MASTER'S THESIS 2020

Classification of Short Text Messages Using Machine Learning

Alexander Goobar, Daniel Regefalk

ISSN 1650-2884
LU-CS-EX: 2020-47

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2020-47

Classification of Short Text Messages Using Machine Learning

Alexander Goobar, Daniel Regefalk

Classification of Short Text Messages Using Machine Learning

Alexander Goobar
ine15ago@student.lu.se

Daniel Regefalk
ine15dre@student.lu.se

August 10, 2020

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Pierre Nugues, pierre.nugues@cs.lth.se
Jianhua Cao, jianhua.cao@sinch.com
Michael Truong, michael.truong@sinch.com

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

In this master's thesis, we evaluate the classification performance of several machine learning models on very short texts, up to 160 characters in length. We evaluate both traditional machine learning algorithms and state-of-the-art deep learning models on binary, multi-class, and multi-label datasets. We also perform benchmarks to compare prediction times. The evaluation was performed on two public datasets and one dataset provided by Sinch Sweden AB, where this project was carried out. Sinch offers services for companies to send business-to-consumer SMS messages, and wants to investigate the possibility of using machine learning to automatically classify messages sent via their platform, to identify messages containing prohibited content. We also compare the machine learning models to Sinch's current solution for message blocking, which is based on regular expressions. Our results show that state-of-the-art deep learning models based on transformers, such as BERT, perform the best on the public datasets. However, some traditional algorithms, such as random forest and support vector machine, perform similarly to these models on the Sinch dataset. We also find that the machine learning models outperform Sinch's current solution for message blocking.

Keywords: natural language processing, machine learning, short text classification, SMS

Acknowledgements

We would like to thank Sinch Sweden AB for giving us the opportunity to carry out our thesis work there. Without access to their systems and data, this project would not have been possible. We would also like to thank our supervisors at Sinch, Michael Truong and Jianhua Cao, for their guidance and practical help. Finally, we would like to thank Pierre Nugues at the Department of Computer Science for his continuous supervision and guidance throughout this project.

Contents

Chapter 1

Introduction

In this thesis, carried out at Sinch Sweden AB, we evaluate different methods of text classification of short messages. Text classification is a task within machine learning, where categories are assigned to texts. A machine learning model is first trained to classify texts, by feeding the model input texts and desired output labels. The trained model can then be used on new data, where new texts serve as the input and the model then attempts to predict the most appropriate output label.

1.1 Problem Background

Sinch Sweden AB, hereinafter referred to as Sinch, provides services for companies to send SMS messages to consumers. A large quantity of their customers are based in the U.S., where there are certain regulations and guidelines regarding the content of SMS messages. Examples of regulated content are messages that fall within the following categories: **S**exual, **H**ateful, **A**lcohol, **F**irearms, **T**obacco (**SHAFT**), as well as controlled drugs, sweepstakes, and contests.

Sinch sends several million SMS messages per day, at a rate of approximately 1,000 messages per second. While many messages come from large trusted customers, Sinch also gets new customers every day, some of which act as an intermediary and are not the original sender.

To prevent their services from being used to send prohibited content, Sinch wants to be able to identify the messages in the categories above. In addition to blocking unwanted messages, they also want to be able to identify which category they belong to for data insight purposes.

1.2 Purpose

The purpose of this thesis is to evaluate different machine learning models and their suitability for classification of short texts. In addition to the general case, we investigate which models perform the best in the case of Sinch's data: business-to-consumer SMS messages. We also compare the current solution used by Sinch, based on regular expressions, to a potential solution based on a machine learning model.

1.3 Contributions

The contributions have been evenly distributed between the two authors. However, the main responsibilities for certain parts of the project have been split. D. Regefalk evaluated the traditional machine learning models and Flair, while A. Goobar had the responsibility for the evaluation of the transformer-based deep learning models. A. Goobar performed the annotation of data, dataset bootstrapping, and the experiment for comparison to regular expressions. D. Regefalk carried out the exploratory data analysis. The remaining parts of the project have had shared responsibility between the two authors.

