was based mostly on logical rules which wouldn't be considered machine learning by today's standards. In the 90's the concepts of supervised learning and classifiers came to be used widely. [22]

Sebastiani defines text classification as approximating the unknown target function, that describes how documents ought to be classified, by means of a function called the classifier (model) using only the knowledge extracted from the documents [22]. The stages of a text classification process are presented in Figure 7. The text is first tokenized to individual words after which they are stemmed. Tokenization aims to separate the text document into individual words. Words can be separated for example with either a whitespace or with different separators such as a dot, a comma or a hyphen. According to Lovins [23] a stemming algorithm is a procedure which reduces all words with the same root to a common form, usually by stripping each word of its derivational and inflectional suffixes. Stemming is typically performed for natural language documents where words have suffixes and different tenses. When stemming is used as a method of feature extraction one key advantage is the reduction of dimensionality as several different words can be grouped under a single word stem. However it is important to note the issue of polysemy, or the issue of a word having multiple meanings. In this case it's impossible to deduce the meaning of the word after stemming. For example the word stem “comput-” could be the stem for either “computer” or “computation”. Stemming clearly has drawbacks as well and as with all dimensionality reduction methods some information is lost. For example Duwairi et al. [24] showed that too aggressive stemming can reduce classifier precision.

As mentioned above, stemming is particularly important for natural language documents. When performing text classification the nature of the data should be considered carefully because a computer generated text can have an entirely different structure than a natural language document. Natural language processing (NLP) is on it's own an interesting branch of text classification with documents containing different nuances and meanings. Sebastiani presents a term called word sense disambiguation (WSD) that is the activity of finding the sense of a particular word occurrence. For example in English the word “bank” can appear in “the Bank of England” as in a financial institution or in “the bank of a river” [22]. Typically with a computer generated text the objective is to form easily understandable text without any nuances. Therefore WSD and stemming are usually not necessary for computer generated documents. NLP won't be discussed further in this thesis because the text data used is computer generated.

The next step after stemming proposed by Ikonomakis et al. in Figure 7 is the

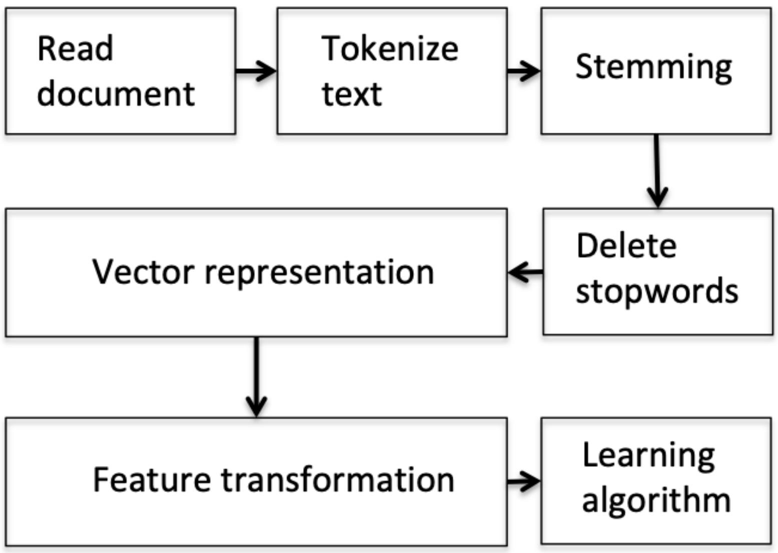


Figure 7: Typical stages in a text classification process.

removal of stop words. Stop words are non-informative words that appear frequently in documents but carry no usable information for the classification. In English common stop words include articles, prepositions and conjunctions such as the, a, an, and, or and so on. These words have no distinguishing potential between categories so they should be filtered before the vector representation to avoid unnecessary noise. [25]

The filtering of stop words from text data has been studied for example by Silva and Ribeiro [25] and Schofield et al. [26]. Silva and Ribeiro found that stop word removal removes information that could mislead the learning machine. They also concluded that among low frequency word removal, stop word removal and stemming, the most significant preprocessing method was the stop word removal. [25]

Schofield et al. studied three different scenarios with stop words: models with stop words intact, models with stop words removed before training and models with stop words removed after training. They found that the model improves with the removal of stop words and that removing stop words after training is generally just as effective as removing them before. Schofield et al. also questioned the use of custom stop word lists as it can compromise the validity of the model. If the analyst is too aggressive in removing words, the resulting models may be biased towards the analyst's personal views on what is important in the collection of documents or corpus. The study found that generating a corpus-specific stop word list provides relatively little utility to training a model. [26]

After stop word removal the descriptive words of the text data should be transformed into a form that is understandable by a computer. According to Silva and Ribeiro "the most common, simple and successful document representation used so far is the vector space model, also known as the Bag of Words." [25]. This means that each document is represented with a vector that has one component for each term occurring in the whole collection i.e. all of the documents. Each component then has as the value the number of times the term occurred in the document. Thus each document in the collection is represented as a point in a vector space with one dimension for every term in the vocabulary. [25]

There are several possible ways to select the features and transform the vectors before inputting the data to the learning algorithm. The simplest is a binary one where the word either appears or does not appear in a document [25]. This representation is usually replaced with the term frequency or *tf*. Term frequency is simply the number of times a term occurs in a document. However it is also possible to add weights to certain words to both find the key words that are important to classification more efficiently and also to possibly reduce the dimensionality of a learning task.

According to Leopold and Kindermann [27] inverse document frequency or *idf* is the most commonly used importance weight for SVMs. This can be further improved with *tf* to get *tf-idf* that increases with the number of times that the term occurs in the document and decreases with the number of times the term occurs in the whole collection of documents altogether. Given a document collection D of N documents with an individual document d and a word w, we calculate

$$w_d = f_{w,d} \log \frac{N}{f_{w,D}}, \tag{13}$$

where $f_{w,d}$ equals the number of times w appears in d and $f_{w,D}$ equals the number of documents in which w appears in D [28]. One way to interpret *tf-idf* is that it weighs terms that occur frequently in a single document while discriminating words that appear in many or all of the documents in the collection. In other words it can be used to find key words that hold the most information about the current document i.e. the features that best separate this document from the rest of the collection. Meanwhile common words such as articles, pronouns and prepositions hold no relevant information on their own and because they occur frequently in all of the documents in the collection they receive a low score and are therefore not considered important for the classification task. One advantage of *tf-idf* then is that intensive stop word removal is not necessary as this score already discriminates words that occur frequently and hold no information.

The last step of the text classification process is to feed the feature vectors to the learning algorithm that then classifies the documents to corresponding classes. In essence the purpose of the previous steps is to find the best separating features for each document. If the documents are very similar then naturally it's much more difficult to classify them.

Nowadays SVMs are widely used in text classification. For example Joachims [15] argues that SVMs are suitable for text classification because they are able to handle the high dimensional input space associated with text classification. Colas and Brazdil [29] compared SVM to kNN and naive Bayes on binary text classification tasks and found that SVM is more appealing theoretically and in practice even though kNN and naive Bayes are faster to use.

3 Materials

The data set used in this thesis consists of the log files collected from RELEX Planning cloud customer environments. Log files are generated in the Planning cloud whenever a user runs an action in the environment. The log files contain different information about the action run and the current status of the environment. In this thesis we focus on the error messages in the log files. The log files are collated in a ticketing system called Kayako in order to compile various metadata regarding the error messages.

3.1 Logging in the RELEX Planning cloud

Log files created by the actions are mainly used internally in the company by the software developers for debugging purposes. Log files can also be used by project managers to find out information about action run failures. There are three levels of log rows: INFO, WARN and ERROR. INFO level log rows contain general information about the status of the run for example what files the system is accessing or at what point of the calculation the run is currently on. These rows mainly interest software developers. WARN level rows contain information about unexpected conditions during the run that may not outright lead to an error but might still interest the user. Typical warnings include unexpected parameter values and missing information. WARN level rows can sometimes be used to investigate run errors. ERROR level rows are the most critical and they are logged whenever an error occurs during the action run. There are many different reasons for ERROR level rows and these will be examined further later. Even if ERROR level rows have been logged the action run may have been completed successfully. Having an error message in the log file does not automatically mean that the action resulted in a catastrophic failure.

An example from the Planning cloud environment featuring a workflow with two actions can be seen in Figure 8. Workflows are a way to chain different actions and filters together and run them in succession. Workflows are run from top to bottom so starting from the top first we have a filter. Filters can have different conditions such as products with a certain sales price or locations with a certain number of products. Instances that fulfill the filter condition continue to the right branch and in this case their value “Product-location: Order parameter” is set to 2.00. Continuing on the workflow the next action is called “Calculate order proposals” which is applied to only the instances that are not filtered out. Other actions include for example parameter optimization, regression analysis, forecasting and replenishment and data management.

