SMS Spam Detection in a Real-World Platform using Machine Learning

Cesar Adolfo Rodriguez Villanueva

Helsinki June 19, 2019

UNIVERSITY OF HELSINKI
Master's Programme in Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Studieprogram — Study Programme
Faculty of Science	Computer Science

Tekijä — Författare — Author
Cesar Adolfo Rodriguez Villanueva

Työn nimi — Arbetets titel — Title

SMS Spam Detection in a Real-World Platform using Machine Learning

Ohjaajat — Handledare — Supervisors
Jyrki Kivinen

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	June 19, 2019	56 pages + 60 appendices

Tiivistelmä — Referat — Abstract

Spam detection techniques have made our lives easier by unclogging our inboxes and keeping unsafe messages from being opened. With the automation of text messaging solutions and the increase in telecommunication companies and message providers, the volume of text messages has been on the rise. With this growth came along malicious traffic which users had little control over. In this thesis, we present an implementation of a spam detection system in a real-world text messaging platform. Using well-established machine learning algorithms, we make an in-depth analysis on the performance of the models using two different datasets: one publicly available (N=5,574) and the other gathered from actual traffic of the platform (N=1,477).

Making use of the empirical results, we outline the models and hyperparameters which can be used in the platform and in which scenarios they produce optimal performance. The results indicate that our dataset poses a great challenge at accurate classification, most likely due to the small sample size and unbalanced dataset, along with nuances in the dataset. Nevertheless, there were models that were found to have a good all-around performance and they can be trained and used in the platform.

Avainsanat — Nyckelord — Keywords
machine learning, classification

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Thesis for the Algorithms, Data Analytics and Machine Learning subprogramme

Contents

1 Introduction

Since the 1990s, the mobile industry started adopting the short messaging technology, opening the doors for people who were seeking to communicate beyond a phone call or voice message. While the popularity of SMS (Short Message Service) took time to gain traction, by the end of 2000, the average user would send 35 messages per month and during the year 2001 more than 15 billion messages were sent around the globe [Smi]. While research shows that its usage among private users has shifted towards online-based messaging applications such as Whatsapp, Facebook Messenger and WeChat [Cro13], automated messaging services provided by corporations are still in the rise and form a vast majority of the messaging traffic nowadays. Examples of these services include emergency alert broadcast, appointment reminders, phone number confirmation and others.

The SMS has allowed companies, clubs, associations, and advertisers to reach out to customers efficiently and in bulk, substituting regular mail, phone calls, and even email. It has become one of the largest branches of mobile marketing where up to 100 million marketing-related messages were sent out every month just in Europe. With the higher adoption of technology in our daily lives, application-to-person (A2P) messages started becoming popular as a means of mobile banking and identity verification. A market research report made in 2017 states that the global A2P market is valued at $62 billion and can rise to $86 billion by 2025 [TMR11].

Along with all the benefits experienced by businesses and individuals, malicious agents also found a new way to harm or take advantage of users through messages. Spam, a form of unsolicited message, and phishing, messages that attempt to steal personal information, started appearing and mobile companies were presented with the challenge of identifying these malicious messages. While the problem is found in different magnitude among countries, in parts of Asia, for example, up to 30% of the messaging traffic was identified to be spam in 2012 [GSM11]. This information pushes companies and motivates academia to research solutions that can mitigate this kind of detrimental traffic.

Text messaging today is not a permission-based service, so that leaves the user open for any kind of message directed at them. While there are device features that can block undesired messages, such as a blacklist of senders, these are not sufficient nor practical and are certainly limited to user action. Also, compared to email and instant messaging services, telecommunication companies do not obtain immediate

feedback on whether a message is spam or not. Email inboxes even offer the possibility to classify messages by categories such as purchases, finance, and updates, while text message inboxes often lack such functionalities. This makes it particularly hard to gather a user-verified collection of messages marked as spam or not and leaves companies and researchers to gather and label messages manually. Motivated to address this spam problem, industry and academia have been employing machine learning techniques to aid in the identification of spam messages, particularly techniques in the field of text classification [GC09, C+08].

While text classification has been of interest for many years before and gained importance with the arrival of email, text messaging opened a new and different challenge. Pioneers made use of existing email classification techniques and adapted them into text messaging spam detection. Even though SMS spam classification is closely related to email spam classification, there are further challenges such as the short length and frequent use of abbreviations in a text message.

1.1 Scope of the Study

This thesis covers the practical implementation of an SMS spam detection system in a real-world platform along with experiments made to aid in the selection of machine learning models. The objective is to find the most appropriate machine learning model for the platform and different scenarios, taking to account both accuracy and efficiency. Due to the nature of the customers using the platform, where a portion of the businesses handles time-sensitive traffic, the classification of messages must be fast enough to not affect the overall flow of the platform, while also minimizing erroneous classifications. It may also be the case that each business user of the platform has distinct requirements about spam handling, thus we have to able to provide the best model for each requirement.

Before the study, the platform did not have an existing dataset of spam and ham messages, thus we manually collect and label a new dataset. Using a publicly available dataset and our own, two sets of experiments are done, with the objective of empirically finding the best model for each dataset and comparing results. Finding the best model depends on which criteria we impose. For example, there may be cases where identifying all the spam is a priority while other cases may require no ham messages to be blocked. Taking this in mind, we will answer the question of which machine learning model can be used in the platform using different criteria.

We acknowledge the opportunity to explore unsupervised and semi-supervised models with data coming through the platform but that is left as a potential topic and it is not in the scope of this study.

1.2 Structure of the thesis

In Section 2, we present an overview of machine learning containing the types of problems, techniques used to solve them and performance metrics. We then go over text classification, and spam detection, reviewing previous studies and algorithms that were developed to solve these classification problems.

In Section 3, we describe the different machine learning models that were used in this thesis, including Naive Bayes, Support Vector Machine and Multi-layer Perceptron. We describe how they work, which advantages and disadvantages researchers have encountered when using them for classification problems, and how they have been used in the spam classification field, particularly in SMS spam detection.

Section 4 contains a description of the datasets we are using in this study, how they were obtained and how they are composed.

In Section 5, we introduce the platform *spice*, used in the company Veoo as its mobile messaging solution, and we describe the technologies involved. It also includes how the spam detection system was built and what were the decisions behind it so the high traffic that the platform handles was not affected.

In Section 6, we present the empirical part of this thesis, containing the comparison of how models work given the different parameters and datasets. Having gathered all the data necessary and comparing results, we can make a decision on which model is to work best on the platform.

Finally, Section 7 contains a brief summary of the findings and conclusions of this study.

2 Background

In this section, we provide background knowledge on machine learning, text classification, spam classification, and the more specific SMS spam classification problem. We review popular approaches used in this domain and the various challenges relevant to this thesis that researchers have faced before.

2.1 Machine Learning

Machine learning is a subfield of computer science, contained in artificial intelligence and related to statistical analysis. Its goal is to extract knowledge from a given dataset by identifying patterns, rules, groups, etc. and use that knowledge to improve the performance of a given task. Machine learning methods are usually applied to problems where an exact algorithm is too difficult or complicated to implement. Take for example the task of determining whether an image contains a cat or a dog. It is possible to build such a system, but it requires special knowledge on image analysis and the system will only be able to handle the particular images it designed to identify. In the case that we want to add more categories, more code needs to be written driving up the complexity of the system. A point will be reached where it may not be trivial for a human to understand how the system works or where the number of categories is so high, it would be impossible for a human to program them all.

This is where machine learning comes in and offers a variety of algorithms, which do not require domain-specific work to extract meaningful knowledge about a dataset. Algorithms of this kind adjust themselves by optimizing a scoring function, thus we say that the machine "learns" about the input data and is able to make predictions about it. They are generic to any dataset in the sense that the algorithm itself does not change depending on the problem, only how the data is presented to it changes.

Machine learning applications include image classification, sentiment analysis, spam detection, recommendation systems, and many others. Problems, where gaining insight or discovering patterns would be hard for a human to do, are most likely using machine learning techniques in one way or another.

2.1.1 Categories

Traditional machine learning algorithms are divided into three categories: supervised, unsupervised and semi-supervised methods. These depend on the availability of labels for the target dataset.

Supervised machine learning algorithms build a mathematical model from a dataset that contains both inputs and expected outputs. In this category, we may be interested in predicting a label (or labels) for every unseen instance, or to calculate a continuous numerical value. We refer to the first case as a classification problem while the second case is known as regression. Regressive learning can be thought of as learning a function and classification is learning to distinguish sets. Finding the topic of a document is an example of a classification problem while predicting the exact time delay of a flight is a regression problem.

Unsupervised algorithms have the task of grouping or clustering a dataset without having the desired outputs, thus relying on finding commonalities between instances. A common unsupervised machine method is clustering which tries to group the dataset into different groups where the instances have commonalities between each other. For example, given a set of customers and their purchase history, attempt to group them and find common patterns in their purchases to predict future ones. This grouping is done without explicit labels, but then it can serve as a parting point into labeling the dataset for it to be used later with supervised machine learning algorithms.

There is a third category of methods which is a combination of supervised and unsupervised techniques called semi-supervised machine learning. It deals with datasets that contain a mix of labeled and unlabeled instances, making use of the labeled data to improve the learning process of the unlabeled instances. When the cost of labeling a dataset is infeasible, and the acquisition of unlabeled data is relatively inexpensive, semi-supervised methods are of great value requiring only a small amount of labeled data.

2.1.2 Training and Evaluation

The process of finding the best machine learning model for a particular problem or dataset involves a process of training, validation, and evaluation. There is no silver bullet algorithm so a method that models a dataset accurately may perform badly when faced with data from another domain. This motivates researchers to make use

of standardized metrics and evaluation techniques to compare algorithms with one another.

There are a few things to consider when training and evaluating a method: reducing variance (overfitting) and bias (underfitting), choosing the best parameters for the learning algorithm and using different training, validation and testing datasets.

Overfitting is the action of modeling a learning method too closely to the dataset such that the accuracy of future predictions is affected. This is due to the fact that data usually contains noise which should be ignored otherwise it is taken as part of the model structure. Underfitting is the result of a learning method not accurately capturing the structure of a dataset. This occurs usually when some parameters containing information of the model are ignored, causing poor predictive performance.

Most machine learning models can be tuned through a set of parameters and their performance may rely on the correct selection of parameter values according to the dataset that is being modeled. Parameters may influence the overfitting/underfitting of a model (like the number of features) thus comparing the possible parameter values is always a good practice.

When training and evaluating a model, it is usually good practice to divide the dataset into training, validation and testing sets. The training set is used to model the machine learning algorithm, the validation set is used to verify the performance, and the testing set is only used at the end to provide a final metric score. When training a model, we are usually interested in tuning its hyperparameters to obtain better classification results. Training a model and validating it with the same set is a common mistake, as the model would usually overfit. That is the reason the validation set is kept separate to find the hyperparameter values of the model which yield the best performance. The testing set is only used when we found the best hyperparameters and we want a "fresh" dataset to test our model and provide a final score. By keeping the sets independent, we avoid leaking details of the validation data into the model and obtain an objective view of the prediction performance.

Another item to consider is that using only using one specific split of training and validation may introduce overfitting to the model, therefore different splits of training and validation should be used to make comparisons across models and their hyperparameter variations. Cross-validation techniques perform partitions of the dataset into several subsets and use different ones as training and validation data at each iteration. The results obtained at the end are combined to give an estimate

of the model's predictive performance. There are several types of cross-validation methods such as *leave-p-out*, *leave-one-out*, and *k-fold*. *Leave-p-out* uses p observations as the validation set and remaining as the training set, while *leave-one-out* is the special case when $p = 1$. At every iteration, the validation and training set are switched, and prediction results are gathered. *K-fold* cross-validation partitions the dataset into k equal parts, and uses 1 as the validation set and the remaining $k-1$ as the training set, with each iteration changing them until all parts were used as validation sets. There are some cases where the dataset may be unbalanced, containing more instances of one label than another, and splitting it may cause some labels to be absent. In *stratified k-fold*, the folds are created to preserve the proportion of classes of the original dataset.

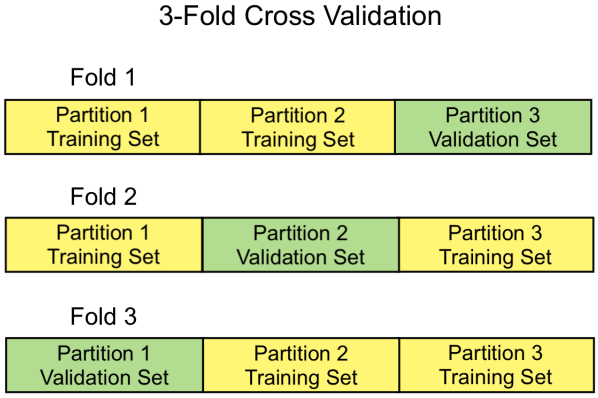


Figure 1: A visual example of 3-fold cross-validation. The splits remain the same, but every iteration a different one is chosen as the validation set.

2.1.3 Data Representation and Feature Selection

Machine learning is a data-driven field, and the use of raw data has an important impact on the effectiveness of a model. Data sanitation and representation have an effect on the final results, sometimes even more than the particular machine learning method that is chosen. Data should be represented in such a way that commonalities and structures can be easily identified. This task may seem trivial but humans are sometimes not equipped with the tools to do it by manual examination, thus

we develop techniques on to obtain empirical data which can give insight on how representations and features affect the performance of an algorithm. The field of *feature engineering* seeks to provide methods to automate the process of extracting the most relevant features from a given dataset. The benefits include facilitating data visualization and data understanding and reducing storage requirements, training and prediction times [GE03]. For big datasets, this can mean a difference of orders of magnitude when storing information and training a model.

An approach to selecting the most representative features is to find features that are highly correlated, meaning they share information, and only keep one to retain the information structure [Hal99]. Another popular method is principal component analysis (PCA) used as a dimensionality reduction analysis to find patterns in data of high dimensions which would otherwise be impossible to find manually. Lastly, there are techniques that take observed variables and combine them into other variables, called latent variables, helping to reduce the dimensionality of data.