Chapter 2
Background

In this chapter, we will introduce subjects and theory relevant to this project. Starting with a broader perspective, we describe natural language processing in general, and then narrowing the scope, ending with a short description and theory behind the models used. This chapter is intended to give the reader a high-level overview of the techniques and models used. For more in-depth explanations we refer the reader to the cited research papers.

2.1 Natural Language Processing

Natural language processing (NLP) is a field at the intersection of linguistics, computer science, and artificial intelligence that strives to process, interpret, and understand written and spoken human language. NLP is used in a large variety of contexts such as translation, transcription, summarization, language generation, classification, and more.

The epithet *natural* is meant to distinguish human language from more structured languages, such as computer code and mathematical notation, where the syntax follows a stricter ruleset and the vocabulary is much smaller. One of the fundamental challenges with natural language is that it does not always follow strict and well-defined rules, and it varies greatly by both context and language.

The first research in the field dates back to the late 1940s, and machine translation is considered to be one of the first computer-based NLP applications (Joseph et al., 2016). The initial approaches in the field were focused on manually creating grammar-based rules in an attempt to understand natural language. Starting with the *statistical revolution* (Johnson, 2009) in the late 1980s, the focus instead moved to utilizing machine learning to implicitly learn these rule sets from training data. For decades, the machine learning approaches focused on simpler models such as logistic regression and support vector machines. With the rise and success of deep learning in related fields over the past few years, the NLP field is increasingly focusing on deep learning and has made impressive advances.

2.1.1 Working with Text

To utilize mathematical models to analyze and understand natural language, the input text must be converted to some numerical representation. There have been numerous approaches for this, from simple one-hot encoding to advanced pre-trained embeddings that take context into account. There are also various methods for determining which words in a sentence are the most important, and which can be discarded. The following sections will explore some of these challenges in more depth.

Embeddings. A token refers to a string of characters that form a meaningful unit, typically a word. A naive approach of converting a token to a numerical representation is one-hot encoding. In this case, each token is represented by a vector of the same size as the vocabulary, with the vector containing only zeroes except one element with the value 1, representing the token. However, using pre-trained embeddings is generally much more effective. Common pre-trained word embeddings, such as *word2vec* and *GloVe*, are instead pre-trained on large text corpora and produce vector representations where similar words (e.g. *cat* and *lion*) have similar numerical representations (Mikolov et al., 2013; Pennington et al., 2014). There are also more advanced embedding approaches that embed homographs (different words that are spelled the same way) differently based on context (Akbik et al., 2018).

TF-IDF. *Term frequency-inverse document frequency* is a metric used to reflect the importance of a token in a document. The TF-IDF value for a token is proportional to the number of occurrences in the document divided by the total number of documents that contain the token in the full corpus. By increasing the weight of less frequent and thus more specific tokens, a machine learning model can better separate signal from noise. The use of TF-IDF scoring shows considerable improvements in many natural language processing tasks (Rajaraman and Ullman, 2011; Jones, 1972). Another common approach to improve the signal-to-noise ratio is to simply remove common words that do not contribute much to a sentence's meaning such as *and*, *the*, and *of*. These words are commonly referred to as *stopwords*. By using TF-IDF these words' weights are usually reduced automatically.

2.2 Traditional Machine Learning

Machine learning refers to the process of a computer learning from data. While the concept of machine learning is not new, it has grown rapidly in popularity over the past decade. Deep learning is a subfield of machine learning, and as such a solid foundation in traditional machine learning is important to understand deep learning.

Machine learning models excel in tasks that are difficult to explain as a set of fixed rules. One such task is text classification. While we humans can classify text easily, it is very hard to explicitly create a set of rules that enables a computer to do the same. This is one of the key challenges machine learning was developed to solve. By instead providing a machine learning model a large dataset of labeled messages, the machine learning model can implicitly learn these rules without the need of an explicitly defined ruleset. For further explanations, see *Machine Learning* by Mitchell (1997) and *Deep Learning* by Goodfellow et al. (2016).