Running actions produces two logs, a Ruby log and a Java log. The Ruby log has a high level overview of what workflow is running and what actions it includes. An example of the first 10 rows of a ruby log can be seen in Appendix A. The Java log has much more low level detail about the tasks the action is handling. In addition to this if an error occurs an additional Java error log is created that has only the ERROR level rows from the Java log. An example of the first 10 rows of a java log

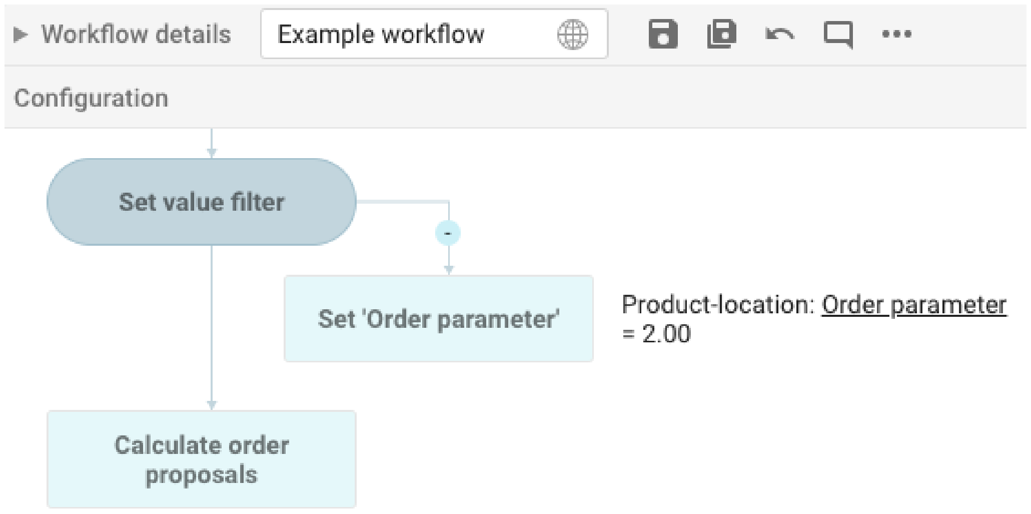


Figure 8: A workflow from the RELEX Planning cloud environment containing one filter and two actions.

generated from the same workflow that produced the abovementioned ruby log can be seen in Appendix B.

Generally speaking the logs have a similar layout regardless of the action that was run. Perhaps the most important feature regarding the contents of the log is the number of instances the action handles. If the workflow changes parameters or calculates order proposals for thousands or even hundreds of thousands of instances then naturally the length of log increases and the possibility for issues rises.

The most common cause for error tickets are missing files in the system. Customers are expected to upload sales data daily and update any old information regarding product-locations, campaigns etc. In case these files are missing the nightly run fails in an error and new calculations can't be made for the next day. Sometimes the customer may simply forget to upload the new files. Other reasons for failure include missing data in files for example missing columns or values, wrong format of the file, typos in the file or the file was sent too late and the scheduled nightly run already started. Other reasons for errors to occur, that are not related to files, are for example when an environment has gone unresponsive, the run was canceled manually, the environment runs out of memory or the environment is not properly configured. Different error types along with their explanations can be seen in Table 2. All error messages have similar structure in that they have a brief explanation of what happened and where the error came from. An example of an error message can be seen in Appendix C.

Most of the error tickets are created because of an error during the nightly run. Majority of the tickets each day are created before 9 a.m. Tickets created later in the day are often caused by smaller runs or by a nightly run being started late. The handling of each error depends heavily on the context in which the error occurred. Sometimes the severity of an individual error is affected by the previous errors and conditions in that environment and sometimes the error doesn't have any connection

Table 2: Example segments of common error messages with different error types and their explanations.

A segment of an error message	Explanation of error type
Error running scheduled job #1666447311725: wrong number of arguments (0 for 1)	A run was started with no arguments when there should have been one.
Transaction_Update_Open_Sales _Order_Allvouchersrelex2018-12-30.*.csv was found 0 times when searched for minimum of 1 occurrences.	The software expected to find a new uploaded file but that file was missing.
Data row contains invalid number of columns. 1 examples of data below.	The specified number of columns in the header of the file is different than the actual number of columns in the data.
Missing data in 1/162 locations.	The software tried to run calculations for several locations but the data was not sufficient for one of them.

to previous runs. Some error tickets could be considered critical only after a certain number of occurrences. Sometimes the same error could be considered critical for one customer but non-critical for another. Deciding what action to take for an individual ticket can be quite arbitrary and it is difficult to establish fixed rules as it is largely dependent on the support team's judgement and previous knowledge on the subject.

The individual's own judgement as well as the decision of the whole support team affects greatly how each error ticket is dealt with. As mentioned previously, it is difficult to establish general rules that apply to all the tickets and this poses a challenge for a machine learner. Typically error tickets concerning unresponsive environments or a similar critical error in a production environment always require additional investigation before the ticket can be closed. However these tickets are a minority and most often the need for a further investigation is a sum of multiple things to consider.

3.2 Kayako

The flowchart of incident detection and ticketing is presented in Figure 9. Customer environments produce several gigabytes of log files daily, depending on the size of the customer environment, and these logs are parsed first by checkers. The checkers filter out all irrelevant log rows so that only the rows containing error messages are further processed. These error row messages are then processed by Kayako [30] which creates new tickets based on the nature of the error.

Kayako is a help desk software for personal and connected customer service. RELEX uses the classic version of Kayako which is suited to ticketed workflows, IT support and service desk use cases. When an error occurs in a customer's Planning

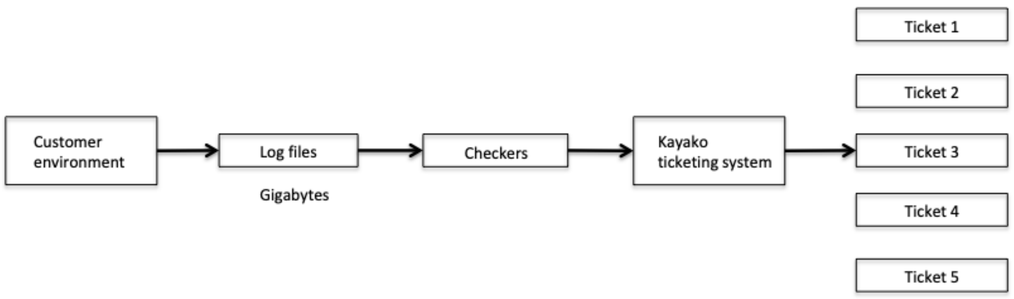


Figure 9: The process of creating tickets from logs produced in customer environments.

cloud environment a ticket is created in Kayako. The contents of the ticket include the actual error message itself and the ticket also has some metadata such as the date when the ticket was created and the priority of the ticket. The support team personnel can also mark down the time worked on each ticket signifying whether the ticket was closed immediately or if a longer investigation was required to solve the issue. This information is vital in this thesis as the predicted classes equal the concepts of “close the ticket immediately” and “investigate the ticket further before closing”.

Each ticket is assigned a priority level based on rules set in Kayako. These rules have been manually set by the support team personnel and in essence they search for certain keywords in the error message. For example one of these rules states that: If **“Body”** Regular expression **“/was found [0-9]+ times when searched for [0-9]+ to [0-9]+ occurrences/”** THEN Change ticket priority: B - High. So in other words if the error message contains that specific string then the priority is changed to B - High. The rules have a specified execution order which means that the priority of the ticket can change depending on the strings found in the error message.

3.3 Log data

The log data for this thesis was gathered daily from error tickets handled by the support team. Every error during a nightly run in any customer environment produces an error ticket. Typically each night around 80 or so tickets are generated. During the day the support team handles each ticket and marks down the time used to solve each ticket. Each ticket then contains information about the error message itself as well as the actual time needed to solve that ticket. Most often, around 90% of the time, the ticket is closed immediately needing no further investigation. In these cases the error that occurred was either not significant, a warning rather than an actual error, or something that is known to be a non-critical issue. The error tickets consist of several columns which are explained here.

1. Ticket ID

    - Each ticket has a unique ID that they are identifiable with. Ticket ID is a running integer with each new ticket receiving an ID that is the previous

ID increased by one.

2. Creation date
   - The date and time of when this ticket was created by the automatic ticketing system.
3. Resolved date
   - The date and time of when this ticket was resolved by a support team personnel.
4. Time worked
   - The amount of time the support team employee used to resolve this ticket. If the ticket is deemed non-critical or otherwise no actions need to be taken the ticket is closed immediately and time worked is marked '0s'. In case some sort of investigation needed to be conducted to discover for example the cause of the error, the severity of the error, error frequency or anything in that regard then the time worked is marked in intervals of 15 minutes with the minimum time being 15 minutes.
5. Priority
   - Priority of the ticket in a scale of:
     - A - Critical
     - B - High
     - C - Medium
     - D - Low

     The priority of each ticket is automatically assigned in Kayako based on word recognition. If the error message contains certain keywords the ticket is then set to a corresponding priority class based on existing rules.
6. Subject
   - The subject of the ticket that briefly explains the type of error that occurred and the customer environment where the error occurred.
7. Contents
   - The actual contents of the error message as generated by the software. The contents of the error messages vary greatly but are often long and can describe the run during which the error occurred, the expected outcome and the actual outcome of the run and the name of the file where the error stemmed from.