2.1.4 Performance Metrics

In order to accurately compare models and make a proper decision, standardized metrics must be used. The metrics also depend on what kind of problem we are solving, classification or regression. Due to the nature of continuous variables in regression problems, its metrics are focused on formulas comparing the difference amongst expected and obtained values such as the sum of squared errors, and usually output numerical values instead of percentages. On the other hand, classification performance metrics have a discrete nature and they consider errors versus successes obtained in the predictions. This study is on a binary classification problem (spam detection) where prediction results are either true or false, and we focus on metrics for these kinds of problems.

There is no single classifier that works on all given problems, and selecting the best classifier may depend on what kind of performance is to be maximized. Popular metrics used to evaluate the quality of a classifier include accuracy, precision, and recall, and they represent ratios between expected and predicted results. In order to define these metrics, let us introduce several terms (in the context of binary classification) that describe classification results. Let P and N be the number of real positive and negative cases, respectively. We define TP as the number of true positives and TN as the number of true negatives the classifier returned; FP and FN are the number of false positives and negatives found. In our case, the positive label

represents spam messages while the negative represents ham. The most commonly used metric is accuracy, which outputs the percentage of correct predictions using the formula

$$Accuracy = \frac{TP + TN}{P + N}.$$

Two other important metrics are recall and precision which formulas are

$$Recall = \frac{TP}{P}$$

and

$$Precision = \frac{TP}{TP + FP}.$$

Recall represents how many positive instances the model was able to classify, while precision gives the percentage of positive predictions that were correct. Specificity and Negative Predictive Rate (NPR) are the negative counterparts of recall and precision, represented by the formulas

$$Specificity = \frac{TN}{N}$$

and

$$Negative\ Predictive\ Rate = \frac{TN}{TN + FN}.$$

In the academic field, recall and precision are the terms often used and people are familiarized with the concept, so for clarity purposes, we will refer to specificity as ham recall and NPR as ham recall. There are datasets or problem domains where the data is not balanced (such as the one in this study) meaning there are more instances of a certain class than the other. Blindly maximizing accuracy in these cases may be misleading. To illustrate the problem, let us consider a dataset composed of 90% *false* labels. If we run a dummy classifier that labels every instance as *false*, we can claim to have an accuracy of 90% and specificity of 100%. Nevertheless, looking at the precision score, it is equal to 0% (or NaN, given we have 0 *true* predictions). An alternative metric called balanced accuracy [UM15] addresses this problem by calculating the arithmetic average of recall and specificity with the formula

$$balanced\ accuracy = \frac{recall + specificity}{2}.$$

Taking the results of our dummy classifier above, the balanced accuracy is 50%, showing it is doing poorly on identifying one of the labels. Another metric which

helps us with unbalanced datasets is the F1-score which takes the harmonic average of precision and recall with the formula

$$F1\text{-}score = 2 \cdot \frac{precision \cdot recall}{precision + recall}.$$

F1-score balances between precision and recall, since a classifier may be good at retrieving a certain class (high recall), but it does so by overcompensating and making mistakes in the labeling (low precision). F1-score gives equal importance to how many instances are correctly classified as well as how precise the classification is. The Receiver Operating Characteristic (ROC) curve is also used to evaluate the tradeoffs between true positive and false positive rates of the classifiers. The ROC curve is a plot of the recall against the false positive rate or fall-out calculated with

$$FPR = 1 - specificity.$$

Using this plot, we can see how the increase of the true positive rate relates to the false positive rate, assuring the model is not labeling everything as *true* to reach a better performance. Calculating the area under the curve (AUROC), we can obtain a score from 0 to 1 which summarizes the performance of a model in the ROC, with 1 being a perfect classification.

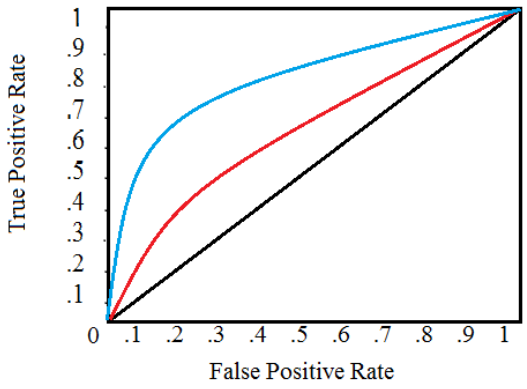


Figure 2: A ROC curve of two methods, and the diagonal marking chance performance (0.5). The AUROC score of the blue line is higher than the red line, thus making it a better classification method.

2.2 Text Classification

Since the World Wide Web was invented and once its first user-friendly interfaces appeared, information started to be generated and stored in massive amounts. With the addition of digital libraries, corporate databases, and email, we find ourselves surrounded by an enormous bulk of text-based documents for which analysis and/or classification can provide very useful information to its owners and users. Text classification is a field of machine learning that seeks to classify documents or pieces of text into categories or clusters by a similarity score. It is relevant to several problem domains such as spam filtering, language identification, sentiment analysis, and others. Until the late 1980s, the community in text classification was mainly looking at *knowledge engineering* for possible solutions; this approach required knowledge experts in the particular corpora to design a set of rules in a program that will be used as parameters of classification [SBF98]. The design and implementation of these systems were often done by pairings of domain experts and researchers, who would work in conjunction to digitize knowledge and incorporate it into a business or research platform. Knowledge engineering was useful in solving a particular problem domain, and its systems were constrained to the rules they were programmed with, and thus could only be improved with human intervention and knowledge. After a decade, and due to the clear limitations of this approach, the community began focusing on machine learning methods to develop algorithms capable of classifying text using a set of pre-classified documents to learn characteristics which can then be used for future classifications. These methods no longer relied on domain-specific, man-made rules, thus they could be used by other problem sets without structural changes. Algorithms, such as the Naive Bayes, SVM, and Neural Networks, are examples of these methods where no expert knowledge is required. Some of them are capable of performing as good or better than knowledge-engineered tools, even with small or complicated training sets. Take for example Koller et al. [KS97], where they implemented a hierarchical multi-topic document classifier that used as little as 10 features to achieve close to 85% accuracy. Other examples include Yu et al. [YX08], where achieved accuracy of 95% using Naive Bayes and SVMs when detecting email spam from a corpus of 6000; or in Boiy et al. [BM09], where they succeeded to identify positive, negative and neutral sentiment in a multilingual set of documents with 83% accuracy.

With the community steering in the direction of machine learning, subfields have emerged within text classification. For example, feature engineering deals with the

fact that documents can be represented not only as a *bag-of-words* (word counts), but also in alternative ways such as phrase extraction, hypernym extraction (word relationship) and word n-grams [SM99]. Phrase-based representation attempts to use the information that groups of words have by taking relevant slices containing two or more words and converting them into features. Hypernym-based representation takes advantage of the semantic information that words contain using hypernymy (a linguistic term for the "is a" relationship), thus it can extract more relevant meta information about the meaning of the text. An N-gram is an N-character (or N-word) slice of a longer string, for example, "foo" is a 3-gram of "food". N-gram-based representation groups words in their respective n-grams and derives the feature counting on these groups rather than exact matches. By representing documents in a different manner, these approaches seek to identify traits in the instances that could yield more meaningful information to the learning model, improving accuracy and overall performance by reducing the number of features. This advantage comes with the cost of a longer feature extraction step as opposed to calculating term frequencies in the bag-of-words representation.

Another important and closely related field to text classification is data mining. It is the process of discovering patterns in datasets without any previous knowledge, where the patterns provide further insight into the dataset, such as anomalies or dependencies [HMS06]. The results can even be used as a supplement to machine learning methods by identifying classes or groups not previously detected [FP96]. Data mining is considered a form of unsupervised machine learning with applications such as categorization, DNA sequence mining, learning engineering and traffic analysis.

We now proceed to define the problem of text classification. Given a set of training documents $D = X_1, ..., X_n$ such that each document is labeled with a class value from a given set of classes $C = 1, ..., k$, and given an unknown document W, predict the label or class for W [AZ12]. The training documents are used to identify common features within the classes, making it possible to extrapolate these commonalities to future predictions. The classification problem assumes discrete values for the labels; in the case of continuous labels, it is called regression modeling. In problems such as multilabel classification, the documents can belong to multiple classes, whereas in binary classification, the documents can only belong to one of two classes. Spam classification is a binary classification problem with several applications such as:

- **News Organization**. A news service may be interested in the automatic

labeling of its articles so the users can easily filter for specific topics. Given the volume of articles produced every day, machine learning models can be trained to label them, saving massive amounts of manual effort.

- **Sentiment Analysis**. In any given review website, there is always a metric which rates a given service or product, but beyond that, companies cannot tell what specific features of the service or product the users like or dislike. By doing sentiment analysis, these highlights can be extracted and a recommender can be trained to take into account features that users may like based on their past reviews.

- **Email Routing**. Most recent email inboxes have implemented different folders or categories where they store incoming messages. Categories such as Trips, Purchases, Updates, and Finance help the user navigate and can be seen in Google Inbox, which has a built-in categorization of emails.

- **Spam Detection**. Undesired and potentially harmful messages are sent daily through several channels, email, text messages and instant messages are some examples. Apart from causing cluttered inboxes, these messages are frequently looking to scam the user of money or personal information, so it is on both the end-user and the company interest to detect and avoid such messages.

2.2.1 Spam Classification

A few ways of identifying spam include whitelists, blacklists, content-based, and any combination of the three. Whitelists and blacklists approach the problem by relying on identifying benign or malign senders to allow/stop incoming messages. This approach may prove effective to some extent, but once senders change their phone number or sender code, these solutions start to struggle. In this paper we approach the problem of spam classification from a content-based view, that is, using information in the text message itself to determine if it is desired or undesired. It is important to note that whitelists and blacklists can be built using content-based solutions and vice-versa.

In the field of machine learning, text classification may deal with multi-class problems, i.e. sets of documents that can be classified into different categories that may or may not be mutually exclusive; spam classification only cares about two classes: spam or ham. Spam can be encountered in places such as blogs, email, social networks, and forums. It can contain harmless (but unsolicited) information such as

marketing campaigns, but it can also form a part of a phishing or malware attack, potentially harming the recipient. One of the first studies was done by Sahami et al. who focused on using the Naive Bayes model as their classifier of email spam [SDHH98]. Making use of domain-specific properties, such as the sender information and the appearance of a manually curated list of words, they achieved close to perfect precision using a corpus of size 1789.

2.2.2 SMS Spam Classification

One of the first studies on SMS spam detection was presented by Almeida et al. [AHY11]. They introduce an SMS spam collection gathered from different sources including Grumbletext Website [Una], the NUS SMS Corpus [Kan11], Caroline Tag's PhD Thesis [Tag09] and SMS Spam Corpus v.0.1 Big [HS11]. This SMS collection is used in our study. It is composed of 4827 legitimate messages and 747 spam messages. Apart from doing an analysis on the frequency and informational gain (IG) of the tokens in the dataset, they performed a duplicate analysis using plagiarism detection techniques to avoid adding similar (yet not identical) messages to their collection. This is a useful technique to be considered when building your own SMS collection. Using this newly constructed dataset, they compared two tokenizers and machine learning models to see which combination would yield the best results. The models included Naive Bayes (and several of its variants), SVMs, MDL, kNN, C4.5, and PART.

Shirani-Mehr [SM13] made use of the SMS Spam Collection v.1 [AHY11] to compare several machine learning models and their results. While no word stemming or misspelling fixes are done with the message instances, spaces, commas, dots or any other special characters are removed during the tokenization step. Also, the most and least frequent tokens found are removed from the dictionary, since they provide no additional information about a message. This is a common practice given that very frequent words can be found in the majority of the messages and rare words may cause the model to overfit based on them. Three additional features, number of dollar signs ($), number of numeric strings and overall message length were also included. Given all these features used, he found that message length has the highest mutual information (MI) of all, meaning it is an especially useful feature to use while classifying. Naive Bayes, SVM, k-NN, Random Forests, and Adaboost were all compared to find the best performing model. Naive Bayes was found to have the best accuracy and recall, even when increasing the complexity of the other

algorithms.

Given that SMS spam detection uses text classification algorithms, it is a very similar problem to email spam detection. The main difference is the fixed length of SMSs. Narayan and Saxena make use of existing email classification techniques in SMS spam detection to test their effectiveness [NS13]. They recognize various challenges stemming from the 160 character limit, including the use of acronyms and abbreviations, special treatment of bigrams and trigrams and overfitting of the model due to a limited corpus. The first experiment presented made use of a Bayesian email classifier with different tokenization methods (naive word split, email-like lemmatizer, n-grams). Using n-grams had the largest accuracy increase with respect to naive word split, but by only 7%. This shows that having complicated tokenizers such lemmatizers can hurt the performance of the classifier. It was also shown that the accuracy of this classifier was not comparable when using it for actual email. They then proceed to describe an improved approach at classifying short text messages using multi-level stacked classifiers. The process is as follows: the first-level Bayesian classifier records a subset of words whose individual probability is higher than a threshold, then this subset becomes the input of another model. If the probability does not meet the threshold, the instance is labeled as ham. Their results show that an SVM on two-level Bayesian classifiers yield the best accuracy of 98.8%, compared to single Bayesian (92.1%) and single SVM (93.9%).

Making use of the well known bag-of-words approach, which represents a text message by the occurrence of its words against a dictionary, Warade et al. study the results of doing feature selection by two methods: chi-square (CHI2) and information gain (IG) [WTS13]. CHI2 measures the independence between two variables, in this case, a feature and a label. The aim is to find features that are highly dependent on a label and discard those that are independent. IG measures the impact in the entropy of removing or adding a feature to the feature set in the performance of a classifier. Besides feature selection, input sanitization is performed to overcome intentionally misspelled words used by spammers such as “m0ney” for “money” or “off3r” for “offer”. This is called poison removal, and it is done by dividing the words into substrings of two characters and comparing them to a similar dictionary to find matches. Poison removal increases the overall accuracy of classification. Making use of Naive Bayes classifier, the two feature selection methods are compared. They discovered that the best result for both techniques is obtained when using the top 20 scoring features, with close to 90%. This is a suboptimal result considering the amount of preprocessing that was done.