In this project, we utilize four traditional machine learning classifiers, namely random forests, support vector machines, naive Bayes, and logistic regression.

Random Forests. Random forest classifiers consist of a large number of individual decision trees, each of which outputs a class prediction based on the input features. By utilizing a large number of diverse, uncorrelated decision trees, the classifier can create predictions that are more reliable than each individual decision tree (Breiman, 2001).

Support Vector Machines. Support vector machines (SVM) are non-probabilistic binary classifiers. SVMs can be used to perform either linear or non-linear classification. For non-linear classification the input features are non-linearly transformed into high-dimensional space. The support vector machine then finds the hyperplane that best separates the classes in the training dataset. By non-linearly transforming the features of the dataset the model's generalization ability can be improved (Cortes and Vapnik, 1995).

Naive Bayes. Naive Bayes classifiers are probabilistic classifiers based on Bayes theorem with strong assumptions regarding independence between features. The probability of a document belonging to a specific class is computed based on the tokens contained in that document. Tokens that commonly occur in documents of that specific class in the training dataset increase the likelihood of classifying the document as belonging to that class (Manning et al., 2008).

Logistic Regression. Logistic regression is the baseline supervised machine learning algorithm for classification in NLP and is closely related to neural networks. Logistic regression is, like naive Bayes, a probabilistic classifier based on supervised learning. The algorithm fits a logistic function to the training data which can then be used to output a probability of a sample corresponding to each class. By introducing a threshold value, such as 0.5, the probability is converted to a binary classification (Jurafsky and Martin, 2000).

2.3 Deep Learning

Traditional machine learning models perform well on many problems in a wide variety of domains. Due to their relatively simple architecture, they do however struggle with more complex AI tasks such as image and speech recognition. Working with complex and high-dimensional data, these traditional models struggle to generalize well to new examples, and the difficulty and computational cost increases exponentially with increased complexity. This shortcoming of the traditional methods motivated the development of deep learning.

Deep learning is a subfield of machine learning, where the models are built as sequences of separate layers. Using this architecture, complex tasks can be broken down into simpler, more manageable sub-tasks at each layer. This structure of multiple sequential layers has given name to the term *Deep Learning*.

While traditional machine learning still depends largely on features manually extracted from the dataset, deep learning enables features to be automatically extracted. This both makes the process less time-consuming by avoiding task-specific feature engineering, and avoids the use of manually extracted features which might be incomplete (Young et al., 2017).

Deep learning has produced state-of-the-art results in many domains, and these models have shown great results in the context of NLP (Young et al., 2017). Even simple deep learning approaches have been shown to outperform the best performing traditional machine learning methods on various NLP tasks such as part-of-speech tagging and named-entity recognition (Collobert et al., 2011).

It is worth noting that deep learning typically requires vast amounts of training data to be able to implicitly learn the rules needed at each layer to perform the task, whereas traditional machine learning can perform better if the training dataset is limited in size. The next section addresses some methods for overcoming these problems.

In this project, we utilize four deep learning classifiers, namely Flair, BERT, DistilBERT, and XLNet.

2.3.1 Transfer Learning

Transfer learning is a method within machine learning, where the knowledge that has been acquired by solving a certain problem is used to help solve another problem. In practical terms, it often means that a model that has been trained on a particular dataset is used as a starting point for a model on another dataset.

In the natural language processing field, transfer learning has seen significant progress over the past three years. The introduction of fine-tuning approaches, similar to what had previously been done in computer vision, is one of the main drivers of this trend. For this approach, a base model is first pre-trained and later fine-tuned for the target task. The base model is typically very large, with over 100 million parameters, and takes significant resources to train. However, the fine-tuning is usually a quick task, and is typically done by adding a single layer (Usherwood and Smit, 2019).