4 Methods

The machine learner is implemented using Spark's machine learning library MLlib. The primary machine learning API is the Dataframe-based spark.ml package so the data was transformed to and handled in the Dataframe-format. Each morning the error tickets of the previous day were analysed and collected into a csv-file. In order to create larger data sets, a collection of tickets over a longer period of time such as a week or a month were compiled into a single csv-file. After loading this file in the program it is transformed into a dataframe. Tickets are labeled according to the "Time worked" column following the rule: if the time worked is "0s" then assign class label 0, if the time worked is anything else then assign class label 1. In other words class 0 means tickets that are non-critical and can be closed without further investigation whereas class 1 means tickets that require at least some investigation before closing. Spark is usable in Python and that was the language chosen because of its simplicity.

4.1 Apache Spark

Apache Spark is an open source analytics engine for large-scale data processing. Spark is designed to handle big data and can be used to analyze very large files efficiently through parallelization. Spark has been used in a wide range of different applications. Most common applications for Spark are for batch processing large data sets such as log files. It has been reported that Spark is used in a company that ingests 1PB of data per day. Spark is also used for interactive queries such as SQL for relational queries. Stream processing and real-time decision making and analytics can also use Spark for example monitoring purposes. Spark is also used for many scientific applications. One group analyzed the brain activity in a larval zebrafish by visualizing the neurons. [31]

The key programming abstraction in Spark are the RDDs or Resilient Distributed Data sets. They are fault-tolerant collections of objects partitioned across a cluster. They can be operated in parallel which makes the cluster very efficient even with larger data sets. RDDs are evaluated lazily which means that the operations called on RDDs are not executed immediately. When the RDDs are evaluated the operations can be fused together to increase performance. The fault-tolerance stems from each RDD tracking each operation that was used to build it. These operations can then be rerun on the base data to reconstruct any lost partitions. [31]

A relatively new addition to the Spark framework is the Dataframe API. A Dataframe is conceptually equivalent to a relational database. It is a distributed collection of data like RDD but Dataframes have named columnar data. This way Spark has more information about the structure of the data. Data in this thesis is handled in Dataframe-format because the Dataframe API can perform relational operations on RDDs and it enables functional integration with the machine learning environment. [32]

Spark was chosen for this project because of its extensive features. It can be used for small data sets but scaling up is also easily available thanks to cluster computing.

Spark offers an easy-to-use programming interface with either Scala, Python or Java. Spark has also been successfully used previously to analyze log data as for example shown by Mavridis and Karatza [33] and Lin et al. [34].

Machine learning library or MLlib is Spark's integrated library offering algorithms and framework for machine learning applications. The library includes common machine learning tasks such as featurization, transformations, model training, model evaluation and optimization. MLlib consists of two packages: spark.mllib which is built on top of RDDs and spark.ml which is built on top of Dataframes. Because the data is handled as a Dataframe in this thesis spark.ml is widely utilised. [32]

MLlib offers several methods for feature extraction, transformation and selection. These methods are integral to this thesis as the raw text data needs to be transformed into feature vectors. MLlib's StopWordsRemover is used to remove the stop words from the error messages. The CountVectorizer class is used to count word occurrences in the error messages. By using MLlib's IDF class we can transform the features with tf-idf to form a new feature vector.

Finally MLlib also supports various different machine learning algorithms that can be used to train different models. There are algorithms for classification, regression, clustering, collaborative filtering and dimensionality reduction [32]. In this thesis SVM and naive Bayes algorithms are used so the classes LinearSVC and NaiveBayes are used correspondingly.

4.2 Support Vector Machines

As discussed earlier SVM has been used successfully in earlier text classification applications. SVM was chosen because of its ability to handle multidimensional data with relative ease and efficiency. Especially with text classification the dimensionality can become an issue as each word represents a dimension in the feature space. One could argue that using dimensionality reduction methods such as principal component analysis (PCA) would be beneficial in simplifying the classification. However as the number of dimensions is reduced so is the information gained from the original data. Using dimensionality reduction leads to losing information that could prove vital for the classification task. With SVM, dimensionality reduction is not necessary and we can utilise the full data set.

Spark's linear support vector classifier LinearSVC offers several parameters to tweak the performance of the model. Firstly because the result is obtained through an optimization algorithm the classifier has a parameter to limit the maximum number of iterations. Excessive iterations slow the calculation but too few reduce classifier accuracy. LinearSVC also has the option to change the constant C or the so-called regularization parameter which can be used to change the intrinsic margin of the SVM as explained earlier. Large values will steer the classifier to a smaller margin hyperplane and smaller values lead to larger margin hyperplanes. The final parameter of interest is the threshold parameter which moves the hyperplane towards either of the classes. Thus in essence this is a constant value that is added to Equation (10) to offset the hyperplane. This way if the training data set has been skewed towards one class we can normalize the model by moving the hyperplane to account for the smaller class.

4.3 Preprocessing

The purpose of preprocessing the error messages is to separate the raw text data into single words. The aim is to have descriptive and informative words that describe the error as accurately as possible. In Table 3 several examples of original error message data and the irrelevant information found in each of them are listed. As we can see the error messages can be quite messy so it's important to properly divide the text to individual words that are usable by the classifier. All the special characters need to be removed as well as numerical data. Dates, timestamps, ID numbers or numerical values can't be used in text classification as they would be interpreted as individual words instead of values having a certain context. In Table 3 we can also see an example of a filepath that should be separated into words with regard to the "/" -separator.

Stop words are also removed as they contain no significant information and they cause unnecessary noise in the data. Spark has a built-in class called StopWordsRemover that drops all the desired stop words from the input data. Spark uses a default list of stop words that can be found in [35]. The list contains for example pronouns, prepositions, conjunctions and other words that offer no meaningful information on their own.

Stemming is not deemed necessary in this case as the error messages are computer generated text instead of natural language. This is supported by the fact that stemming is a dimensionality reduction method that naturally leads to a loss of information. With computer generated text it's important to avoid excessive data loss as there aren't many filler words or issues with semantics when compared to natural language.

Two different methods are chosen for constructing the feature vector. The first one is a simple word count vector where each occurrence of a word is calculated and the resulting sums are placed in a vector accordingly. Using word count vectors requires efficient preprocessing and stop word filtering. Otherwise the resulting feature vector contains irrelevant word counts that may skew the classification. For example, if not removed, frequently occurring articles (the, a, an) in the input data will appear in the word count vector as having a significant importance in the text because of their high word count, but in reality offer no meaningful information in the context of the text as a whole.

The second method is tf-idf which weighs words based on their frequency in the collection of documents. Spark has a built-in IDF class that takes as an input a vector of word counts that is the tf or term frequency vector described above. It is important to note that the IDF model is trained only with the training data so that the words occurring in the training data serve as a baseline for tf-idf. The test set is then transformed based on the trained IDF model so that correct weights are assigned to words in the test data. In other words the IDF model has no preconception about the new data that is to be classified i.e. the test data.

Table 3: Examples of segments from the original error messages and the final word list after cleaning accordingly. Customer name has been redacted. Brackets are irrelevant and are removed along with dashes, colons, commas and slashes. Date and time information as well as other numerical values offer no information with regards to classification. Excessive whitespace and special characters are removed and the filepath is divided into words with regard to the slashes.

A segment of an original error message	Word list after cleaning
[lCheckPreconditionCommand] - Missing data in 1/162 locations. 5651: 2018-12-31 02:00:08,225/EET ERROR [lCheckPreconditionCommand] - Data was missing in locations: [1205]. Not processing these locations further in the chain.	lCheckPreconditionCommand, Missing, data, locations, EET, ERROR, lCheckPreconditionCommand, Data, missing, locations, processing, locations, chain
2018-12-30 21:08:56,623/EET ERROR [AbstractRunner] - Could not process any rows because the data for each of the rows was invalid. Rows with error: 4	EET, ERROR, AbstractRunner, Could, process, rows, data, rows, invalid, Rows, error
wrong number of arguments (0 for 1) /data/apps/processor-customer/releases/20181204135455-6.4.366_r0/WEB-INF/app/util/cube_export.rb:160:in ‘write’	wrong, number, arguments, data, apps, processor, customer, releases, WEB, INF, app, util, cube, export, write
Value exceeded adapter input threshold. Field=value, value=1.3200502400050111E14	Value, exceeded, adapter, input, threshold, Field, value, value

4.4 Performance comparison

In order to compare the classifier performance with different data, several different data sets were formed. The baseline is given by the full unabridged data set where no instances have been filtered out. Next we create a so-called “50:50-split” set for the aforementioned full data set by first picking all the class 1 instances and then taking a random sample of similar size of the class 0 instances. This way the 50:50-split sets contain approximately the same number of class 1 and class 0 tickets. The exact number of class 0 tickets is not actually always the same in these sets because of how the Spark’s sample method works but the difference is only marginal. This method is called undersampling where we take a subset of the overrepresented class to counter the class imbalance.

The priority of the tickets was also considered worth investigating so another subset was formed from the full data sets called the “priority split”. It was theorized that higher priority tickets would have better separating features that is tickets with

Table 4: The structure of the systematic testing. For each data set two different feature representations are chosen as well as three different values of C resulting in a total of six test setups for each data set.