Most of the previous work on SMS spam detection has been using content-based features, meaning using only information gathered from the text itself. Qian Xu et al.
take a different approach and make use of non-content features such as sender and recipient as well as the network information and metadata of the message [XXY+12]. They divide such features into three categories: static, temporal and network-related. Static features include the total number of messages sent and the total size of the messages. Temporal features are the average sending time, number of messages during a day and number of recipients in a day. Lastly, network-related features include the total number of recipients and the clustering coefficient describing how a sender is connected to its recipients. An SVM is chosen as the learning model after an initial comparison with kNN. They perform experiments using only one category of features and all three at the same time. The results show that using all three categories results in almost the same improvement as only using temporal and network features alone. Using PCA, they extract the most important features with number one being the total size of messages, the next eleven being temporal features, tailed by a network feature.

While extensive work has been done using supervised machine learning methods for SMS spam detection, they all have the weakness that a sufficiently large and reliable training corpus is required. The unsupervised or semi-supervised counterparts have not had great attention, but Gianella et al. employ character n-gram counts to develop a semi-supervised algorithm, capable of reaching high accuracy scores with a small training set. This technique was based on an email spam detection procedure developed by Uemura et al. [UIKA11] which assumes ham messages tend to contain more complex content than the spam messages. Using an extension of the model in Resnik et al. [RH10] as their probabilistic generative model, they estimate its parameters from unlabeled and labeled data. They found that their unsupervised and semi-supervised models outperformed other supervised models such as Semiboost and SVM when the training set was small and performed comparably when the training set was larger. Further on, active and transfer learning can also address the issue of the limited number of training sets available. For example, transfer learning attempts to reduce the size of the required training set by employing trained models from a different domain, such as email spam, to improve the results of a model only using SMS corpus alone.

3 Classification Models

In this section, we review the machine learning methods used in this study: Naive Bayes, Support Vector Machines and Neural Networks. We present an overview of each algorithm and any previous relevant research on spam detection.

3.1 Naive Bayes

Naive Bayes is a type of machine learning algorithm that uses Bayes' theorem and assumes conditional independence between the features. While this assumption is false in most cases, the model can still perform difficult classification tasks very well and sometimes better than more complex models. Bayes' theorem estimates the probability of an event based on prior knowledge of conditions in which said event occurred [Rou18]. Naive Bayes has been a popular machine learning algorithm for many years due to its simplicity and acceptable performance and a lot of research has been done to address its weaknesses. While it can provide classification probabilities, these are not very accurate, and people often use it as a classifier where the class with the highest probability is chosen. The application of the Naive Bayes classifier to spam filtering was initially proposed by Sahami et al. [SDHH98] who used it as a decision framework. It was proven to be an effective model without much manual tuning with a precision of 97% and a recall of 94% using only words as features. They also studied how accurate the model was when also adding hand-picked features such as the presence of "spammy" words like FREE, (!!!), or $$$, reaching close to perfect scores with their testing set. With its lack of complexity, Naive Bayes has been proven to be an effective model in text classification and it has become one of the preferred methods used in spam classification.

Rennie et al. [RSTK03] list several systemic problems in the Naive Bayes classifier. One problem is that the model performs poorly with an unbalanced training set (occurrences of one class are higher than the other). Another problem is that the conditional independence assumption ignores completely the evidence of a relation between two or more words. They also argue that multinomial Naive Bayes does not model text well and propose using inverse document frequency (idf) which gives greater weight to words with rare occurrences. They provide experimental results that show that their three improvements to Naive Bayes result in better performance of the algorithm.

There are several types of Naive Bayes classifiers, including Gaussian, multinomial

and Bernoulli. Given the nature of text classification, the type used in this study is Multinomial Naive Bayes. McCallum et al. [MN+98] found that Multinomial Naive Bayes performed better than other models at text classification even with a small training set, and one could reduce its error significantly by increasing its vocabulary size, although this increase is not linear.

3.1.1 Multinomial Naive Bayes

Multinomial Naive Bayes (MNB) is the defacto Naive Bayes algorithm used in text classification since it is the most suitable to represent a document. In the multinomial model, a document is represented as a vector of counts of events, where each count is the number of times a word from the gathered vocabulary appears in the document. This is what is known as the bag-of-words or term frequency representation. The naive assumption is that the appearance of a word is independent of the other words in the same document. Making use of the prior parameters of each class, we can estimate the posterior probability given the evidence (our bag-of-words representation), by multiplying the probability of each word in the document and selecting the class with the highest probability (ham or spam). One of the main critiques of the Multinomial Naive Bayes is that it completely ignores the order that words occur in a document, although this is part of the independence of features assumption.

Let us describe MNB in more detail by analyzing the formulas and rules it is based on. Let C be a set of classes, let N be the size of our corpus or vocabulary. MNB assigns a test document d to the class c that has the highest *posterior* probability $P(c|d)$ defined by Bayes' theorem as

$$P(c|d) = \frac{P(c)P(d|c)}{P(d)},$$

where $P(c)$ is the class *prior* calculated by dividing the number of documents with class c by the total number of documents C. $P(d|c)$ is called the *likelihood* and it is the probability that we see document d in class c. Using the chain rule to calculate the likelihood, we expand it as

$$P(d|c) = P(t_1|t_2, ..., t_n, c)P(t_2|t_3, ..., t_n, c)...P(t_{n-1}|t_n, c)P(c),$$

where t_i represents the term i in the document t. But since we assume that each

term t_i is conditionally independent of every other term, then we can rewrite it as

$$P(t_1|t_2, ..., t_n, c) = P(t_1|c).$$

Thus, the likelihood formula is reduced to

$$P(d|c) = P(c) \prod_{i=1}^{n} P(t_i|c),$$

where $P(t_i|c)$ is the probability that term t_i appears given class c. It is estimated with

$$P(t_i|c) = \frac{a + \sum_{d \in D_c} n_{t_i d}}{b + \sum_{t'} \sum_{d \in D_c} n_{t' d}},$$

where D_c is all the documents of class c, t' is all the terms and the denominator returns the count of all words in all documents of class c. a and b are the correcting terms which are used so $P(t_i|c)$ does not equal to zero in the case that t_i does not exist in the corpus. This is important otherwise our likelihood formula above, $P(d|c)$, will automatically be zero if one of the terms is not in the corpus. When we choose $a = 1$ and $b = N$, it is called Laplace smoothing, which assigns an initial count of 1 to every term, and normalizes their probabilities by dividing by our dictionary size, N. The formula can be viewed as a division of counts, the numerator being the number of times t_i appears in documents of class c, divided by the total number of words in documents of class c. It is the proportion that a word takes in the subdictionary of class c.

As mentioned before, Naive Bayes and its variants are generally not used to calculate the exact posterior probability but only to obtain the class with the highest probability; this lets us ignore the $P(t)$ in the denominator and only calculate the likelihoods for all classes and take the highest value. We can express this classifier that assigns a class c to $\hat{y}$ with the formula

$$\hat{y} = \arg\max_{c \in C} P(c) \prod_{i=1}^{n} P(t_i|c).$$

Kibriya et al. [KFPH04] claim that all the Multinomial Naive Bayes modifications presented in Rennie et al. [RSTK03] are not necessary and that by only transforming the word counts using *term frequency inverse document frequency* (TF-IDF)

yields improved results without much complexity and extra calculations. TF-IDF comes from the multiplication of *term frequency*(TF) and *inverse document frequency* (IDF), hence its name. Its formula is

$$\text{TF-IDF}(word, D) = \log(f + 1) \log \frac{D}{df},$$

where f represents the frequency of $word$, D is the total number of documents in the corpus and df the number of documents $word$ appears in. It is a numerical statistic used in information retrieval to reflect how relevant a word is to a document in a collection of documents. Its value increases with the number of appearances of the word in the document and is reduced by the frequency of the word in the corpus. If instead of term frequencies, TF-IDF is used in MNB, the likelihood formula for a term t_i given class c would change to be

$$P(t_i|c) = \frac{\sum_{d \in D_c} \text{TF-IDF}(t_i, d)}{\sum_{t'} \sum_{d \in D_c} \text{TF-IDF}(t', d)}.$$

A term t_i which appears several times solely in class c documents will return a higher score since it is taken as more representative of class c. In Section 6, we perform our own experiments of MNB versus a TF-IDF MNB.

3.2 Support Vector Machines

Support Vector Machines (SVMs) are a group of supervised machine learning algorithms commonly used for classification problems but also used for regression. They were first introduced as a learning algorithm for binary classification problems [CV95] and then extended to multiclass classification and regression [G+98]. Their core idea is to separate instances into two classes by finding a hyperplane which maximizes the distance from each group of instances. They were conceived for linearly separable binary problems and extended to nonlinearly separable problems with the use of the kernel trick. Kernel functions allow algorithms to operate in high-dimensionality domains by being able to compute a similarity score between two feature vectors. They are highly accurate and work well with small training sets. Their main weakness is their training time can be slow (even if the number of features is low), making it non-ideal for larger datasets classification. They do not provide probabilistic estimates although these are not needed for the classification problem in this study. Since SVMs attempt to split the classes in the given

dimensional space, they are hardly affected by unbalanced training sets.

In one of the first experimental studies of SVMs, Chapelle et al. [CHV99] used a dataset with 14 categories each with 100 images for a total of 1400 images, representing them by mapping their pixels to a histogram which can easily be used by an SVM. After experimenting with various kernel types and parameters, their results show that error rates as low as 11% could be achieved. While this may seem a high error rate, it should be noted that there were 14 categories with not many instances, confirming the claim that SVMs perform well with small datasets.

Within the field of text classification, Kwok experiments with SVMs using news stories classified around 135 financial topics [Kwo98]. As a preprocessing step, he applied inflectional stemming, stop word removal, conversion to lowercase and elimination of words which appeared in less than 3 documents. He coded the resulting documents using the TF-IDF representation. The performance was measured by recall and precision and compared to kNN. While the SVM results were favorable, the difference was not outstanding, but the advantage that SVMs have over the kNN algorithm is their ability to take in new training instances fast and without taking much memory.

In a more related study, Drucker et al. [DWV99] approach the spam classification problem with SVMs. Using different feature representations like term frequency and binary and comparing to other classification algorithms like boost and decision trees, they find low error rates of 5% and better performance times. A particularly interesting find was that converting words to lowercase yielded more accurate results, which makes sense since this normalizes the data across the set.

3.2.1 Kernels

SVMs need a similarity measure that is used to compare instances to each other and find the separating hyperplane. A *kernel* is a function that takes two vectors and returns a measure on how similar they are to each other. The most basic kernel used is the dot product or *linear kernel* calculated by the dot product where the vectors represent two instances and their features. The dot product is sensible to the magnitude of the vectors, thus it is important to normalize the input vectors, otherwise feature vectors with higher magnitude will produce a higher dot product than their lower magnitude counterparts.

Hsu et al. [HCL+03] suggest that a linear kernel function should be used when the

instance set and number of features are large for better computational performance. In our use case, the number of instances is high due to the traffic and our training set, and speed is of high importance, so we can sacrifice some accuracy for a faster SVM.

There are problems where the data cannot be correctly separated using a linear kernel since it is nonlinearly separable. Take for example Figure 3 where the left dataset can be linearly separated and the right side requires a more complex function.

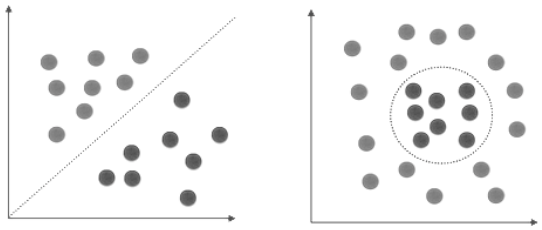


Figure 3: Two different datasets, linearly and nonlinearly separable respectively.

Given that the dataset is to be split using a hyperplane, a nonlinearly separable problem can be remapped onto a different dimensionality space using a different kernel. Expanding on the linear kernel, we also have the *polynomial kernel* calculated as

$$k(x, w) = (x \cdot w + b)^d,$$

where b is the intercept coefficient and d is the degree of the kernel. To better illustrate how polynomial kernels overcome the obstacle of nonlinearly separable datasets, let us look at Figure 4 where we have a one-dimensional dataset which can be easily separated. Then, in Figure 5, we see another dataset with the same dimensionality but it cannot be separated. Lastly, using a 2-degree polynomial kernel, we see in Figure 6 that now a hyperplane can be obtained to correctly separate this dataset.

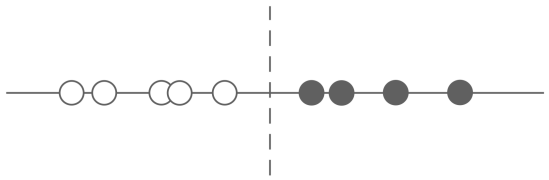


Figure 4: A one-dimension linearly separable dataset.

Figure 5: A one-dimension nonlinearly separable dataset.

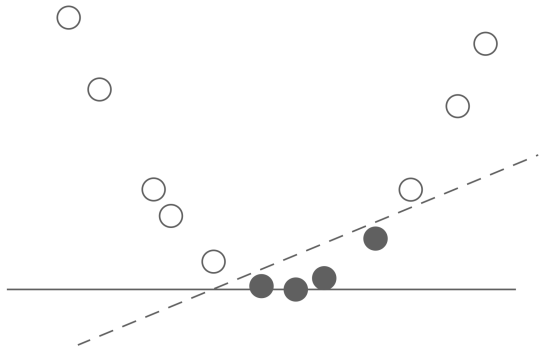


Figure 6: The same dataset of Figure 5 but mapped into a 2-dimensional space using a 2-degree polynomial kernel. Note that the dataset becomes linearly separable after the kernel is used.

There are still cases where even a high-degree polynomial kernel function may not suffice to accurately classify a dataset. These datasets are often addressed by using the Radial Basis Function or *RBF kernel*, which calculates the distance between two vectors with the formula

$$RBF(x, w) = \exp\left(-\frac{||x - w||^2}{2\sigma^2}\right).$$

One can recognize $||x - w||^2$ as the Euclidean distance between two vectors. $\frac{1}{2\sigma^2}$ is

commonly referred to as the γ parameter, which simplifies the formula to

$$RBF(x, w) = \exp\left(-\gamma||x - w||^2\right).$$

We end up with a decaying function where the closer the vectors are to each other, the higher similarity score. In fact, this function is of the form of a bell-shaped curve and larger values of γ yield narrower bell curves. In more general terms, the γ parameter dictates how far the influence of a single training example reaches, with higher values having a smaller sphere of influence than lower values. A small γ may cause overfitting since two instances need to be very close together to be considered similar. Conversely, a large γ may lead to high bias due to the relaxation on the similarity between two instances [VD04].