Domain adaptation. Domain adaptation refers to the process of adapting a model trained on one task to a new, similar task. In NLP, this can refer to taking a pre-trained general language model and then fine-tuning it for some specific task. It can also be used to take a model that has been trained on the same task, but in another context. There has been much work in the field of spam detection, and by utilizing these models on a new dataset much of the groundwork can be avoided.

2.3.2 Recurrent Neural Networks

Recurrent neural networks (RNN) is a deep learning network architecture used to model temporal relationships in data. This is particularly useful in the NLP domain as previous words and sentences are crucial for understanding the context of following words and sentences. In recurrent neural networks, some output from the network is fed back as input to the network in next time-step, creating a recurrent connection that enables the network to model relationships over time.

A popular RNN architecture used in the NLP domain is *long short-term memory*. The LSTM architecture, initially introduced in 1997, is an efficient architecture for modelling temporal relationships over extended time periods. One key improvement over previous

RNN architectures is the concept of *constant error flow*, which helps prevent the vanishing/exploding gradient problem inherent in training recurrent neural networks through backpropagation. The LSTM architecture is also able to truncate the error gradient where appropriate, making the training process much more efficient while not considerably lowering prediction performance (Hochreiter and Schmidhuber, 1997).

2.3.3 Flair

Flair is an NLP library originally developed by Zalando Research in 2018, and it achieved state-of-the-art results on numerous NLP tasks when it was released. The main idea behind the framework was to provide a simple, unified interface for conceptually different types of word and document embeddings (Akbik et al., 2019). As such, much of the embedding-specific engineering complexity is hidden and the user can test and use different embeddings easily. Flair is based on a "vanilla bidirectional LSTM" architecture (Akbik et al., 2018), indicating that the state-of-the-art performance is achieved mainly through efficient embeddings rather than advanced network architecture.

One of the novel improvements with Flair is the notion of *contextual string embeddings*. These embeddings have the advantage that the same word or token will be embedded differently based on its surrounding context. This is achieved utilizing a bidirectional LSTM architecture, where one LSTM network is looking at the context before the token and one LSTM network is looking at the context after the token (Akbik et al., 2018). This is in contrast to e.g. word2vec which always produces the same representation for a given word, regardless of its context. Flair also enables the use of *stacked embeddings*. By stacking different embedding types, such as contextual string embeddings and traditional word2vec embeddings, the unique strengths of each approach can be combined to achieve state-of-the-art results on various NLP tasks (Akbik et al., 2018).

Another improvement is how Flair tokenizes the input text. Instead of splitting text at word boundaries, the Flair embeddings were trained without an explicit notion of words and as such they model text as a sequence of characters, instead of as a sequence of words. This both helps reduce the likelihood of out-of-vocabulary words, and can improve the correct detection of misspellings (Akbik et al., 2018).

2.3.4 Transformer

The transformer, introduced in 2017, is a deep learning model that laid the foundation for several modern NLP architectures. Like RNNs, they model temporal relationships in data. Unlike recurrent neural networks however, the sequences do not need to be processed in order when using transformers, which allows for more parallelization during training. This facilitates the use of larger datasets for training the model. As a result, systems pretrained on huge corpora have been developed based on the transformer model (Vaswani et al., 2017).

Another improvement of transformers over RNNs is that they have been shown to better model long term dependencies. In theory, RNNs are also capable of this, but in practice they have been shown to struggle in this regard (Bengio et al., 1994).

Transformers are based entirely on an attention mechanism, and as such do not require recurrent or convolutional neural network architectures. Experiments on machine trans-

lation tasks have shown that this approach produces superior results while also requiring significantly less time to train (Vaswani et al., 2017).

2.3.5 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a language representation model introduced by Google in 2018. The model architecture is a multi-layer bidirectional Transformer encoder. It has been pre-trained on a huge training corpus, consisting of the BooksCorpus with 800M words and English Wikipedia with 2500M words (Devlin et al., 2019).