Data set	Word count	C1
		C2
		C3
	tf-idf	C1
		C2
		C3

a high priority would be more important for the classification because their features would strongly point to either one of the classes. The priority split sets are formed so that a random sample of each priority level is picked from the full data set. From a full data set 50% of priority A, 25% of priority B, 15% of priority C and 10% of priority D are picked to form a priority split set accordingly.

The performance of the classifier is examined using accuracy, balanced accuracy, precision, recall and AUC. Spark has a built in method to calculate the AUC of a classifier and the rest of the metrics are calculated using the equations presented before. Although none of these metrics can on their own explain the performance or goodness of a single model, we can examine the efficiency of the classifier by considering all the metrics together. Theoretically the optimal model would have 1.0 as the value of all of these performance metrics but in reality this is difficult to achieve and performance is almost always a trade-off between metrics.

The effect of C or the regularization parameter of SVM was also investigated. Classifier performance was compared with values of 10, 1 and 0.1 as C. These values were obtained by trial and error by iterating different values of C and seeing how it affected the results. It was discovered that values outside of this range either did not change the results noticeably or that they deteriorated classifier performance. Values within this range produced the best performing classifiers.

Spark's implementation of SVM also supports manually moving the hyperplane towards either of the two classes. This can be done by giving the "threshold"-parameter either a negative or a positive value to move the hyperplane closer to either class correspondingly. When SVM was used on an imbalanced data set a threshold value of -1 was applied. It was discovered that values smaller than this often produced the opposite of the original problem and most if not all of the instances were misclassified as belonging to the underrepresented class. When using the 50:50-split sets the division was balanced and so the threshold value was set back to the default value of 0. In this case by moving the hyperplane slightly towards the underrepresented class we get comparable results to uniform data sets.

Each run is conducted so that from each data set 80% is chosen as the training data and the remaining 20% is the test data. The training and test data are sampled so that each run the data sets have different instances. Different combinations of the aforementioned test conditions are formed so that between each run the data set, the value of C or the feature representation is different compared to the previous run.

An example of the test setup can be seen in Table 4 and for each data set a total of six different test setups are constructed. This way we can cover all the possible test condition combinations and find the optimal one. After each run we calculate the performance metrics for the classification in question. Because the training and test data sets are sampled before each run there is some natural variation in the instances. Therefore each run is run five times and the values of the performance metrics are averaged in order to get more reliable results. These steps are repeated for all of the different data sets.

5 Results

Log data was gathered daily over the period of three months. In order to compare classification performance with different size data sets, both a one month data set and a full three month data set are utilised. The one month data set contains 2150 tickets and the full three month set contains 6385 tickets. The one month set contains 2015 class 0 tickets or tickets that are non-critical and have been closed immediately and only 135 class 1 tickets or tickets that required attention from the support personnel. Similarly for the bigger set there are 6083 class 0 tickets and 302 class 1 tickets. It can be clearly seen that the class division is not uniform and class 1 is greatly underrepresented.

In order to fix this class skew in the data sets we form new sets based on class division. As mentioned before we create the “50:50-split” set which helps balance the class division. We also create the “priority split” set to see if the ticket priority has an effect on the classification. An example of a priority split can be seen in Table 5. Note that again because of how Spark’s sample method works the amounts of different priorities in the priority split sets do not follow the aforementioned percentages precisely. The differences are marginal however as the values change with each new sampling when new priority split sets are created for each run.

5.1 Results with Support Vector Machines

The results using SVMs with the one month data set and its derivatives are presented in Table 6. The values shown for the different performance metrics are averaged over five runs. For each data set the effect of feature representation and regularization parameter is also considered. Similarly in Table 7 the results for the three month data set and its derivatives are presented.

The first thing we can see when looking at the results is that there is a lot of variation in the performance metrics over the different settings. For example in Table 6 recall can be as low as 0.023 or as high as 0.991 and similarly in Table 7 the range is from 0 to 1. Even within a certain data set the results can vary a lot. In Table 7 there are also two cells that don’t have a numerical value and that

Table 5: An example of a priority split set composition. The numbers correspond to tickets with a certain priority in the given data set that is either the full set or the new priority split set.

	One month	Three months	Prio split 1	Prio split 3
A - Critical	148	239	82	121
B - High	478	1653	113	415
C - Medium	1453	4335	218	663
D - Low	71	158	5	11
Total	2150	6385	418	1210

Table 6: Averaged results from five runs with the one month data sets and different parameters. The full one month data, 50:50-split of one month data and the priority split of the one month data were used. Feature representation was performed with a word count vector and tf-idf feature transformation. Different values of C were tested. The measured metrics are accuracy, balanced accuracy, precision, recall and AUC.

			Acc	Balacc	Precision	Recall	AUC
1 mo. full	Word count	$C = 10$	0.834	0.489	0.046	0.076	0.417
		$C = 1$	0.679	0.502	0.05	0.398	0.596
		$C = 0.1$	0.843	0.500	0.068	0.127	0.599
	tf-idf	$C = 10$	0.92	0.493	0.031	0.023	0.469
		$C = 1$	0.897	0.516	0.098	0.061	0.530
		$C = 0.1$	0.820	0.515	0.097	0.174	0.579
1 mo. 50:50	Word count	$C = 10$	0.479	0.721	0.558	0.846	0.7
		$C = 1$	0.469	0.629	0.451	0.951	0.589
		$C = 0.1$	0.578	0.583	0.578	0.795	0.665
	tf-idf	$C = 10$	0.546	0.557	0.543	0.991	0.586
		$C = 1$	0.549	0.593	0.580	0.766	0.597
		$C = 0.1$	0.582	0.608	0.567	0.852	0.603
1 mo. prio	Word count	$C = 10$	0.744	0.504	0.084	0.215	0.607
		$C = 1$	0.846	0.489	0.033	0.028	0.624
		$C = 0.1$	0.760	0.487	0.047	0.200	0.492
	tf-idf	$C = 10$	0.879	0.500	0.069	0.095	0.521
		$C = 1$	0.743	0.455	0.021	0.16	0.533
		$C = 0.1$	0.665	0.508	0.107	0.352	0.587

is because over the five runs we were unable to calculate balanced accuracy even once. Looking at Equation (2) we can see that if either of the two denominators are zero the equation cannot be calculated. In other words the model has predicted every instance of the data as either a class 0 or as a class 1 resulting in either of the denominators becoming zero. As for the zeros in the precision and recall columns of Table 7 following Equation (3) and Equation (4) the classifier did not correctly predict a single class 1 instance over the five runs. The single 1 in the recall column of the same table shows the opposite where the classifier managed to classify all class 1 instances correctly each run without misclassifying any but then again because accuracy and precision have the same value and we were unable to calculate balanced accuracy we know that the classifier actually just predicted all instances as class 1.

What's interesting in Table 7 is that all of these aforementioned anomalies occurred when $C = 10$. This suggests that with smaller values of C the model is able to predict classes more evenly without classifying every instance as only one of the either two. Predicting classes evenly is of course a much more probable and desirable outcome as the test data fed into the model almost never has instances only

Table 7: Averaged results from five runs with the three month data sets and different parameters. The layout is similar to Table 6.

			Acc	Balacc	Precision	Recall	AUC
3 mo. full	Word count	$C = 10$	0.935	0.533	0.113	0.017	0.430
		$C = 1$	0.917	0.520	0.081	0.069	0.538
		$C = 0.1$	0.907	0.508	0.066	0.038	0.550
	tf-idf	$C = 10$	0.941	0.475	0	0	0.456
		$C = 1$	0.883	0.520	0.087	0.108	0.585
		$C = 0.1$	0.93	0.588	0.218	0.025	0.531
3 mo. 50:50	Word count	$C = 10$	0.529	N/A	0.529	1	0.591
		$C = 1$	0.570	0.623	0.572	0.763	0.603
		$C = 0.1$	0.534	0.688	0.507	0.977	0.611
	tf-idf	$C = 10$	0.489	N/A	0.492	0.8	0.605
		$C = 1$	0.525	0.536	0.524	0.915	0.567
		$C = 0.1$	0.557	0.610	0.544	0.933	0.614
3 mo. prio	Word count	$C = 10$	0.88	0.489	0.036	0.048	0.447
		$C = 1$	0.862	0.486	0.031	0.058	0.504
		$C = 0.1$	0.909	0.53	0.113	0.092	0.571
	tf-idf	$C = 10$	0.91	0.474	0	0	0.499
		$C = 1$	0.912	0.525	0.104	0.077	0.610
		$C = 0.1$	0.883	0.497	0.047	0.07	0.484

from one class. Other than that it doesn't look like C has much of an effect on the performance metrics if any. Because the results presented here are an average over five runs the effect of C is not as prominent but when using a large value of C the effect is seen in individual runs when misclassifications are not tolerated well. This led to some odd behaviour as some runs only had one of the classes predicted for all the test data instances. As described above this happened a couple of times for all of the five runs. This issue was alleviated with smaller values of C as the margin of the SVM grows and some misclassifications are allowed.

One very noticeable feature in the results is the prominently different results with the 50:50-split set compared to full data set and the priority split set in both tables. Whereas precision and recall barely reach even 0.1 with full data and priority split set, with 50:50-split set we can get upwards of 0.5 precision and for recall we can reach 0.9 frequently. This means that with these data sets the model is quite efficient at correctly predicting class 1 instances however the precision value tells us that there are also quite many class 0 instances being misclassified as being in class 1.