3.2.2 Separating Hyperplane

To better explain how an SVM creates the hyperplane separating a dataset, we take the linear kernel SVM as an example. This variation allows for faster computations due the simple calculations required by its similarity measure. The linear SVM assumes that there exists a hyperplane such that it completely separates the instances into two groups representing their classes. This hyperplane is created by maximizing the distance from the nearest point from each class. The hyperplane can be written as

$$w \cdot x - b = 0,$$

where w is the normal vector to the hyperplane, x is the slope of the hyperplane and b is an offset constant hyperparameter. Figure 7 illustrates how this function separates two classes of points and their corresponding margin functions, which are obtained by making the hyperplane function equals to 1 and -1.

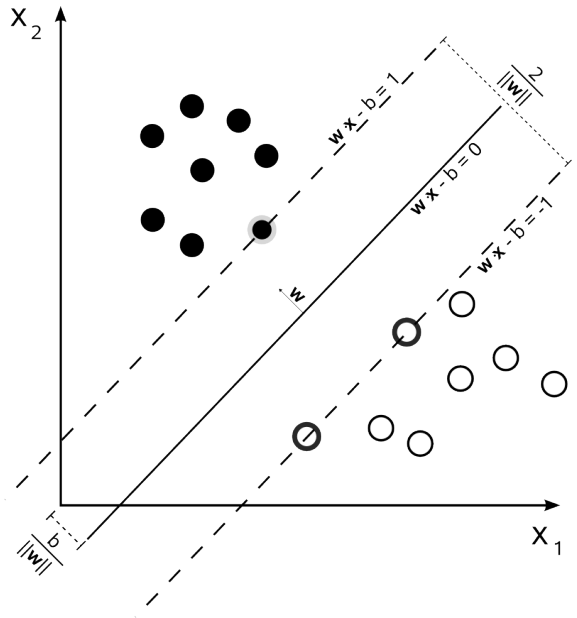


Figure 7: Illustration of how the hyperplane obtained by an SVM separates and classifies points with two different classes.

This approach assumes that the dataset is linearly separable, but in the case of nonlinearly separable problems, we introduce a *hinge loss* function that maximizes the margin by which the hyperplane separates each point, allowing for errors in classification but minimizing them. A hinge loss function is defined as

$$L(y) = max(0, 1 - ty),$$

where y is the output of the hyperplane function and t is the intended output $(-1, 1)$ in a binary classifier. It can be seen that when both y and t have the same sign and $|y| \geq 1$, $L(y) = 0$. When their sign is different, the penalization increases linearly with y, and similarly, if the classification is correct but the margin of the prediction is not sufficient ($|y| < 1$), there is still a penalty returned.

3.2.3 C parameter

Apart from the kernel selection and kernel parameters, SVMs have an additional parameter C which tells them how large is the penalty for misclassifying a training sample when calculating the separating hyperplane. Larger values of C cause the

SVM to find a hyperplane that separates all training samples correctly, which may lead to overfitting of the model. Smaller values of C relax this constraint by allowing more misclassifications in the training phase, which can be useful to preserve the generalization of a model but may also cause underfitting of the data. Experimenting with different values of C is usually done using exponentially growing sequences such as $10^{-4}, 10^{-2}, ..., 10^{4}$.

3.3 Neural Networks

Neural Networks (NN) have been a popular topic in the last years, especially with the advent of *Deep Learning*. At the basic level, a NN consists of many simple interconnected units called neurons, and each of them produces real-valued activations depending on the input given [Sch15]. Deep Learning methods aim at learning features by forming different levels and hierarchies of learning by the composition of lower levels. They are inspired by human cognition and how neurons interact with each other to form what it is conceived as human learning. While the first NN were being developed as early as the 1970s, Deep Learning started to become feasible since the early 2000s with the rise of more powerful hardware and the increase of data accumulation. With these advantages, they outperform kernel-based methods, such as SVMs, in complicated problems. Another reason for its rise in popularity was the ever-lowering computing cost and cheap parallel processing power, which NNs benefit greatly from due to their naturally parallel structure.

An NN text classifier is a network where each neuron receives the term frequencies (or any other feature representation) of a document as input and returns a real value representing the activation of the neurons that can be used to predict the category or class of the document. Each neuron has a set of term weights that are used to compute the activation function. A frequently used activation function is the linear activation function calculated with

$$p_i = A \cdot X_i,$$

where X_i contains the term frequencies of document i, and A the term weights in a neuron; note that both are the same length. Considering a binary classification problem, the sign of p_i will represent the class of document i. Thus, the goal is to learn the correct weight values of A with the use of training data. A naive approach is to start with random weights and iteratively update them when a mistake is

found in the training phase. The update magnitude is called the *learning rate* (μ) and we stop once all training samples are correctly classified or we reach a maximum number of iterations. This is the core idea of the perceptron algorithm [Ros58]. A pseudocode implementation is presented in Algorithm 1.

Data: training documents $\mathbf{X}$, training labels $\mathbf{y}$, learning rate μ
Initialize weight vectors $\mathbf{A}$ to 0 or random real values
converged $\leftarrow$ false
repeat
  converged $\leftarrow$ true
  for X_i, $y_i \in \boldsymbol{X}$, $\boldsymbol{y}$ **do**
    `// Apply training data to the neural network`
    $p_i = \mathbf{A} \cdot X_i$
    if $sgn(p_i) \neq y_i$ **then**
      `// update weights A by learning rate` μ
      $\mathbf{A} = \mathbf{A} + \mu \cdot y_i \cdot X_i$
      converged $\leftarrow$ true
    end
  end
until *converged is false*;

Algorithm 1: Perceptron pseudocode.

The perceptron is a linear classifier, thus it requires a linearly separable dataset, otherwise, it will not converge. If the problem is not linearly separable, there are several approaches that can be taken. One is to implement a pocket algorithm which wraps around the perceptron and keeps the best solution found so far by calculating the classification error of the weight vector in each iteration. If no optimal weights are calculated after k iterations, we return the pocketed solution. The pseudocode for the pocket algorithm is shown in Algorithm 2.

Data: training features $\mathbf{X}$, training labels $\mathbf{y}$, learning rate μ, max iterations k
Result: weight vector $\mathbf{A}_{pocket}$
Initialize $\mathbf{A}_{pocket}$ to 0 or random real values
for $i = 0...k$ **do**
  $\mathbf{A}_t = \mathbf{A}_{pocket}$
  for X_i, $y_i \in \boldsymbol{X}$, $\boldsymbol{y}$ **do**
    $p_i = \mathbf{A}_t \cdot X_i$
    $\mathbf{A}_t = \mathbf{A}_t + \mu \cdot (y_i - p_i) \cdot X_i$
  end
  if $error(\boldsymbol{A}_t) < error(\boldsymbol{A}_{pocket})$ **then**
    $\mathbf{A}_{pocket} = \mathbf{A}_t$
  end
  if $error(\boldsymbol{A}_{pocket} == 0)$ **then**
    break
  end
end

Algorithm 2: Perceptron pseudocode with a pocket implementation.

While the pocket version of the perceptron is guaranteed to return a solution, it may not be the optimal solution, rather the best found so far. Further variations address this caveat, like the Maxover algorithm [Wen95] which converges to a global optimal for separable data and to a local optimal in the case of non-separable data. The perceptron is an online algorithm, meaning the training is done piece-by-piece and does not need the entire training set from the start, making it capable of fitting new samples later on. This is particularly useful in cases where new labeled instances can be obtained after training. For example, an email classifier can be set up to organize a userś inbox, but also the user is continuously tagging emails so the model is updated according to this feedback.

The neuron can be thought of as the most basic unit in an NN while the perceptron is the algorithm the neurons perform for fitting. As mentioned before, the perceptron is suited for separable datasets, but the use of multiple layers of neurons can be used in order to create non-linear classification boundaries, producing the Neural Network. The effect of having multiple layers is to induce piecewise linear boundaries, which in the end approximate enclosing regions containing specific classes.

3.3.1 Multilayer Perceptron

Pal et al. [PM92] describe the multilayer perceptron (MLP) as a network "consisting of multiple layers of simple, two-state, sigmoid processing elements (nodes) or neurons that interact using weighted connections". It is composed of a minimum of three layers: input, hidden and output layers and it may contain more than one hidden layer. There are no connections between neurons within a layer, but all nodes are connected with all nodes of neighboring layers, forming what is called a fully-connected network. The weights represent the correlation between activities happening across nodes in the layers. Figure 8 contains a graphical representation of a multilayer perceptron.

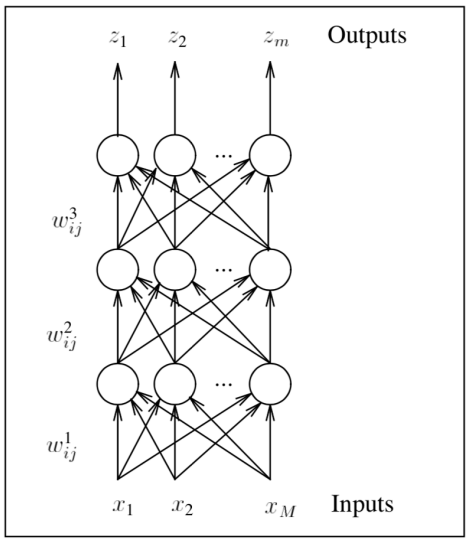


Figure 8: Multilayer Perceptron. Each node is connected to the other nodes in the adjacent layers. Their connection weights are represented by w_{ij}^l where i is the source node, j the destination node and l the layer. Inputs and outputs are represented by x_m and z_m respectively.

3.3.2 Backpropagation

A typical way of training a NN is error backpropagation, in which the classification error is sent back to the network so the parameters can be adapted to reduce the error between the target output and the network output [Seb02]. To perform error

backpropagation we need a loss function that helps us evaluate how far apart are the results from the network with respect to the target results. This function is of the form $f(y_{out}, y_{target})$ and returns a real number depending on the magnitude of the difference between the two values. A commonly used loss function is the sum of squares function calculated with

$$loss(\mathbf{y}_{out}, \mathbf{y}_{target}) = |\mathbf{y}_{out} - \mathbf{y}_{target}|^2,$$

where $\mathbf{y}_{out}$ is the predicted value calculated by the network and $\mathbf{y}_{target}$ is the correct output value we look for. There are a couple of reasons for using the sum of squares as a loss function. First, it does not matter if the network underestimates or overestimates, the error will always be positive, so the goal is to reduce it or reach zero. Second, big errors in the network are more heavily penalized than small errors due to the squared value. If we would plot the obtained prediction value in the x-axis against the loss value in the y-axis, we would get a parabola (sum of squares). We can say that our objective is to "descend" the error parabola to find our desired weight value. To figure out if the weight should be decreased or increased for a smaller error, we use the partial derivative of the loss function with respect to the weights, as it tells us the rate of change at that given point. This is called the *gradient*. Since we are minimizing the loss function, we follow the opposite direction of the gradient. This leads us to the *learning rate*, which is a constant (usually very small) used to update the weights slowly and smoothly in the opposite direction of the gradient, in other words, towards a minimum error. The algorithm which performs this loss function minimization is called *gradient descent* and is widely used in machine learning along with other methods such as *stochastic gradient descent*, *AdaGrad* and *Adam* [RM51, DHS11, KB14]. The backpropagation occurs after the error is calculated at the output layers and its weights are updated; it is then passed back to the hidden layers where the gradient is obtained again but now with respect to their own weights, and the process is repeated until reaching the input layer. This is a simple and intuitive explanation of how error backpropagation works but the mathematical background behind it is more involved.

The strength of NN classifiers is that they can model the data very precisely, leading to high accuracy in the classification but it may also cause overfitting of the training data, so when using a large number of features, NNs may perform worse due to overfitting. Schutze et al. [SHP95] design a NN which minimizes overfitting by having a validation dataset, one-third of the size of the training set. At every iteration, they

run the validation set to obtain its error rate, when this rate goes up, then it means the network is starting to overfit on the training data and they stop, resulting in a more generalized network. The overfitting problem in neural networks is a widely studied topic and extensive effort has been made on finding techniques that reduce variance. Some examples include cross-validation, early stopping, dropout, among others [MB90, CLG01, SHK+14].

Clark et al. [CKP03] present of the first studies on spam classification using NN, which used a fully connected multilayer perceptron using 20 to 40 hidden layers and the validation set error as the early stop. They found that using TF-IDF as document representation yielded the best results and outperformed NB, kNN, and TreeBoost, with performances close to perfect in some datasets.

4 Data Set

Before going into the training of a model, the dataset used must be first transformed into a representation that is both accepted by the model and advantageous to its performance. The structure, the representation and, the information of the input data will strongly affect the performance of a classifier. As we have seen in Sections 2.2 and 2.2.1, the traditional representation of bag-of-words has proven superior to other more complex feature representations. Given that this approach is very fast both for training and prediction phases, we make use of it throughout the empirical part of this study except for the TF-IDF-based Multinomial Naive Bayes. Nevertheless, it is a potential aspect which can be analyzed in more detail for future studies.

We make use of two different datasets throughout the study. The first is the publicly available dataset by Almeida et al. [AHY11] which is composed of several collections:

- A collection of 425 spam messages from the Grumbletext website, where UK users report spam messages [Una]. Almeida et al. had to manually review the complaint reports to extract the actual message content, but this guarantees the data comes from real traffic and labeled by an end-user.
- A corpus of 3,375 ham messages of the NUS SMS Corpus, which is a dataset of over 10,000 legitimate messages gathered by the Department of Computer Science at the National University of Singapore, where the messages were obtained from volunteers [Kan11].
- 450 of ham messages from Caroline Tag's PhD Thesis [Tag09].
- The SMS Spam Corpus v0.1 containing 1,002 ham messages and 322 spam messages [HS11]. This dataset has been used in other academic studies such as Scott et al. [SM99] and Cormack et al. [CGHS07].