BERT is able to represent a word-based on both previous and upcoming context, hence the "bidirectional" notation. To achieve this, two strategies have been used during pre-training. The first one is the masked language model, where some words are masked in the pre-train data and the model tries to predict the missing words by looking at the context before and after. In practice, this means that 15% of the words are replaced with a `[MASK]` token during pre-training (Devlin et al., 2019).

The second strategy used during pre-training is next sentence prediction, where the model learns how sentences relate to each other. During the training, the model is presented with a pair of sentences and then has to predict whether the second sentence comes after the first sentence in the original document or not (Devlin et al., 2019).

Pre-training the model is very resource-intensive, but since this has already been done and Google has published the model weights, anyone can use the pre-trained model and then fine-tune it for a specific task. During fine-tuning, typically only one additional output layer has to be added, which is significantly less resource-intensive to train (Devlin et al., 2019).

When pre-processing data for training and prediction using BERT, the messages have to converted to the format expected by the model. The beginning of every input should be a `[CLS]` token, and sentences are separated using a `[SEP]` token, as seen in Figure 2.1. BERT uses WordPiece embeddings, with a 30,000 token vocabulary, as token embeddings (Devlin et al., 2019). WordPiece embeddings improve the handling of rare words by dividing them into a limited set of common sub-words. This method provides a good balance between the efficiency of word-delimited models and the flexibility of character-delimited models, improving the overall performance of the network. This process also removes the possibility of out-of-vocabulary words (Wu et al., 2016).

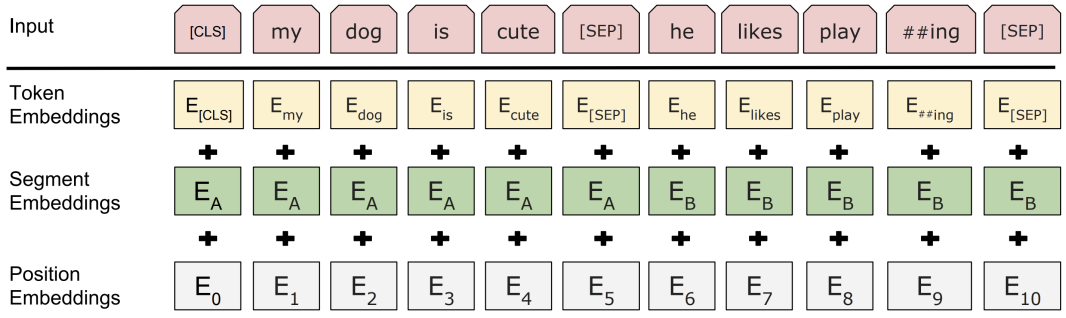


Figure 2.1: Representation of BERT input. After Devlin et al. (2019)

In addition to token embeddings, segment embeddings indicate which sentence the word belongs to and position embeddings indicate its position in the sequence. For every token, the input representation is constructed by summing the corresponding three embeddings, as seen in Figure 2.1 (Devlin et al., 2019).

2.3.6 DistilBERT

As previously mentioned, BERT is very resource-intensive to train and use, especially when compared to simpler machine learning models. To reduce the resource usage, the Hugging Face team introduced the DistilBERT model, a distilled version of BERT. To achieve this, the authors used knowledge distillation. This is a compression technique where a smaller model is trained to reproduce the results and behaviour of a large model. DistilBERT is 40% smaller and 60% faster than BERT, but still retains 97% of the performance (Sanh et al., 2019).

2.3.7 XLNet

XLNet is another model based on the transformer approach, more specifically TransformerXL. The creators of XLNet argue that using `[MASK]` tokens during pre-training create a discrepancy between pre-training and fine-tuning, since masks are not present in the real data during the fine-tuning process (Yang et al., 2019).