Another feature that differentiates the 50:50-split sets from the other two data sets are the accuracy and balanced accuracy values. With full data and priority split the accuracy is easily around 0.9 whereas the balanced accuracy is closer to 0.5. As it was mentioned before, accuracy is not a reliable performance metric with imbalanced data sets. It is easy to reach a high accuracy by just predicting the class that is

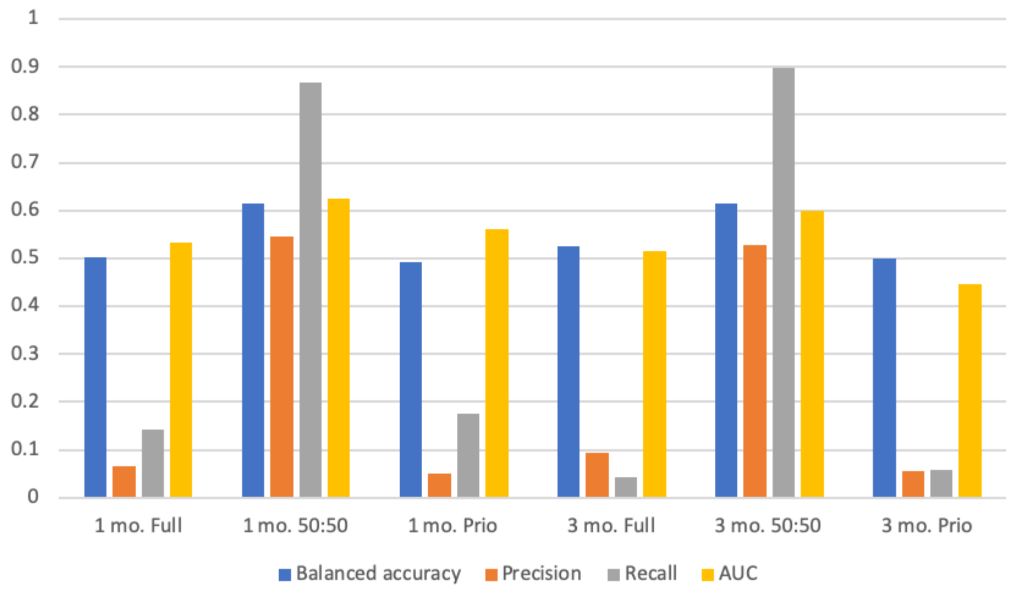


Figure 10: Average values of performance metrics for all of the different data sets from Tables 6 and 7. Averaging the values means that we are disregarding the choice of feature representation and the value of C and compare only the different data sets. Accuracy is omitted because of its unreliable nature in this case. The two 50:50-split sets perform drastically better on recall and precision and slightly better on balanced accuracy and AUC than the other data sets.

overrepresented in the data. Looking at the balanced accuracy values we can see that these models are not very accurate in reality. With the 50:50-split sets however accuracy values are around 0.5 but balanced accuracy values are in the upwards of 0.6. We can see that even though the accuracy is not much better compared to the other two it is still a slight improvement.

It's also worth noting that the choice of feature representation did not seem to affect the results much if at all. Whether the model used a simple word count vector or a feature vector with features transformed using tf-idf the performance metrics seem relatively unaffected. Because of this we can say that the one factor that is contributing most to the differences between the results is the data set composition. In Figure 10 are the average values of performance metrics for all of the different data sets. Accuracy is omitted because of its unreliable nature in this case. We can see that the two 50:50-split sets perform drastically better on recall and precision and slightly better on balanced accuracy and AUC than the other data sets.

It is clear that the models built using the balanced 50:50-split sets performed better than the models built from the other data sets. Clearly the class imbalance of the data set is an important factor playing a key role in the classification problem. By looking at the values of AUC for the 50:50-split sets we can see that these models respond better to new and unforeseen data. Because they have been trained on a balanced data set they have a better inherent capability of distinguishing the different classes.

5.2 Comparison with a naive Bayes classifier

To get a comparison for the results presented above a naive Bayes classifier was also used to classify the data. As discovered before, the 50:50-split sets performed best for the classification task with SVMs. Because of this only the 50:50-split sets were chosen to be used with a naive Bayes classifier. The test setup was otherwise similar so that two different feature representation methods were utilised and different performance metrics were gathered. The results of the naive Bayes classifier are

Table 8: Averaged results with a naive Bayes classifier from five runs with 50:50-split sets. The choice of feature representation seems to affect the results only slightly but it is interesting to note that only AUC seems to improve when using tf-idf and otherwise word count vector gives slightly better results on other metrics. This would suggest that using tf-idf gives a better model overall but with word count vector the classification results may be better for some test cases.

		Acc	Balacc	Precision	Recall	AUC
1 mo. 50:50	Word count	0.563	0.575	0.577	0.759	0.603
	tf-idf	0.514	0.531	0.518	0.740	0.658
3 mo. 50:50	Word count	0.567	0.632	0.551	0.899	0.524
	tf-idf	0.536	0.533	0.536	0.819	0.573

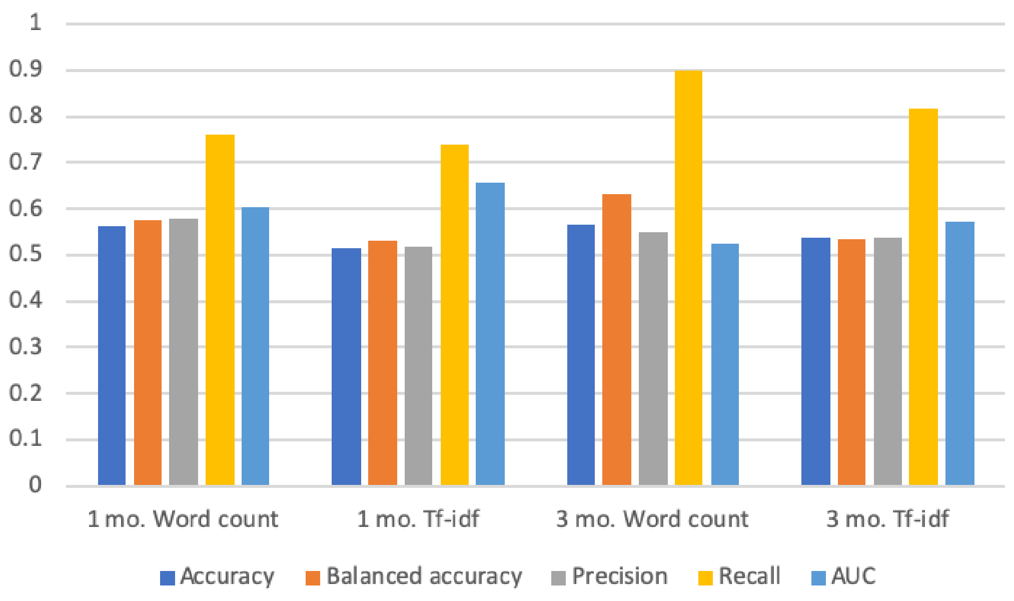


Figure 11: Plotted results of the naive Bayes classifier from Table 8. The choice of feature representation only affects the results slightly. Using word count vector gives only slightly improved results for all metrics except AUC. One noticeable feature is that recall is quite a bit better when using the larger three month data set compared to the one month data set and this could be explained by a bigger training data set.

presented in Table 8 and Figure 11.

When comparing the results of the naive Bayes classifier to the results from the SVM with the 50:50-split sets we can see that they are very similar. Balanced accuracy is around 0.6 and recall is able to reach 0.9. We can also verify that the choice of feature representation is not that significant as there is only a slight change in the performance metrics between different feature representations. Interestingly only AUC seems to improve when using tf-idf and otherwise word count vector gives slightly better results on other metrics. This would suggest that using tf-idf gives a better model overall but with word count vector the classification results may be better for some test cases. However after verifying the results of the SVM with a naive Bayes classifier it seems that the structure of the text data itself is the greatest contributor to the classification so that we do not get that different results with a different model or algorithm.

6 Discussion

The results show that the classifiers trained on the balanced 50:50-split sets performed far better than the classifiers trained on the imbalanced full and priority split sets. The results obtained with a SVM were further confirmed by obtaining similar results with a naive Bayes classifier. It is interesting to notice how similar the results actually were because a naive Bayes classifier works differently and is based on such a different principle than a SVM. Although the naive Bayes assumption assumes that the features in an instance are independent from each other this is clearly not the case with text classification where certain words frequently occur with others. The model performed well in this case most likely because the text data itself is not very natural and is instead computer generated where the text does not naturally flow from one topic to another. This would suggest that the features are indeed quite independent from each other, allowing the model to perform well.