Together, the collection contains 4,827 ham and 747 spam messages for a total of 5,574 messages. The messages were modified to protect the privacy of the users by changing names, addresses, and other personal information. The number of messages in this dataset allows us for enough room to test how training models with a different number of messages affect their performance.

The second dataset we use is gathered from the traffic of the Veoo platform. It is composed of 1,318 ham and 159 spam messages, which were also modified to

hide personal and sensitive information about the users. While a first iteration of the spam detection system was implemented using the dataset in Almeida et al. [AHY11] with favorable results, a dataset of the actual traffic was created so more controlled experiments can be made on the efficacy of training models using these two datasets, either together or separate.

4.1 Preprocessing

Datasets often contain pieces of information that may degrade the performance of a learning algorithm. There are several studies on text classification which make use of preprocessing techniques such as stop-word removal, tokenization, and word-stemming to eliminate or convert this data into more valuable information [SR03, LSTO10, CT94]. The preprocessing done was mainly to ensure that messages had a valid encoding and characters, and obscuring personal details, everything else was left as-is.

5 Platform Architecture

5.1 Company

Veoo is a mobile solutions platform that enables businesses to deliver mobile services by simplifying technical, commercial and regulatory complexities. Businesses looking to implement mobile marketing campaigns, mobile payments, and other mobile-based services can employ Veoo's processes without having to work through the difficulties of implementing their own messaging technology and having to worry about the regulations that different regions or countries may have. It offers different interfaces that allow clients from varying technical backgrounds to develop text messaging solutions. Specific examples of these interfaces are HTTP and the SMPP protocol. The SMPP (Short Message Peer to Peer) protocol was established and managed by the SMS Forum organization which provided a flexible solution for the sending and receiving of mobile short message data between external entities and message centers. The protocol supports several two-way messaging functions such as:

- Transmitting messages from an ESME (External Short Message Entity) to single or multiple destinations via an SMSC (Short Message Service Center).
- Querying the status of a short message.
- Canceling or replacing a short message.
- Defining the data coding of a message, examples of supported encoding include GSM, UTF-8, and Latin-1.
- Providing a service type to a message e.g. voicemail notification, cellular paging.

SMSCs commonly require third-parties to communicate with them using this protocol, and due to some nuances in the protocol, it is not unusual to see different implementations that may be incompatible with each other. Veoo offers its users a transparent SMPP connection to several SMSCs without having to deal with the differences between their protocols.

5.2 Architecture

The platform, called *spice*, provides customers with SMPP and HTTP gateways to submit high volumes of SMS traffic. *Spice* is built using technologies such as Go, Ruby on Rails, Python, Kubernetes, AWS, RabbitMQ, Redis and MySQL and it is composed of four main components: *ESME, router, supplier* and *canopener*. *ESME* handles the bind or connection that a client uses to send their text message requests, thus there are several of them running at the same time in the platform, each with its own configuration. The received requests are forwarded to *router*, which is in charge of figuring out what *supplier* must be used to handle the request. It does so by querying *kannel routing*, our message routing component, to obtain the *supplier* to be used. After forwarding the message to the appropriate *supplier*, the *router* also sends an identical copy of the request to *canopener*, the process that handles the spam detection. Both *router* and *canopener* are designed to be horizontally scalable. *Canopener* then uses the previously-trained machine learning model to perform the classification and sends the results to the data aggregation system, HostedGraphite. The *supplier* process is in charge of processing messages from *router* and it maintains an SMPP bind to our telecom providers, who perform the delivery of the text message. Since there are up to 200 different providers available, there is one process per *supplier*, each with its own configuration.

5.3 Canopener: A Spam Detection System

The component in charge of handling the spam detection is called *canopener*. It is a process written in Python which makes use of the widely-known machine learning library *scikit-learn*. It consumes from a RabbitMQ queue of messages and publishes its results to HostedGraphite. These results contain the client (*ESME*) ID that submitted the message and the spam or ham label. In this way, the message classifications are aggregated for each *ESME* and thresholds are set to the number of spam messages that can be sent in a given amount of time. If any ESME presents a large volume and/or spike of spam traffic, an alert is immediately sent to the support team which then can corroborate that the traffic is indeed malicious. Due to the nature and importance of the traffic, a design decision was made to only alert on such spam spikes, rather than blocking the traffic automatically, since further human verification is needed to confirm that the incoming messages are not desired. This notification is received in a very short time, making the alarm system highly

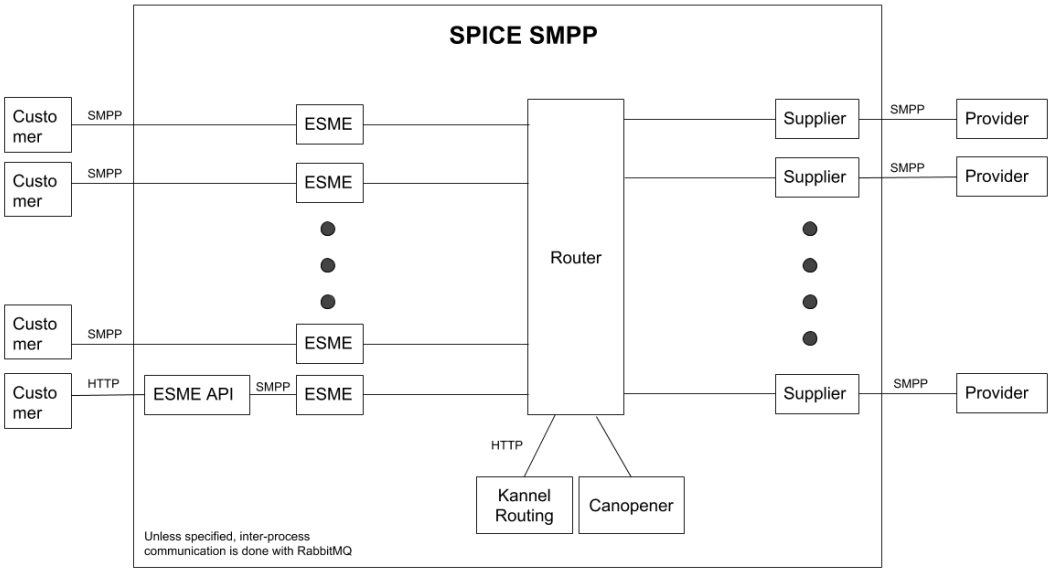


Figure 9: Diagram illustrating a high-level overview of the *spice* platform architecture.

responsive to incoming spam traffic spikes. If the traffic is found to be malicious, then the rejection of spam messages is activated. In this scenario, the particular *ESME* is marked, and any message received through it is tagged. When *router* receives messages with this tag, it will not route it directly to the *supplier*, it will only be sent to *canopener*. Here the request is processed as usual, but if it is found to not be spam, then it will be forwarded to *supplier*. If it is spam, then it is forwarded to a small service in charge of sending rejected receipts, and the request is not processed further.

Since the content of identified spam traffic needs to be manually checked, we display the messages' content using a word cloud tool, so it is straightforward to do a quick judgment of the overall content of the spam without having to look at individual instances. In the case that deeper inspection is needed, there is also a visualization tool that allows a person to look at the individual instances of spam traffic. Making use of HostedGraphite, we have a dashboard to illustrate the spam messages coming from a specific client; Figure 10 shows an example of such dashboard.

We separate the spam detection process from the main business logic since we want

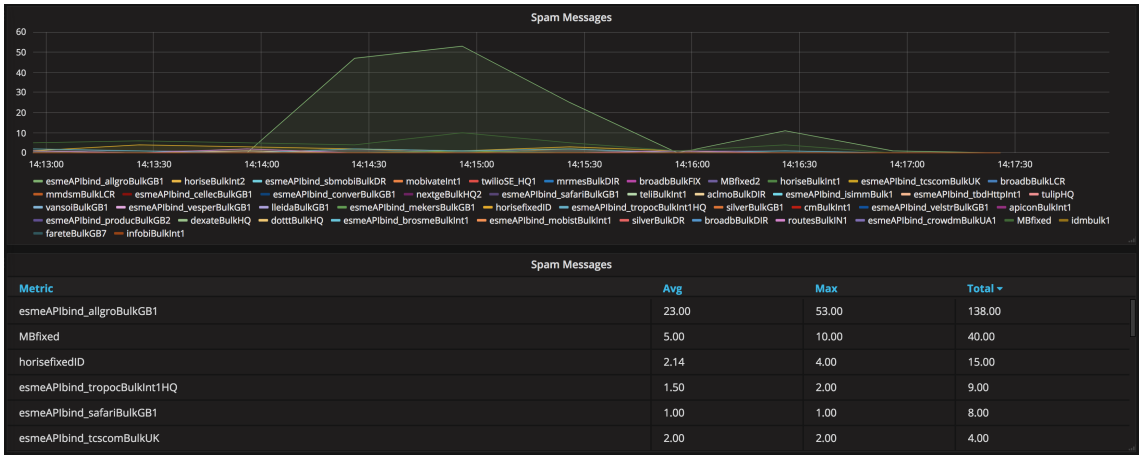


Figure 10: HostedGraphite dashboard showing the spam messages traffic of each individual ESME.

to avoid degradation of the quality of service in the case of large spikes of traffic or other problems which may arise with *canopener*, given that it is not a required part of the sending flow.

Besides the normal classification of a message, we also have an analytics process where different models and tokenization techniques are tested and their results published to our data aggregator. This process is independent of our business logic and was created to obtain experimental results of the different models and techniques that can be employed.

6 Experiments

In this section, we present the methodology used to evaluate the different learning models, their results and how they can be applied to the spam detection platform. In Section 6.1, we present the different criteria we are considering when comparing models. In Section 6.2, we describe the models used and the hyperparameters being tested. In Section 6.3, we present the experiments made with the publicly available dataset of text messages to get a basis of how well the models work. Then, in Section 6.4, we present the results obtained with our own dataset of text messages and compare it with the previous one.

6.1 Criteria

Taking to account the unbalanced dataset we are handling and that, depending on the situation, the cost of a false positive may be different from a false negative, we present different criteria as to how models are compared depending on specific needs. We list these criteria along with their motivations and the metrics to represent them.

- *Overall Performance.* The aim is to obtain the most accurate model overall, without focusing on specific spam or ham prediction. For this, we will use balanced accuracy as the metric.
- *Overall Spam Performance.* The objective is to detect as much spam and as accurately as possible using spam F1-score.
- *Detect all spam.* Spam messages are considered especially harmful in this scenario, and blocking a few ham messages is tolerable. Using the spam recall metric, we can choose the model which is best at detecting the most spam.
- *Detect only spam.* While spam messages may be harmful, we want to make sure only spam is blocked from traffic and the valid traffic is unaffected. Spam precision is the metric chosen for this.
- *Overall Ham Performance.* We want to focus on valid traffic overall, thus we choose the ham F1-score balancing between ham recall and ham precision.
- *Detect all ham messages.* Here the priority is to avoid blocking any valid traffic, considering businesses may want to let some spam go through to make sure all ham messages are delivered. We use ham recall to compare models.

- *Detect only ham messages.* Valid traffic is still of priority, and only ham messages should be let through. Using ham precision we optimize for this.

Any trivial classifier can achieve perfect recall or precision on a certain class by labeling all instances as that class. By optimizing directly on these metrics, we might choose a model that does approximately that, causing deficiencies in the classification of the other class. We decided to maximize these four metrics (spam/ham recall/precision) in a separate step. We collect the models with the best spam F1-score and look at their spam recall and precision to select the best among them. The same is done for ham recall and precision using the best ham F1-score models. Selecting the models this way, we ensure that the classification of one class is not sacrificed for better precision or recall of the other.

Since the number of figures required to present all the combinations of models, their hyperparameters and different metrics are very high, and it would make it difficult for the reader to follow along, we present only the key figures in this section. Any other figures will be located in the Appendix.

6.2 Models Used

During this empirical study, the three main models to be tested were Naive Bayes, SVMs, and Multilayer Perceptron. Each of them can have variations and tests were performed with the different parameters to obtain the best results. They were also tested with different numbers of features. The initial range of features was from 500 to 5000 with increments of 500. Each feature represents a word in the corpus, and its value is the number of occurrences in a given document. The number of features is equal to the n most common words. We observed that some models benefitted from fewer features, or their optimal scores appeared to lie between 500 and 1500 features. For these cases, we also tested ranges including a lower number of features or more values between 500 and 1500.

Multinomial Naive Bayes, which works well for data involving counts, was tested both with term frequencies and TF-IDF as features. We will refer to them as MNB and TF-IDF MNB.

There are several kernels that can be chosen when using SVMs. We compare linear, polynomial, and RBF kernels. The machine learning library used, *scikit-learn*, offers two different linear kernel implementations: *LinearSVC* and *SVC* with *kernel='linear'*. The difference between them is the C-library used, with the former

using *liblinear* and the latter *libsvm*. The *scikit-learn* documentation does not mention any comparison between the two, therefore, we decided to run experiments with both models and we refer to them as Linear SVM and LinearK SVM, respectively. We executed preliminary tests to find suitable degrees of the polynomial kernel and avoid running unnecessary tests with poorly performing kernels. We found that polynomial kernels of degree higher than 3 produced lower scores, thus they are not included in the in-depth experiments. RBF SVM uses an extra parameter, γ, and we use the default value set by the library (1/number of features). The performance of an SVM is also dependent on its hyperparameter C, the penalization factor, and during the experiments, we seek to find the best value for each kernel used. An initial range of $[10^{-5}, 10^{4}]$ with logarithmic intervals was used to have an overview of the performance scores. If the scores appeared to have further room for improvement beyond this range, the range was extended for that particular SVM until no more improvement was seen. On the other hand, there were values of C which dramatically lowered the scores to values less than 0.5, and they are not shown in the figures. With these remarks in mind, note that the figures obtained for each kernel and score have different ranges of C emphasizing ranges with the best performing hyperparameter combinations.

In the Multilayer Perceptron (MLP), the structure of the network can take a wide variety of configurations. We chose to limit the experiment to a maximum of 5 layers and a maximum of 2000 nodes per layer since preliminary experiments with more layers and nodes showed longer training time without significant improvement. The range of nodes used was $[5, 25, 50, 75, 100, 500, 1000, 2000]$. Since it is not possible to clearly visualize the results for all the network configurations at the same time, we only report the best score out of all configurations of layers and nodes. There are other hyperparameters which can be tuned in the MLP such as the learning rate, the regularization term, and the stopping policy. The learning rate was set at 0.001, the default of 0.0001 L2 penalty was used and early stopping was activated.

6.3 Base Experiments

We employ the publicly available dataset by Almeida et al. [AHY11] composed of 4,827 ham messages and 747 spam messages to carry out our initial experiments and establish a reference for the results done on our own dataset.

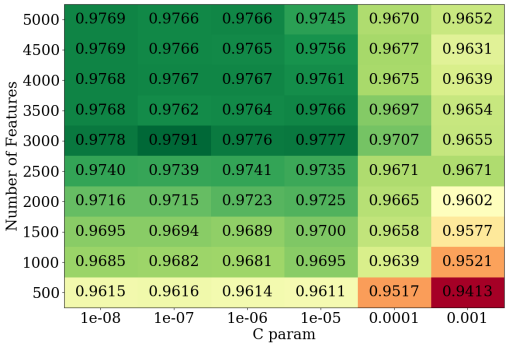
To reduce the variance and bias of the results, we perform our tests using 10-fold cross-validation, and ten different iterations are carried through. We also keep a

separate testing set, composed of 10% of the original dataset, to obtain the final scores.

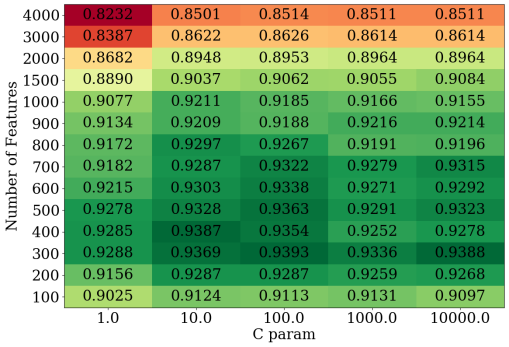
To illustrate the process of comparing models, their parameters and criteria, let us present the results for Linear SVM and RBF SVM using balanced accuracy as the metric. In Figure 11a, we see that the best balanced accuracy obtained by Linear SVM was 0.9791 with 3500 features and C=1e-07. In Figure 11b, the highest balanced accuracy obtained by RBF SVM was 0.9393 with 300 features and C=100. Note that the ranges of C and the number of features are different since we are focusing on value ranges with the highest scores. Then, using our separate testing set, we train both models with the given parameters and obtain their final balanced accuracy scores: 0.9794 for Linear SVM and 0.9742 for RBF SVM. Therefore, we can conclude that the most appropriate model to use when the target metric is overall performance (measured as balanced accuracy) is Linear SVM with 3500 features and C=1e-07. A head-to-head learning curve comparison of these two models with the hyperparameters mentioned is shown in Figure 11c.

After performing the same procedure for all models, we present Table 1 containing the summary of results from the different models. They contain the best hyperparameters for each model using balanced accuracy, spam F1-score, and ham F1-score. In a different table, we include the hyperparameter values used that produced the scores, with F representing the number of features, C as the C value for SVMs, and L as the network structure for MLPs. These scores were obtained by training the model with the given hyperparameters and using the separate set for testing. Note that the error is only included for MLPs since they are the only models in which separate evaluations of the same set can produce different results due to their initial random state. After this, we collect the models with best spam F1-score, evaluate their spam precision and recall, and keep the top 3. We do the same for ham precision and recall. The scores are located in Appendix 16. The top 3 models for each score can be seen in Table 3.

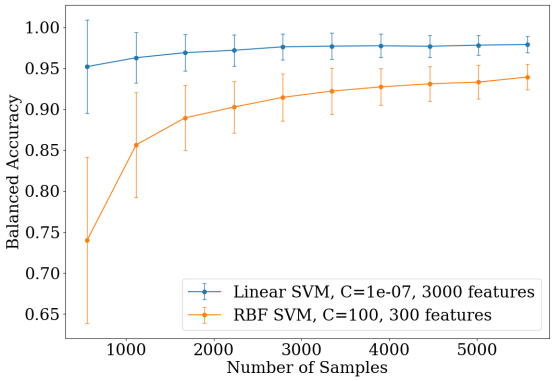
On closer inspection, we start to find some patterns on the value of the hyperparameters. For example, most top-performing models use a large number of features (2500 to 5000) with the exception of RBF SVM with 300 features and Poly-2 SVM with 400. We also see that MNB (F=3500) appears on 5 out of the 7 scoring criteria with the same parameters, which means we could use one model for several purposes, making it a good candidate for all-around classification. From Section 2.1.4, we know that ham recall is the percentage of ham messages the model was able to



(a) Linear SVM balanced accuracy heatmap



(b) RBF SVM balanced accuracy heatmap



(c) Balanced accuracy learning curves

Figure 11: Linear SVM and RBF SVM balanced accuracy score heatmaps and learning curves.

label, and given our unbalanced dataset, labeling most messages as ham would result on a high ham recall score. Let us look at a concrete example. In Appendix 14e, we see that RBF SVM obtained a perfect ham recall score by using 4500 features and setting C=1.0. However, the balanced accuracy of this RBF SVM is 0.8232 and its spam recall is 0.6466; this leads us to conclude that the model has a high ham recall score because it labels most instances as ham, and not because it models the data accurately. This is an example that demonstrates the decision to preselect models with high F1-score before optimizing for recall and precision.

Figure 12a shows that Linear SVM has a particularly flat learning curve, with a high balanced accuracy, which indicates it can be used in the case the dataset is small, since it performs well with small number of samples. This coincides with what was

Model	Balanced Accuracy	Spam F1-score	Ham F1-score
MNB	0.9812	0.9606	0.9951
TF-IDF MNB	0.9355	0.9244	0.9913
Linear SVM	0.9794	0.9291	0.9912
LinearK SVM	0.9545	0.9355	0.9893
Poly-2 SVM	0.9446	0.9322	0.9923
Poly-3 SVM	0.9456	0.9076	0.9894
RBF SVM	0.9742	0.9440	0.9932
MLP	0.9626 ±0.0061	0.9624 ±0.0099	0.9947 ±0.0008

Table 1: Best balanced accuracy, spam F1-score and ham F1-score gathered from each model.

Model	Balanced Accuracy	Spam F1-score	Ham F1-score
MNB	F=3500	F=3500	F=3500
TF-IDF MNB	F=2000	F=2000	F=2000
Linear SVM	F=3000, C=1e-7	F=3000, C=0.0001	F=3000, C=0.0001
LinearK SVM	F=400, C=0.01	F=700, C=0.01	F=400, C=0.01
Poly-2 SVM	F=300, C=100	F=400, C=10	F=400, C=10
Poly-3 SVM	F=200, C=1000	F=300, C=100	F=300, C=100
RBF SVM	F=300, C=100	F=400, C=10	F=400, C=10
MLP	F=3000, L=2000-2000-2000	F=2500, L=1000-1000	F=1500, L=2000-2000

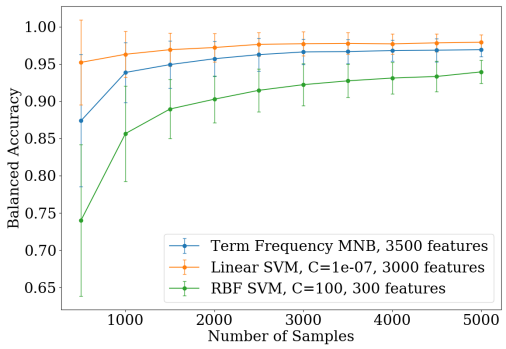
Table 2: The hyperparameters used to obtain the best balanced accuracy, spam F1-score and ham F1-score from each model.

said in Section 3.2 about SVMs performing well with small datasets.

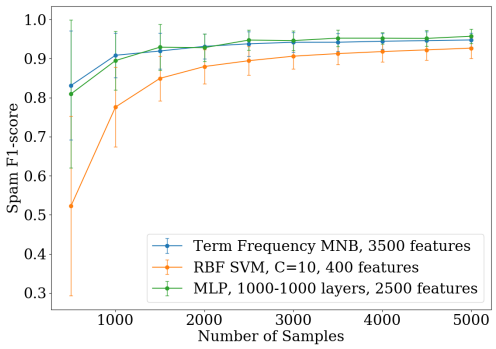
The time performance of the models is also of interest to the platform. Since the time taken to fit a model does not particularly concern us given that the fitting can be done separate from the platform workflow, we rather look at the time spent scoring new instances. It was a general trend for all models that the number of features greatly affected the time taken to score, thus it is a good rule of thumb to choose models with the lower number of features. For example, consider an MLP with 1 hidden layer of 25 nodes; if it is trained using 500 features, the scoring time is 4.34 times less than using 2000 features. The trade-off in time performance is noticeable given that 2000 features only yield a 0.25% improvement on balanced accuracy compared to 500 features. Also, as expected, an increment in the number of nodes made the scoring slower, thus networks with fewer nodes overall are preferred

Score	Models		
Balanced accuracy	MNB (F=3500)	Linear SVM (F=3000, C=1e-7)	RBF SVM (F=300, C=100)
Spam F1-score	MLP (F=2500, L=1000-1000)	MNB (F=3500)	RBF SVM (F=400, C=10)
Ham F1-score	MNB (F=3500)	MLP (F=1500, L=2000-2000)	RBF SVM (F=400, C=10)
Spam Recall	MNB (F=3500)	MLP (F=2500, L=1000-1000)	RBF SVM (F=400, C=10)
Spam Precision	Poly-2 SVM (F=400, C=10.0)	MLP, (F=2500, L=1000-1000)	TF-IDF MNB (F=2000)
Ham Recall	Poly-2 SVM (F=400, C=10.0)	TF-IDF MNB (F=2000)	MLP (F=1500, L=2000-2000)
Ham Precision	MNB (F=3500)	RBF SVM (F=400, C=10)	Linear SVM (F=3000, C=0.0001)

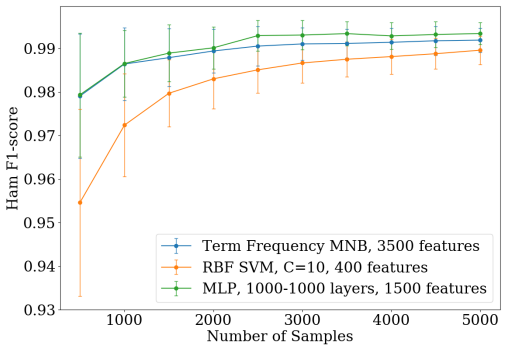
Table 3: The top 3 models (and their parameters) in descending order for each score.



(a) Balanced accuracy learning curve



(b) Spam F1-score learning curve



(c) Ham F1-score learning curve

Figure 12: Learning curves for balanced accuracy, spam F1-score and ham F1-score with their top 3 models.

if scoring time must be fast.

Another parameter to consider when looking at time performance is C from SVMs. For the linear SVMs, larger values of C produced smaller scoring times. Since C acts as the penalizing factor of the misclassified samples, a larger C produces a better

separating hyperplane which requires fewer support vectors, thus the scoring time is shorter. For the RBF kernel SVM, larger values of C drive the algorithm to consider more and more samples as class boundaries, incrementing the number of support vectors, and causing longer scoring times.

We take the models from Table 3 and present their average scoring times in Table 4. The scoring time is the time taken to evaluate the entire validation set during cross-validation. We see that the two slowest algorithms are RBF SVM and LinearK SVM by a large margin, although for lower number of features, RBF SVM becomes one order of magnitude faster. MNB and Linear SVM are the two fastest scoring models, which is an important factor to consider since the platform requires a high volume of messages to be handled.

Model	Average scoring time (s)
MNB (3500)	0.0190 ±0.0006
Linear SVM (3000, C=1e-07)	0.0486 ±0.0072
MLP (2500, 25)	0.0578 ±0.0002
MLP (1500, 1000-1000)	0.2019 ±0.0709
MLP (2500, 1000-1000)	0.3286 ±0.0812
MLP (5000, 2000)	0.4777 ±0.0002
RBF SVM (300, C=100)	0.8059 ±0.0136
RBF SVM (400, C=10)	1.7414 ±0.6352
LinearK SVM (4000, C=0.0001)	11.4040 ±0.4896
RBF SVM (4500, C=1.0)	22.8556 ±0.0042

Table 4: Average scoring times of the top performing models in 3.

6.4 Real Traffic Dataset Experiments

The second set of experiments was done using the dataset we gathered from the Veoo platform, thus providing real-life training instances and results similar to what we would see if the spam filter is used. This dataset is composed of 1,318 ham and 159 spam messages, and we perform 3-fold cross-validation and ten different iterations to avoid overfitting over the splits. We keep a separate set composed of 10% of the samples to be used as the final testing set.

At first sight, we note that most of the scores are lower than the first dataset. The largest difference is at the spam F1-score. Judging by the results, spam is particularly hard to detect, this is not surprising considering the size of the dataset and its class imbalance. In Table 7, we show the models with their best spam recall and precision. Let us inspect their results from other metrics. For example, Linear

Model	Balanced Accuracy	Spam F1-score	Ham F1-score
MNB	0.9517	0.8000	0.9759
TF-IDF MNB	0.8926	0.7879	0.9761
Linear SVM	0.9762	0.8235	0.9795
LinearK SVM	0.8960	0.8125	0.9796
Poly-2 SVM	0.9273	0.8485	0.9829
Poly-3 SVM	0.8960	0.8485	0.9829
RBF SVM	0.9517	0.8235	0.9795
MLP	0.9376 ±0.0212	0.8337 ±0.0159	0.9804 ±0.0020

Table 5: Best balanced accuracy, spam F1-score, and ham F1-score gathered from each model.