The imbalance of the data sets proved to be a key element in the classification problem. When using the full data set the performance was surprisingly bad. Sometimes even guessing the class randomly would have resulted in a better outcome with this data. It was also surprising to see that the priority level of the ticket didn't have much of an effect on the classifier performance as the priority split set performed just as bad as the full data set. Even though the 50:50-split sets performed much better compared to the others there is one obvious drawback with them. Whereas the full three month data set contained 6083 tickets the 50:50-split set derived from this data set only has 604 tickets and the one month 50:50-split set has only 270 accordingly. Naturally with a smaller sample size there is much more uncertainty in the classification as there are less instances to train the model with. An even larger data set containing equal amounts of both classes could have resulted in a more reliable classification.

A very interesting point to note in the results is that there is a lot of variation in the values even within the same data set. For example with one month 50:50-split set the AUC varies between 0.586 to 0.7 and the balanced accuracy goes from 0.557 all the way up to 0.721. It is natural for some variation to occur but seeing how these values were an average over five runs the variation seems excessive. With individual runs the variations were even greater. These variations also seem to disregard the value of C or the feature representation method chosen and they show no obvious point of origin. The remaining logical conclusion is then that the nature of the instances in the data itself is such that there aren't many separating features between them. If an instance has features that are similar to both classes then it could be classified into either class depending on the run. In other words it is hard to distinguish between the classes so the label of a test instance is difficult to determine. This is quite unexpected because each ticket is different and the error message changes depending on the error that occurred. Admittedly the messages often follow a similar pattern where the message contains an environment name, some filepath and a description of the error. Maybe the descriptions of the errors were not descriptive enough to separate different cases enough.

Concerning the nature of the original classification problem, we would want to catch as many of the critical issues as possible. In the sense of software reliability and stability it is much more severe if we accidentally misclassify critical issues as non-critical rather than consider some non-critical issues to be critical issues. Therefore misclassifying some non-critical issues as critical is not that big of a drawback if it means that we correctly classify more actual critical issues. In the sense of performance evaluation this translates to having a model with a high recall value. In this case recall describes how many critical issues were correctly classified. Furthermore having a model with low precision means that a large number of non-critical issues were misclassified as critical. The predictions of the classifier would still need to be verified by the support team and therefore having many misclassified non-critical issues just means more work for the support team. Therefore in order to ensure reliability we want a model with high recall and in order to ensure minimal workload we want a model with high precision.

Looking at the results we can see that we were able to achieve very high recall values of around 0.9 with the models trained on the 50:50-split sets. However even with these models the precision is around 0.5. It means that although nearly all of the critical issues were correctly classified, about half of the issues classified as critical are actually non-critical. It is difficult to say if this kind of a classifier would be useful to the support team but at least it could cut down the number of non-critical issues to investigate by some margin. According to the results the model with the best recall was trained on the three month 50:50-split set and used the word count vector and regularization parameter $C = 10$. In this case however the model predicted every instance of the data as critical as explained in the previous section. This model would therefore be useless as it does not cut down the workload of the support team at all. With the second best model trained on one month 50:50-split set using tf-idf and $C = 10$ a recall of 0.991 was achieved. This model could actually be useful although with a low precision of 0.543 there would still be plenty of non-critical issues misclassified as critical.

We can see that the balance between recall and precision is a question about reliability and excess workload. By using the built in threshold parameter of Spark's SVM we can move the hyperplane towards either of the classes. This can be used to achieve a higher recall value with the model. Especially in this case where we don't want any critical issues to be misclassified, it could be interesting to optimize the parameter to find the best balance between recall and precision.

Three different values of C were used with the SVM in this thesis. The three values were selected by trial and error and the models seemed to perform the best within that range. However it could be worthwile to further optimize this parameter for each model. One option would be to use cross validation and find a model with for example the highest AUC using different values of C. Then the test set would be applied to this optimized model.

During the preprocessing stage numerical data was filtered out from the error messages. Numerals were found for example as dates and times, parameter values and as a part of some filepaths. This kind of data is challenging because for example with parameter values there always needs to be some context where the number appears, on its own the number doesn't have any meaning. Date and time values could be useful for example if some errors are known to occur periodically during some part of the day or week. In this case though there were no such occurrences. Including numerical data as features of the text data could be hugely beneficial to the classification if they are handled correctly by the classifier. For example the parameter values should be connected somehow to meaningful terms so that the value is interpretable and not just considered a single feature on its own. Including numerical values as is could probably harm the classification as the classifier then has more features to consider and most likely they wouldn't be separating features between the classes.

6.1 Challenges and how to improve

Even though in the best case scenario with the 50:50-split sets a balanced accuracy of 0.6 was reached with both SVMs and a naive Bayes classifier, this is not a very high value. It would mean that the model misclassifies 40% of the instances in the data set. This cannot be considered reliable because many critical error tickets would be missed or lots of non-critical tickets would erroneously be considered critical in this case. There are multiple issues making the classification challenging with the current setup leading to poor classifier performance.

First of all the error messages are considerably messy. A lot of cleaning and preprocessing is required before a feature vector can be formed. In addition to the stop words that are filtered out, different punctuation marks are removed from the text including for example periods, commas, colons, slashes, hyphens and brackets. If not removed, words such as "[ERROR]" and "ERROR:" would be interpreted as different words and as different features correspondingly. One problem that arises with cleaning the data is deciding what is irrelevant to the classification. It can be very much a personal opinion on what needs to be removed from the text data and thus a person's own interpretation of the nature of the data can affect the final classification result. Because there is no clear consensus on what is enough preprocessing and how to best perform the feature extraction for each application it can be iterated almost indefinitely. Cleaning and preprocessing the data is an interesting problem where there needs to be a balance between conserving enough relevant information and discarding irrelevant data.

As was already mentioned before the error messages often follow a similar pattern. Most messages contain a short description of the error, an environment where the error occurred and a filepath that may have caused the error. This can be problematic if different issues have the same word patterns for example. If the description is not clear or descriptive enough it may not be enough to differentiate issues from each other.

When looking at the error messages the criticality and priority of a ticket sometimes seems completely random. Similar error messages could be considered critical and needing investigation one day, but non-critical and to be closed immediately the other. Sometimes the priority level does not seem to have much of an effect on the criticality of the ticket. It also seems there is some interaction between the tickets where if a certain type of error occurred on one day then the error ticket on the next day is critical but otherwise it's not. Understandably this makes it challenging for the classifier to distinguish between the two classes.

Finally the fact that the data was not uniform in the sense of class balance proved to be highly problematic. It would probably make sense to gather data over even a longer period of time and then form a larger 50:50-split set. The problem rises when undersampling the overrepresented class as we are basically disregarding a part of the data resulting in a smaller data set. However a new problem arises if data is gathered over a longer period of time. The software is updated periodically and these updates change how the calculations in the Planning cloud are run and therefore also change the structure of the logs. If data is gathered over a long period of time the error messages for certain issues can change. Some error messages can change greatly between version updates while others not so much.

To ensure that the structure of the logs doesn't change between version updates there would need to be improvements in the way the logs are generated. Improving log generation overall can also give better features for the classification. To improve classification results the messages should first be cleaned somehow so that they are more straightforward and they should probably contain more descriptive words. Though it is difficult to assimilate what is enough to distinguish between different tickets and how much description each message should contain. On one hand short and concise messages would not overly complicate the feature space but the message should still hold enough information. If the error messages themselves stay the same, one way to change the classification performance would be to improve the text cleaning and preprocessing. However as mentioned earlier this is quite troublesome because it is challenging to choose the best words or features from the data set that should be chosen for the feature extraction.

In this thesis the 50:50-split sets were formed by undersampling the overrepresented class or by choosing a sample of instances from the majority class to get a new data set where both classes were equally represented. However there is also a method called oversampling which could be used to improve the results presented here. For example Chawla et al. [36] present a method called SMOTE (Synthetic minority oversampling technique) where new instances of the minority class are randomly generated. This method uses kNN to examine the features of the neighbours of a randomly selected instance and then based on the features of these neighbours, creates a new synthetic instance. This way the new instances can be used to create a balanced data set without disregarding any of the data in the overrepresented class. Using oversampling instead of undersampling would allow the classifier to learn from a larger training data. It could also be interesting to use other methods to produce simulated training data to get a larger training set.

Different algorithms could also be considered. SVM is currently the most widely used text classification tool but maybe a different method would outperform it. Concerning the monitoring and the support process in general at RELEX, perhaps Kayako could do more than just search for certain keywords in the messages. Maybe the checkers before Kayako could do some sort of preclassification. Although it is difficult to say what this preclassification should be based on and how it would be implemented.

7 Summary

This thesis investigated the possibility of using machine learning to automatically detect incidents from log files in RELEX customer environments. Currently a designated support team in RELEX goes manually through the error messages found in the log files daily. This is understandably quite laborious as each error message requires at least some form of investigation as to what caused the error. Some error messages can be considered non-critical and further investigation is not needed but some error messages require thorough investigation and fixes to the software.

Having a machine learner classify the error messages to non-critical and critical issues could save a lot of resources and allow the support team to focus their work on other pressing matters. The classifier should be trained so that misclassifications are avoided. If multiple non-critical issues are classified as critical this actually causes excessive workload to the support team. Misclassified issues can compromise software usability or cause unnecessary investigations. The fundamental objective of implementing a machine learning classifier is to save time and resources. Therefore the usability of such a system depends on the classification performance.