Model	Balanced Accuracy	Spam F1-score	Ham F1-score
MNB	F=1000	F=300	F=300
TF-IDF MNB	F=500	F=500	F=500
Linear SVM	F=600, C=1e-05	F=400, C=0.001	F=400, C=0.001
LinearK SVM	F=1200, C=0.001	F=800, C=0.001	F=800, C=0.001
Poly-2 SVM	F=300, C=10.0	F=300, C=10.0	F=300, C=10.0
Poly-3 SVM	F=100, C=100.0	F=300, C=10.0	F=300, C=10.0
RBF SVM	F=100, C=10.0	F=100, C=1.0	F=100, C=1.0
MLP	F=1000, L=2000-2000-2000-2000	F=1500, L=1000-1000-1000	F=1500, L=1000-1000-1000

Table 6: The hyperparameters used to obtain the best balanced accuracy, spam F1-score and ham F1-score from each model.

Model	Spam Recall	Spam Precision
MNB	0.8125	0.7500
TF-IDF MNB	0.8125	0.7647
Linear SVM	1.0000	0.7778
LinearK SVM	0.8125	0.8125
Poly-2 SVM	0.8750	0.8125
Poly-3 SVM	0.8125	0.8462
RBF SVM	0.9375	0.8125
MLP	0.8562 ±0.0447	0.7820 ±0.0056

Table 7: Best spam recall and precision gathered from each model.

SVM with 900 features and C=1e-7 produced a perfect spam recall but looking at Appendix 15e, its spam precision was 0.6154, thus may not be a great candidate for overall spam detection. Choosing the best spam recall score may require us

to considerably sacrifice the performance of other metrics. Let us now consider balanced accuracy as the main criteria. We find model configurations that have both high balanced accuracy and low spam recall, such as the previously mentioned Linear SVM (see Appendix 15a and 15d). This is a shortcoming we face when optimizing solely on one metric without considering the others, thus another selection strategy is needed to overcome this.

In Table 8, we present the top 3 models of each metric. It can be used as a quick guide for choosing the best model and its hyperparameters, however, we present a more in-depth analysis and a final list of recommendations for each criterion.

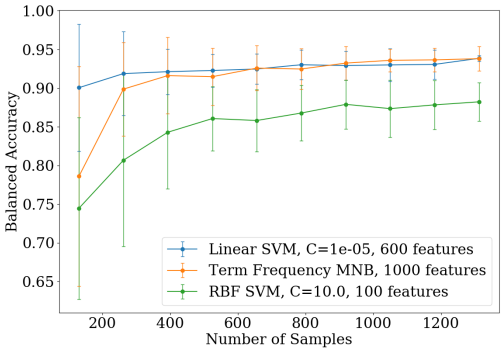
Score	Models		
Balanced accuracy	Linear SVM (F=600, C=1e-05)	MNB (F=1000)	RBF SVM (F=100, C=10.0)
Spam F1-score	Poly-2 SVM (F=300, C=10.0)	Poly-3 SVM (F=300, C=10.0)	MLP (F=1500, L=1000-1000-1000)
Ham F1-score	Poly-2 SVM (F=300, C=10.0)	Poly-3 SVM (F=300, C=10.0)	MLP (F=1500, L=1000-1000-1000)
Spam Recall	Linear SVM (F=900, C=1e-07)	RBF SVM (F=100, C=10.0)	Poly-2 SVM (F=100, C=100.0)
Spam Precision	Poly-3 (F=200, C=0.1)	RBF SVM SVM (F=800, C=0.1)	LinearK SVM (F=300, C=0.001)
Ham Recall	Poly-2 SVM (F=5000, C=0.1)	Poly-3 SVM (F=5000, C=0.1)	RBF SVM (F=5000, C=0.1)
Ham Precision	Linear SVM (F=600, C=1e-05)	RBF SVM (F=100, C=10)	Poly-2 SVM (F=300, C=10.0)

Table 8: The top 3 models (and their parameters) in descending order for each score.

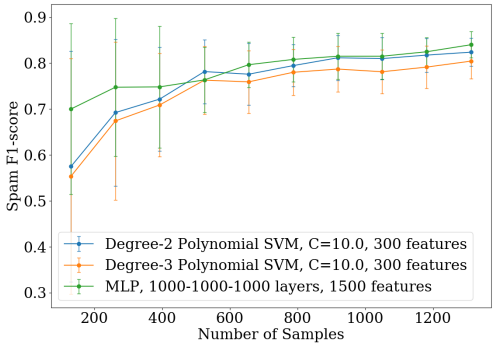
6.4.1 Alternative Model Selection

Considering that spam precision and recall are the main deficiencies, we take an alternative approach to choosing a model. First, we prioritize spam F1-score, since it balances between spam precision and recall, and collect the models which produced the highest spam F1-scores. Next, we look at the criteria (see Section 6.1) we are interested in, and select the best score. After inspecting the testing results of Poly-2 SVM, Poly-3 SVM, MLP, Linear SVM and RBF SVM with their best spam F1-scores hyperparameters, we present the following model recommendations for each criteria:

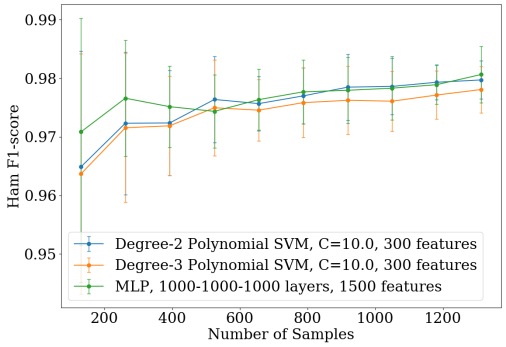
- Balanced Accuracy. MLP (F=1500, L=1000-1000-1000).
- Spam F1-score. Poly-2 SVM (F=300, C=10.0) or Poly-3 SVM (F=300, C=10.0).
- Ham F1-score. Poly-2 SVM (F=300, C=10.0) or Poly-3 SVM (F=300, C=10.0).
- Spam Recall. MLP (F=1500, L=1000-1000-1000).
- Spam Precision. Poly-2 SVM (F=300, C=10.0) or Poly-3 SVM (F=300, C=10.0).



(a) Balanced accuracy learning curve



(b) Spam F1-score learning curve



(c) Ham F1-score learning curve

Figure 13: Learning curves for balanced accuracy, spam F1-score and ham F1-score.

- Ham Recall. Poly-2 SVM (F=300, C=10.0) or Poly-3 SVM (F=300, C=10.0).
- Ham Precision. MLP (F=1500, L=1000-1000-1000).

The main difference among the polynomial kernel SVMs and the MLP is that the SVMs were better at collecting spam and MLPs at accurately detecting spam. It should also be mentioned that the ham F1-scores of the five aforementioned models are greater than 0.97, thus placing them as strong candidates for ham-related criteria, regardless of their spam F1-score performance.

6.4.2 Time Performance

Lastly, we present the time performance of the models in Table 8. Similar to the first set of experiments, we observe that MNB and Linear SVM are the fastest scoring models and that a smaller amount of features is a major factor on time performance.

Looking at the final three recommended models, we observe that Poly-2 SVM is 10% faster than the other two. If time performance is a priority, it is certainly a factor to consider.

Model	Average scoring time (s)
MNB (F=1000)	0.0038 ±0.0003
Linear SVM (F=600, C=1e-05)	0.0071 ±0.0003
Linear SVM (F=900, C=1e-07)	0.0109 ±0.0004
Poly-2 SVM (F=100, C=100.0)	0.0335 ±0.0055
RBF SVM (F=100, C=10.0)	0.0414 ±0.0014
LinearK SVM (F=300, C=0.001)	0.0892 ±0.0017
Poly-3 SVM (F=200, C=0.1)	0.1222 ±0.0064
Poly-2 SVM (F=300, C=10.0)	0.1645 ±0.0195
MLP (F=1500, L=1000-1000-1000)	0.1801 ±0.0251
Poly-3 SVM (F=300, C=10.0)	0.1814 ±0.0085
RBF SVM (F=800, C=0.1)	0.7011 ±0.0211
RBF SVM (F=5000, C=0.1)	6.4759 ±0.1071
Poly-2 SVM (F=5000, C=0.1)	6.8377 ±0.1885
Poly-3 SVM (F=5000, C=0.1)	6.9241 ±0.3715

Table 9: Average scoring times of the top performing models in 8.

7 Summary

In this thesis we approached the spam detection machine learning problem from a real-world point of view, gathering our own data and using the empirical results to make informed recommendations on which model was most appropriate given certain criteria. We discovered that real traffic is much harder to classify than the dataset by Almeida et al. [AHY11], particularly spam messages. Models and their hyperparameters differed greatly between these two sets of experiments; whereas the first set of experiments had models such as MNB, MLP and RBF SVM as the best ones throughout metrics, we found models such as Poly-2 SVM and Poly-3 SVM to perform better. The main challenge was that spam recall and precision were very low for models where other metrics were high. Difficulty detecting spam may be due to different reasons. For example, our own dataset is one-fifth of the size, therefore even a small amount of noise in the data might cause the algorithm to model irrelevant information. Also, the class imbalance is even more pronounced, with our dataset having 10% versus 13.5% on the other one. Finally, we have several messages in languages other than English, which would make it hard for the model to classify correctly unless keywords are shared with other messages.

The experimental results led us to take another approach to select models, and avoid only picking the model with the best score, since they may be inferior with respect to other metrics. For this, we chose models with good spam classification performance using spam F1-score, then obtained all their metric scores to get a better picture of their other strengths. Using this approach we concluded that there were 3 models that could be used for any criteria, without sacrificing performance on others. The final recommendations to use in the platform are listed at the end of Section 6.4.

The findings of this thesis leave further room for future work and improvements. Given the small size of the dataset, it would be beneficial to continue sampling traffic and collecting spam messages, preferably reducing the proportion of ham to spam, and including instances in other languages. Class imbalance problems have been studied a lot, and we can make use of existing solutions to gain performance improvements. Another potential improvement would involve preprocessing messages, like removing stop words or attempting to extract information from URLs, since a great number of messages include them. Lastly, we acknowledge the fact that the machine learning models used in this study are only a small fraction of what is currently available, and it would be beneficial to explore other models on text and spam classification to find more suitable solutions.

References

AHY11 Almeida, T. A., Hidalgo, J. M. G. and Yamakami, A., Contributions to the study of sms spam filtering: new collection and results. *Proceedings of the 11th ACM Symposium on Document Engineering*. ACM, 2011, pages 259–262.

AZ12 Aggarwal, C. C. and Zhai, C., A survey of text classification algorithms. In *Mining Text Data*, Springer, 2012, pages 163–222.

BM09 Boiy, E. and Moens, M.-F., A machine learning approach to sentiment analysis in multilingual web texts. *Information Retrieval*, 12,5(2009), pages 526–558.

C+08 Cormack, G. V. et al., Email spam filtering: A systematic review. *Foundations and Trends in Information Retrieval*, 1,4(2008), pages 335–455.

CGHS07 Cormack, G. V., Gómez Hidalgo, J. M. and Sánz, E. P., Spam filtering for short messages. *Proceedings of the 16th ACM Conference on Information and Knowledge Management*. ACM, 2007, pages 313–320.

CHV99 Chapelle, O., Haffner, P. and Vapnik, V. N., Support vector machines for histogram-based image classification. *IEEE Transactions on Neural Networks*, 10,5(1999), pages 1055–1064.

CKP03 Clark, J., Koprinska, I. and Poon, J., A neural network based approach to automated e-mail classification. *Proceedings of IEEE/WIC International Conference on Web Intelligence*. IEEE, 2003, pages 702–705.

CLG01 Caruana, R., Lawrence, S. and Giles, C. L., Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Proceedings of Advances in Neural Information Processing Systems*. NeurIPS, 2001, pages 402–408.

Cro13 Crocker, P., Converged-mobile-messaging analysis and forecasts. *Giga Omni Media, New York*.

CT94 Cavnar, W. B. and Trenkle, J. M., N-gram-based text categorization. *Proceedings of the 3rd annual Symposium on Document Analysis and Information Retrieval*, volume 161175. IEEE, 1994.

CV95 Cortes, C. and Vapnik, V., Support-vector networks. *Machine Learning*, 20,3(1995), pages 273–297.

DHS11 Duchi, J., Hazan, E. and Singer, Y., Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, pages 2121–2159.

DWV99 Drucker, H., Wu, D. and Vapnik, V. N., Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, 10,5(1999), pages 1048–1054.

FP96 Fawcett, T. and Provost, F. J., Combining data mining and machine learning for effective user profiling. *Proceedings of Knowledge Discovery and Data Mining*. AAAI, 1996, pages 8–13.

G+98 Gunn, S. R. et al., Support vector machines for classification and regression. Technical Report, University of Southampton, 1998.

GC09 Guzella, T. S. and Caminhas, W. M., A review of machine learning approaches to spam filtering. *Expert Systems with Applications*, 36,7(2009), pages 10206–10222.

GE03 Guyon, I. and Elisseeff, A., An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3,Mar(2003), pages 1157–1182.

GSM11 GSMA, Sms spam and mobile messaging attacks introduction, trends and examples. *GSMA*. URL `https://www.yumpu.com/en/document/view/22110268/sms-spam-and-mobile-messaging-attacks-introduction-gsma/2`.

Hal99 Hall, M. A., *Correlation-based feature selection for machine learning*. Ph.D. thesis, University of Waikato Hamilton, 1999.

HCL+03 Hsu, C.-W., Chang, C.-C., Lin, C.-J. et al., A practical guide to support vector classification. Technical Report, National Taiwan University, 2003.

HMS06 Hand, D., Mannila, H. and Smyth, P., *Principles of Data Mining*, volume 2. Wiley Online Library, 2006.

HS11 Hidalgo, J. M. G. and Sánz, E. P., Sms spam corpus v.0.1, 2011. URL `http://www.esp.uem.es/jmgomez/smsspamcorpus/`.

Kan11 Kan, M.-Y., Open source release - corpus, 2011. URL `http://www.comp.nus.edu.sg/entrepreneurship/innovation/osr/corpus/`.

KB14 Kingma, D. P. and Ba, J., Adam: A method for stochastic optimization. *ICLR*.

KFPH04 Kibriya, A. M., Frank, E., Pfahringer, B. and Holmes, G., Multinomial naive bayes for text categorization revisited. *Proceedings of Australasian Joint Conference on Artificial Intelligence*. Springer, 2004, pages 488–499.

KS97 Koller, D. and Sahami, M., Hierarchically classifying documents using very few words. Technical Report, Stanford InfoLab, 1997.

Kwo98 Kwok, J. T.-Y., Automated text categorization using support vector machine. *Proceedings of the International Conference on Neural Information Processing*. ICONIP, 1998.

LH17 Lota, L. N. and Hossain, B. M., A systematic literature review on sms spam detection techniques. *International Journal of Information Technology and Computer Science*, 9, pages 42–50.

LSTO10 Laboreiro, G., Sarmento, L., Teixeira, J. and Oliveira, E., Tokenizing micro-blogging messages using a text classification approach. *Proceedings of the 4th Workshop on Analytics for Noisy Unstructured Text Data*. ACM, 2010, pages 81–88.

MB90 Morgan, N. and Bourlard, H., Generalization and parameter estimation in feedforward nets: Some experiments. *Proceedings of Advances in Neural Information Processing Systems*. NeurIPS, 1990, pages 630–637.

MN+98 McCallum, A., Nigam, K. et al., A comparison of event models for naive bayes text classification. *Proceedings of AAAI Workshop on Learning for Text Categorization*, volume 752. AAAI, 1998, pages 41–48.

MY14 Mujtaba, G. and Yasin, M., Sms spam detection using simple message content features. *J. Basic Appl. Sci. Res*, 4,4(2014), pages 275–279.

NAAN14 Najadat, H., Abdulla, N., Abooraig, R. and Nawasrah, S., Mobile sms spam filtering based on mixing classifiers. *International Journal of Advanced Computing Research*, 1, pages 1–7.

NS13 Narayan, A. and Saxena, P., The curse of 140 characters: evaluating the efficacy of sms spam detection on android. *Proceedings of the 3rd ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. ACM, 2013, pages 33–42.

PM92 Pal, S. K. and Mitra, S., Multilayer perceptron, fuzzy sets, classification. *IEEE Transactions on Neural Networks*, 3, pages 683–697.

PY+10 Pan, S. J., Yang, Q. et al., A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22,10(2010), pages 1345–1359.

RH10 Resnik, P. and Hardisty, E., Gibbs sampling for the uninitiated. Technical Report, Maryland University, 2010.

RM51 Robbins, H. and Monro, S., A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Ros58 Rosenblatt, F., The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65,6(1958), page 386.

Rou18 Routledge, R., Bayes's theorem, 2018. URL `https://www.britannica.com/topic/Bayess-theorem`.

RSTK03 Rennie, J. D., Shih, L., Teevan, J. and Karger, D. R., Tackling the poor assumptions of naive bayes text classifiers. *Proceedings of the 20th International Conference on Machine Learning*. ICML, 2003, pages 616–623.

SBF98 Studer, R., Benjamins, V. R. and Fensel, D., Knowledge engineering: principles and methods. *Data & Knowledge Engineering*, 25,1-2(1998), pages 161–197.

Sch15 Schmidhuber, J., Deep learning in neural networks: An overview. *Neural networks*, 61, pages 85–117.

SDHH98 Sahami, M., Dumais, S., Heckerman, D. and Horvitz, E., A bayesian approach to filtering junk e-mail. *Proceedings of AAAI Workshop on Learning for Text Categorization*, volume 62. AAAI, 1998, pages 98–105.

Seb02 Sebastiani, F., Machine learning in automated text categorization. *ACM Computing Surveys (CSUR)*, 34,1(2002), pages 1–47.

SHK+14 Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15,1(2014), pages 1929–1958.

SHP95 Schütze, H., Hull, D. A. and Pedersen, J. O., A comparison of classifiers and document representations for the routing problem. *Proceedings of the 18th annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 1995, pages 229–237.

SM99 Scott, S. and Matwin, S., Feature engineering for text classification. *Proceedings of International Conference on Machine Learning*, volume 99. ICML, 1999, pages 379–388.

SM13 Shirani-Mehr, H., Sms spam detection using machine learning approach. Technical Report, Stanford University, 2013.

Smi Smith, M., More than 200 billion gsm text messages forecast for full year 2001. URL `https://web.archive.org/web/20020215194430/http://www.gsmworld.com/news/press_2001/press_releases_4.shtml`.

SR03 Silva, C. and Ribeiro, B., The importance of stop word removal on recall values in text categorization. *Proceedings of International Joint Conference on Neural Networks*, volume 3. IEEE, 2003, pages 1661–1666.

Tag09 Tagg, C., *A corpus linguistics study of SMS text messaging.* Ph.D. thesis, University of Birmingham, 2009.

TMR11 TMR, Global a2p sms market, 2011. URL `https://www.transparencymarketresearch.com/global-a2p-sms-market.html`.

UIKA11 Uemura, T., Ikeda, D., Kida, T. and Arimura, H., Unsupervised spam detection by document probability estimation with maximal overlap method. *Information and Media Technologies*, 6,1(2011), pages 231–240.

UM15 Urbanowicz, R. J. and Moore, J. H., Exstracs 2.0: description and evaluation of a scalable learning classifier system. *Evolutionary intelligence*, 8,2-3(2015), pages 89–116.

Una Unavailable, A., Grumbletext, uk consumer complaints. URL `http://www.grumbletext.com/`.

VD04 Valentini, G. and Dietterich, T. G., Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. *Journal of Machine Learning Research*, 5, pages 725–775.

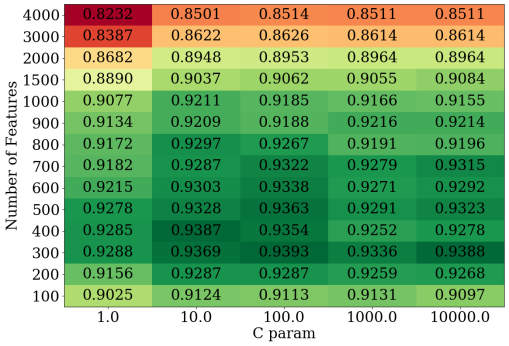
Wen95 Wendemuth, A., Learning the unlearnable. *Journal of Physics A: Mathematical and General*, 28,18(1995), page 5423.

WTS13 Warade, S. J., Tijare, P. A. and Sawalkar, S. N., Implementation of sms spam detection system. *International Journal of Research in Advent Technology*, 4,5(2013).

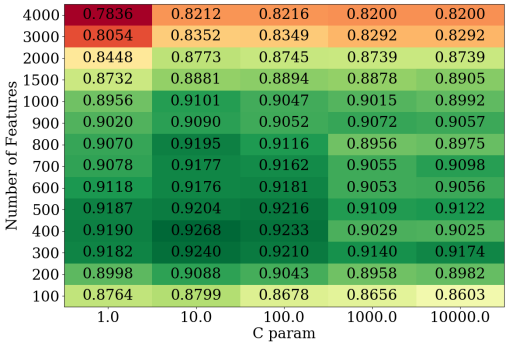
XXY+12 Xu, Q., Xiang, E. W., Yang, Q., Du, J. and Zhong, J., Sms spam detection using noncontent features. *IEEE Intelligent Systems*, 27,6(2012), pages 44–51.

YX08 Yu, B. and Xu, Z.-b., A comparative study for content-based dynamic spam classification using four machine learning algorithms. *Knowledge-Based Systems*, 21,4(2008), pages 355–362.

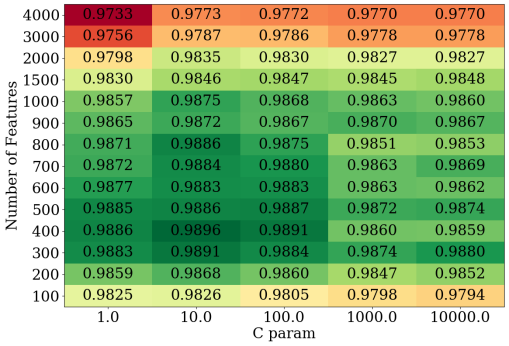
A Supporting Figures



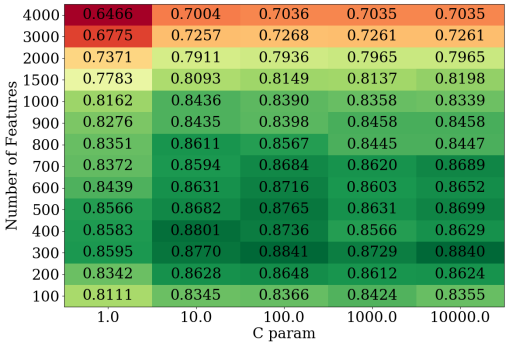
(a) Balanced accuracy



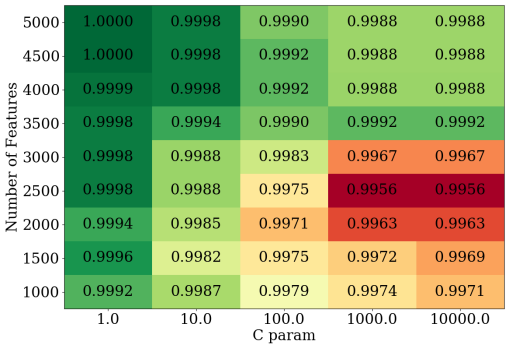
(b) Spam F1-score



(c) Ham F1-score

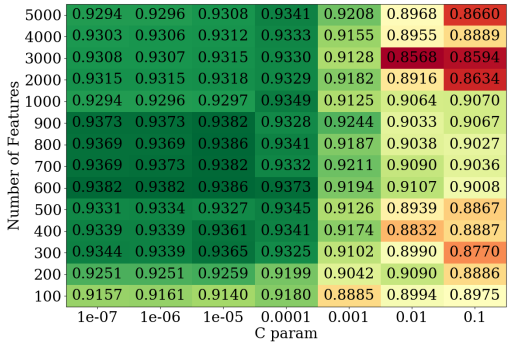


(d) Spam Recall

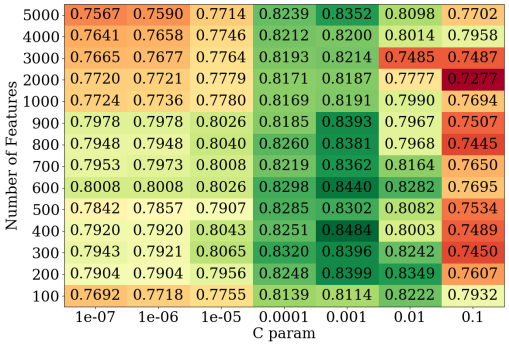


(e) Ham Recall

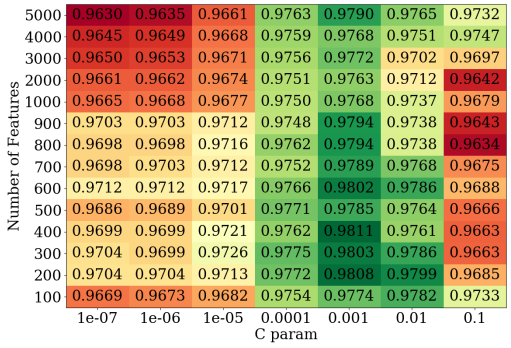
Figure 14: RBF SVM scores using the data set from Almeida et al.



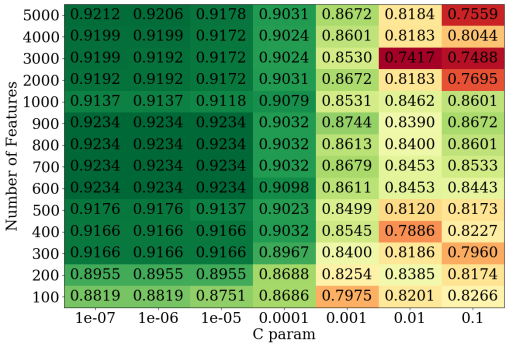
(a) Balanced accuracy



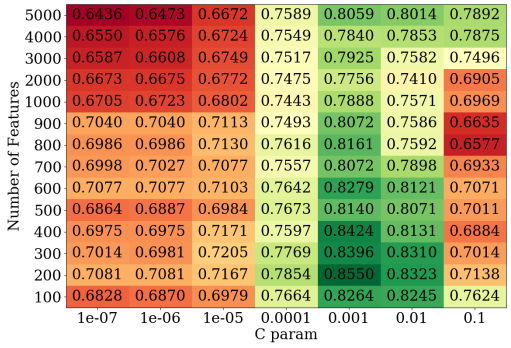
(b) Spam F1-score



(c) Ham F1-score



(d) Spam Recall



(e) Spam Precision

Figure 15: Linear SVM scores using the Veoo dataset.

Model	Spam Recall	Spam Precision
MNB (NF=3500)	0.9683	0.9531
TF-IDF MNB (NF=2000)	0.8730	0.9821
Linear SVM (NF=3000, C=0.0001)	0.9365	0.9219
LinearK SVM (NF=700, C=0.01)	0.9206	0.9508
Poly-2 SVM (NF=400, C=10.0)	0.8730	1.0000
Poly-3 SVM (NF=300, C=100.0)	0.8571	0.9643
RBF SVM (NF=400, C=10.0)	0.9365	0.9516
MLP (NF=2500, NN=1000-1000)	0.9408 ±0.0098	0.9851 ±0.0148

(a) Spam recall and precision scores

Model	Ham Recall	Ham Precision
MNB (NF=3500)	0.9942	0.9961
TF-IDF MNB (NF=2000)	0.9981	0.9846
Poly-2 SVM (NF=400, C=10.0)	1.0000	0.9922
Poly-3 SVM (NF=300, C=100.0)	0.9961	0.9903
Linear SVM (NF=3000, C=0.0001)	0.9903	0.9847
LinearK SVM (NF=400, C=0.01)	0.9883	0.9827
RBF SVM (NF=400, C=10.0)	0.9942	0.9922
MLP (NF=1500, NN=2000-2000)	0.9981 ±0.0002	0.9914 ±0.0015

(b) Ham recall and precision scores

Figure 16: Spam recall, spam precision, ham recall and ham precision scores for the models chosen by the best spam and ham F1-scores.