First the error messages were preprocessed so that the text data is separated into individual words. Different punctuation marks and stop words were filtered out. This word data is used to build two different feature vectors using word frequency and tf-idf. Data was gathered over the period of three months and a full three month data set was used in addition to a smaller one month data set. The data sets were further processed to form sets based on class division and error ticket priority. The machine learning classifier was then implemented using SVMs. The results from the best SVM classifier were also compared to a similar setup using a naive Bayes classifier.

It was discovered that the feature with the biggest contribution to classifier performance was the data set itself. Using highly imbalanced data sets the classifier performance was surprisingly bad with a balanced accuracy of around 0.5. In other words this simply means the classifier is randomly guessing which class an instance belongs to. On the other hand classifiers using data sets with equal amounts of both classes performed much better in terms of precision and recall and the balanced accuracy was able to reach 0.6. Even though the accuracy is not significantly better it is still an improvement compared to randomly guessing the classes.

The balanced data set was then also used with a naive Bayes classifier to see if it performed better than the SVM. The classification results of the naive Bayes classifier were very similar to the SVM in this case. It was also discovered that the choice of feature representation does not seem to be significant as it did not have much of an effect on the results. Based on these results we can answer the research questions presented in the beginning of this thesis.

Question 1 *Can we use machine learning to reliably classify error messages found in log files?*

Based on the results we obtained we can say that classifying error messages in full data sets with a class imbalance is challenging. There are different methods to compensate for the imbalance and they would be required to make the classification feasible. If the data sets are processed to balance the class division we can reach better results. Even in this case however the accuracy of the classifier is not high enough to consider the classifier reliable. An accuracy of 0.6 means that 40% of the error tickets are misclassified. If critical issues are classified as non-critical the usability of the software is compromised and the reliability of the system is not high.

Question 2 *What kind of a machine learning system has the best classification performance?*

It was found that the best results are obtained using a balanced class with equal amounts of instances from both classes. The two feature representation techniques chosen, word count vector and tf-idf, performed similarly with the classifiers and the choice of feature representation between these two did not seem to have much of an effect on the results. Similarly the choice of algorithm between a SVM and a naive Bayes classifier did not seem to have much of an effect on the results. The characteristics of the data set largely determined the classification results and to obtain the best results the data sets need to be processed before inputting them to the classifier.

Question 3 *Does the system produce measurable value, and if so, how much and how is it measured?*

The machine learner is naturally able to perform the classification of error messages received daily in a matter of seconds whereas for the support team this can take several hours. However with a poor classification accuracy the performance of the classifier is not reliable meaning the support team would have to go through the results of the machine learner either way. In this case the measurable value of time saved would not be much at all in reality no matter how quick the classification itself would be. We can assume that the accuracy of the support team checking the error messages manually is close to 100% so the accuracy of the machine learner should be much higher than 60% to actually benefit from the faster classification. Even when using models with a very high recall there would still be plenty of misclassified non-critical issues because of low precision, meaning more workload for the support team. Therefore in this case the system does not produce measurable value with regard to time saved.

The results showed that even when using a data set that was balanced with regard to the class division the classifier performance was quite poor. Using a different algorithm or a different feature representation did not seem to improve the results. Therefore we can conclude that the characteristics of the text data play an important role in the classification task.

The error messages in the log file are quite messy and contain a lot of unnecessary and uninformative filler words as well as excessive punctuation marks. The error messages can also have similar word patterns and structures even if the cause of the error was different which makes it difficult to distinguish between two entirely different issues. Some issues can also cause other issues to occur meaning that the same issue can be critical on one day but non-critical the other depending on the context. It is difficult to determine how the priority levels of the tickets affect issue criticality. To improve the classification results the contents of the error messages should be cleaned. Getting rid of the unnecessary clutter in the text data would better differentiate each issue as an individual instance.

7.1 Suggestions for future research

In order to improve the quality of the text data the contents of the error messages should be investigated. Having more descriptive words for each issue would differentiate error messages from one another more efficiently. Currently at RELEX there is work being done to improve the overall logging system of the Planning cloud. The objective is to make each error message important and the structure of the error messages will change to key-value pairs. These fields would then contain all the necessary metadata for each error message with the object of better differentiating separate issues. By applying machine learning to this kind of better structured data the classification results should also improve. It is also suggested to improve the importance of priorities where a higher priority would more likely point to a critical issue. Regarding the monitoring process at RELEX in general it could be beneficial if the checkers before Kayako could do more than just search the error message rows from the log files. Perhaps the checkers could be used to perform a preclassification based on issue type that might simplify the final classification.

Currently one of the biggest problems with the data set is the overwhelming amount of non-critical issues compared to critical issues necessitating undersampling the overrepresented class. One very interesting direction for possible development would be to try different oversampling methods for example SMOTE. Having a large enough training data set is essential for the classifier to perform reliably.

It would also be interesting to try different algorithms for text classification. Although when it comes to text classification SVMs are currently the most widely used method, perhaps new algorithms can bring new discoveries. For example using neural networks might prove fruitful. Further optimizing SVM in this application is also worth looking into. Optimizing the C and the threshold parameter may improve the results and bring new discoveries. Kernel methods should also be looked into because of the difficult nature of data in this application. Kernel trick is not currently supported in Spark MLlib but together with the parameter optimization there is an opportunity for improvement in this application in those areas.

There was a lot of numerical data in the error messages in addition to words and in this thesis the numerical data was filtered out in the preprocessing stage. It's suggested to further study the effect of numbers on the text classification problem. One interesting element is differentiating between dates and parameter values. The numbers would most likely need to be connected to some context because on their own they would probably offer no valuable information. Including numerical data in the classification problem would be quite challenging but it is an interesting aspect to consider nevertheless.

References

[1] M. Christopher, *Logistics & Supply Chain Management.* Pearson UK, 2016.

[2] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[3] T. M. Mitchell *et al.*, "Machine learning. 1997," *Burr Ridge, IL: McGraw Hill*, vol. 45, no. 37, pp. 870–877, 1997.

[4] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[5] M. Junker, R. Hoch, and A. Dengel, "On the evaluation of document analysis components by recall, precision, and accuracy," in *Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR'99 (Cat. No. PR00318)*, pp. 713–716, IEEE, 1999.

[6] J. Huang and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[7] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerging Artificial Intelligence Applications in Computer Engineering*, vol. 160, pp. 3–24, 2007.

[8] R. E. Bellman, *Adaptive Control Processes: A Guided Tour*, vol. 2045. Princeton university press, 2015.

[9] J. C. Schlimmer and D. Fisher, "A case study of incremental concept induction," in *AAAI*, vol. 86, pp. 496–501, 1986.

[10] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent Data Analysis*, vol. 6, no. 5, pp. 429–449, 2002.

[11] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine Learning*, vol. 29, no. 2–3, pp. 131–163, 1997.

[12] A. McCallum, K. Nigam, *et al.*, "A comparison of event models for naive Bayes text classification," in *AAAI-98 Workshop on Learning for Text Categorization*, vol. 752, pp. 41–48, Citeseer, 1998.

[13] V. Cherkassky and Y. Ma, "Practical selection of SVM parameters and noise estimation for SVM regression," *Neural Networks*, vol. 17, no. 1, pp. 113–126, 2004.

[14] G. Baudat and F. Anouar, "Feature vector selection and projection using kernels," *Neurocomputing*, vol. 55, no. 1-2, pp. 21–38, 2003.

[15] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *European Conference on Machine Learning*, pp. 137–142, Springer, 1998.

[16] H. Drucker, D. Wu, and V. N. Vapnik, "Support vector machines for spam categorization," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1048–1054, 1999.

[17] O. Chapelle, P. Haffner, and V. N. Vapnik, "Support vector machines for histogram-based image classification," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1055–1064, 1999.

[18] S. Hua and Z. Sun, "A novel method of protein secondary structure prediction with high segment overlap measure: support vector machine approach," *Journal of Molecular Biology*, vol. 308, no. 2, pp. 397–407, 2001.

[19] S. Dumais, J. Platt, D. Heckerman, and M. Sahami, "Inductive learning algorithms and representations for text categorization," 1998.

[20] E. Alpaydin, *Introduction to Machine Learning.* MIT press, 2009.

[21] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[22] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[23] J. B. Lovins, "Development of a stemming algorithm," *Mech. Translat. & Comp. Linguistics*, vol. 11, no. 1-2, pp. 22–31, 1968.

[24] R. Duwairi, M. Al-Refai, and N. Khasawneh, "Stemming versus light stemming as feature selection techniques for arabic text categorization," in *2007 Innovations in Information Technologies (IIT)*, pp. 446–450, IEEE, 2007.

[25] C. Silva and B. Ribeiro, "The importance of stop word removal on recall values in text categorization," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 3, pp. 1661–1666, IEEE, 2003.

[26] A. Schofield, M. Magnusson, and D. Mimno, "Pulling out the stops: Rethinking stopword removal for topic models," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Short Papers*, vol. 2, pp. 432–436, 2017.

[27] E. Leopold and J. Kindermann, "Text categorization with support vector machines. how to represent texts in input space?," *Machine Learning*, vol. 46, no. 1-3, pp. 423–444, 2002.

[28] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the First Instructional Conference on Machine Learning*, vol. 242, pp. 133–142, 2003.

[29] F. Colas and P. Brazdil, “Comparison of svm and some older classification algorithms in text classification tasks,” in *IFIP International Conference on Artificial Intelligence in Theory and Practice*, pp. 169–178, Springer, 2006.

[30] Kayako. `https://www.kayako.com/`, 2019. Accessed: 2019-03-12.

[31] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, *et al.*, “Apache spark: a unified engine for big data processing,” *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[32] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, “Big data analytics on apache spark,” *International Journal of Data Science and Analytics*, vol. 1, no. 3–4, pp. 145–164, 2016.

[33] I. Mavridis and H. Karatza, “Performance evaluation of cloud-based log file analysis with apache hadoop and apache spark,” *Journal of Systems and Software*, vol. 125, pp. 133–151, 2017.

[34] X. Lin, P. Wang, and B. Wu, “Log analysis in cloud computing environment with hadoop and spark,” in *2013 5th IEEE International Conference on Broadband Network & Multimedia Technology (IC-BNMT)*, pp. 273–276, IEEE, 2013.

[35] PostgreSQL CVS Repository, “English stopwords.” `https://anoncvs.postgresql.org/cvsweb.cgi/pgsql/src/backend/snowball/stopwords/english.stop`, 2010. Accessed: 2019-02-19.

[36] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: synthetic minority over-sampling technique,” *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

A Ruby log

```
# Logfile created on 2019-05-06 13:23:19 +0300 by logger.rb/v1.2.7
[2019-05-06 13:23:19 +0300] [INFO ] Using Planning Cube: true
[2019-05-06 13:23:19 +0300] [INFO ] Workflow run 12193: Starting:
   optimization_full_name: Test optimization / Test folder
   optimization_id: 13535, optimization_clone_id: 13535, user_id: 481,
   job_id: 150323855741, auto_accept: , 4 actions
[2019-05-06 13:23:19 +0300] [INFO ] Workflow initialized in 292 ms.
[2019-05-06 13:23:20 +0300] [INFO ] Default filter id updated for 4823679
   product-locations in 1152 ms.
[2019-05-06 13:23:20 +0300] [INFO ] Creating a sub manager of Filter ()...
[2019-05-06 13:23:20 +0300] [INFO ] Filter (Filter) started at 2019-05-06
   T13:23:20.968+03:00.
[2019-05-06 13:23:20 +0300] [INFO ] Workflow run 12193 action_id 27810
   action_name Filter (Filter) (filter): Running action
[2019-05-06 13:23:20 +0300] [INFO ] Workflow run (no run_id, job_id =
   150323855741) action_id 27810 (filter): Filtering matching rows
[2019-05-06 13:23:21 +0300] [INFO ] Workflow run (no run_id, job_id =
   150323855741) action_id 27810 (filter): Updating filter_id for
   matching rows
```

B Java log

```
2019-05-06 13:23:19,654 +0300 INFO [Snapshot           ] - Loaded
   snapshot /data/apps/processor-cliff/shared/processor-cliff_production.
   db/2019/2019-05-06/2019-05-06_10-23-19.643.1557138199643 in 0 ms.
2019-05-06 13:23:19,706 +0300 INFO [Trx                ] - Starting
   commit for trx on snapshot 1557138199643:
2019-05-06 13:23:19,707 +0300 INFO [Trx                ] - Pending data
   for fi.relex.processor2.fastorm.Trx@42338a14/1557138199643 [OPEN]
   takes 807 bytes
  Table optimizations: updates 39 bytes, inserts 1 rows / 70 bytes, new
     strings 0 bytes, deletes 0 rows / 0 bytes
  Table optimization_runs: updates 0 bytes, inserts 1 rows / 46 bytes, new
      strings 0 bytes, deletes 0 rows / 0 bytes
  Table search_criteria: updates 0 bytes, inserts 2 rows / 150 bytes, new
     strings 0 bytes, deletes 0 rows / 0 bytes
  Table optimization_actions: updates 156 bytes, inserts 6 rows / 346
     bytes, new strings 0 bytes, deletes 0 rows / 0 bytes
2019-05-06 13:23:19,775 +0300 INFO [Trx                ] - Finished
   commit in 69 ms -> 1557138199771
2019-05-06 13:23:19,776 +0300 INFO [Snapshot           ] - Loaded
   snapshot /data/apps/processor-cliff/shared/processor-cliff_production.
   db/2019/2019-05-06/2019-05-06_10-23-19.771.1557138199771 in 1 ms.
2019-05-06 13:23:19,776 +0300 INFO [Snapshot           ] - Loaded
   snapshot /data/apps/processor-cliff/shared/processor-cliff_production.
   db/2019/2019-05-06/2019-05-06_10-23-19.771.1557138199771 in 0 ms.
2019-05-06 13:23:19,786 +0300 INFO [Trx                ] -  Rebased trx
   fi.relex.processor2.fastorm.Trx@251b96d/1557138074808 [OPEN] in 0 ms
   => fi.relex.processor2.fastorm.Trx@19b6e647/1557138199771 [OPEN]
2019-05-06 13:23:22,165 +0300 INFO [Trx                ] - Starting
   commit for trx on snapshot 1557138199771:
2019-05-06 13:23:22,166 +0300 INFO [Trx                ] - Pending data
   for fi.relex.processor2.fastorm.Trx@56160bcd/1557138199771 [OPEN]
   takes 184 bytes
  Table statistics: updates 91 bytes, inserts 1 rows / 93 bytes, new
     strings 0 bytes, deletes 0 rows / 0 bytes
2019-05-06 13:23:22,189 +0300 INFO [Trx                ] - Finished
   commit in 24 ms -> 1557138202185
```

C Error example

```
2019-05-06 05:19:56,058 +0300 ERROR [ForecastCommand    ] - Error
   generating forecasts for product_locations: 55834588871#naive_ref_1/
   TUKKU
java.lang.ArrayIndexOutOfBoundsException: Index -65 out of bounds for
   length 2
       at fi.relex.collections.lists.ImmutableDoubleList$Builder.set(
          ImmutableDoubleList.java:378) ~[processor-core-6.9-master.22752.
          jar:9b3935c7f6d]
       at fi.relex.processor2.forecast.util.NaiveReferenceForecaster.
          overwriteForecastsWithNaiveForecast(NaiveReferenceForecaster.
          java:68) ~[processor-core-6.9-master.22752.jar:9b3935c7f6d]
       at fi.relex.processor2.forecast.Forecaster.
          lambda$applyNaiveReferenceForecast$3(Forecaster.java:335) ~[
          processor-core-6.9-master.22752.jar:9b3935c7f6d]
       at java.util.Optional.ifPresent(Optional.java:183) ~[?:?]
       at fi.relex.processor2.forecast.Forecaster.
          applyNaiveReferenceForecast(Forecaster.java:335) ~[processor-
          core-6.9-master.22752.jar:9b3935c7f6d]
       at fi.relex.processor2.forecast.Forecaster.forecast(Forecaster.java
          :120) ~[processor-core-6.9-master.22752.jar:9b3935c7f6d]
       at fi.relex.processor2.command.ForecastCommand$1.process(
          ForecastCommand.java:271) [processor-core-6.9-master.22752.jar
          :9b3935c7f6d]
       at fi.relex.processor2.fastorm.cube.CubeView$Processor.process(
          CubeView.java:1057) [processor-core-6.9-master.22752.jar:9
          b3935c7f6d]
       at fi.relex.processor2.fastorm.cube.CubeView$CubeViewProcessor.
          process(CubeView.java:948) [processor-core-6.9-master.22752.jar
          :9b3935c7f6d]
       at fi.relex.processor2.fastorm.cube.CubeView$Processor.process(
          CubeView.java:1052) [processor-core-6.9-master.22752.jar:9
          b3935c7f6d]
       at fi.relex.processor2.fastorm.cube.CubeView$2.run(CubeView.java
          :859) [processor-core-6.9-master.22752.jar:9b3935c7f6d]
       at fi.relex.processor2.fastorm.util.WorkerPool$TaskBatch$WorkerTask
          .compute(WorkerPool.java:594) [processor-core-6.9-master.22752.
          jar:9b3935c7f6d]
       at fi.relex.processor2.fastorm.util.WorkerPool$TaskBatch$WorkerTask
          .compute(WorkerPool.java:555) [processor-core-6.9-master.22752.
          jar:9b3935c7f6d]
       at java.util.concurrent.RecursiveTask.exec(RecursiveTask.java:94)
          [?:?]
       at java.util.concurrent.ForkJoinTask.doExec(ForkJoinTask.java:290)
          [?:?]
       at java.util.concurrent.ForkJoinPool$WorkQueue.topLevelExec(
          ForkJoinPool.java:1020) [?:?]
```

```
at java.util.concurrent.ForkJoinPool.scan(ForkJoinPool.java:1656)
    [?:?]
at java.util.concurrent.ForkJoinPool.runWorker(ForkJoinPool.java
    :1594) [?:?]
at java.util.concurrent.ForkJoinWorkerThread.run(
    ForkJoinWorkerThread.java:177) [?:?]
```