To capture bidirectional context, XLNet uses permutation language modeling, where the idea is to train the model on all possible permutations of words in a sentence. Instead of fixed right-left or left-right modeling, XLNet works by maximizing the expected log likelihood over all different permutations of a sequence. The authors claim that XLNet can capture more long-term dependencies compared to BERT, using this approach. In the article introducing XLNet, the model is shown to outperform BERT on 20 tasks, including sentiment analysis (Yang et al., 2019).

Chapter 3
Related work

Text classification is a common task within natural language processing, which has led to a large quantity of research in this field. Approaches for classification of very short texts have previously been investigated. However, the datasets for short text classification usually come from online comments, tweets, or reviews. There is very limited previous work on SMS messages, specifically in the context of business-to-consumer communication.

3.1 Traditional vs. Deep Transfer Learning

The comparison of traditional machine learning algorithms against newer models that utilize deep transfer learning was performed in the paper *Low-Shot Classification: A Comparison of Classical and Deep Transfer Machine Learning Approaches* (Usherwood and Smit, 2019). In this paper, the authors investigate how different models perform, more specifically in the context of low-shot classification, meaning that one or more classes have a significantly low sample count. The traditional algorithms used were support vector machines and naive Bayes, which were compared to the transfer learning models ULMFiT and BERT. They reached conclusive results, where BERT performed the best in every case.

3.2 Datasets

The public datasets used in this project, AG News and Wikipedia Toxic Challenge, have been part of previous research. However, in contrast to our work, the datasets have been used in their original state without filtering by text length. The paper that introduced the XLNet model, *XLNet: Generalized Autoregressive Pretraining for Language Understanding*, used the AG News dataset to benchmark the model, achieving state-of-the-art results for this particular dataset (Yang et al., 2019).

The dataset from the Wikipedia Toxic Challenge was used in the paper *Towards non-toxic landscapes: Automatic toxic comment detection using DNN* (D'Sa et al., 2019). In this paper, the authors compared how multiple deep learning models, such as BiLSTM and CNN, compared to BERT in a binary toxic vs non-toxic classification task. They also used the models with different word embeddings. Using BERT embeddings yielded the best results for the smaller deep learning models, but the overall best performing option was BERT fine-tuning.

Chapter 4
Data

The foundation for all machine learning is the dataset used to train the model. This project has utilized three main datasets: AG News, Wikipedia Toxic Comments, and Sinch's messages.

First AG News and Wikipedia Toxic Comments, two public and pre-annotated datasets, were used to evaluate different models and approaches. However, these were modified to better represent the challenges of short text messages and limited annotated dataset size, as described below. Finally, Sinch's message data was manually annotated and used to train the final models.

4.1 AG News

The original AG News dataset is a collection of more than 400,000 categorized articles, collected from over 2000 news sources. A modified version of this dataset contains 120,000 articles from the four largest categories: world, sports, business, and sci/tech. Each category has 30,000 messages. We used the modified version for this project.

To make the dataset more suitable for a comparison with Sinch's actual dataset, we only included the news articles with a length of up to 160 characters (the limit of a single SMS message). We finally sampled 1,250 articles of each category to get a final dataset of 5,000 articles. This way, the property of being a completely balanced dataset, with an equal number of samples of each class, was preserved. 5,000 was chosen as we expected the final annotated Sinch dataset to be of approximately this size.

4.2 Wikipedia Toxic Comments

Toxic Comment Classification Challenge is a Kaggle challenge launched at the end of 2017. The dataset contains 159,571 manually annotated comments from Wikipedia, each of which is

annotated according to six categories: *toxic*, *severe toxic*, *obscene*, *threat*, *insult*, and *identity hate*. The dataset is multi-label, meaning that each comment belongs to zero or more of the above categories. (Kaggle, 2017)

Just as with the AG news dataset, we removed all comments with a character count higher than 160, and then randomly sampled 5000 comments to be used to train the models. The category distribution of the sampled comments is shown in Figure 4.1. This imbalanced distribution is relatively similar to that of Sinch's messages and as such served as a good benchmark for multi-label classification.

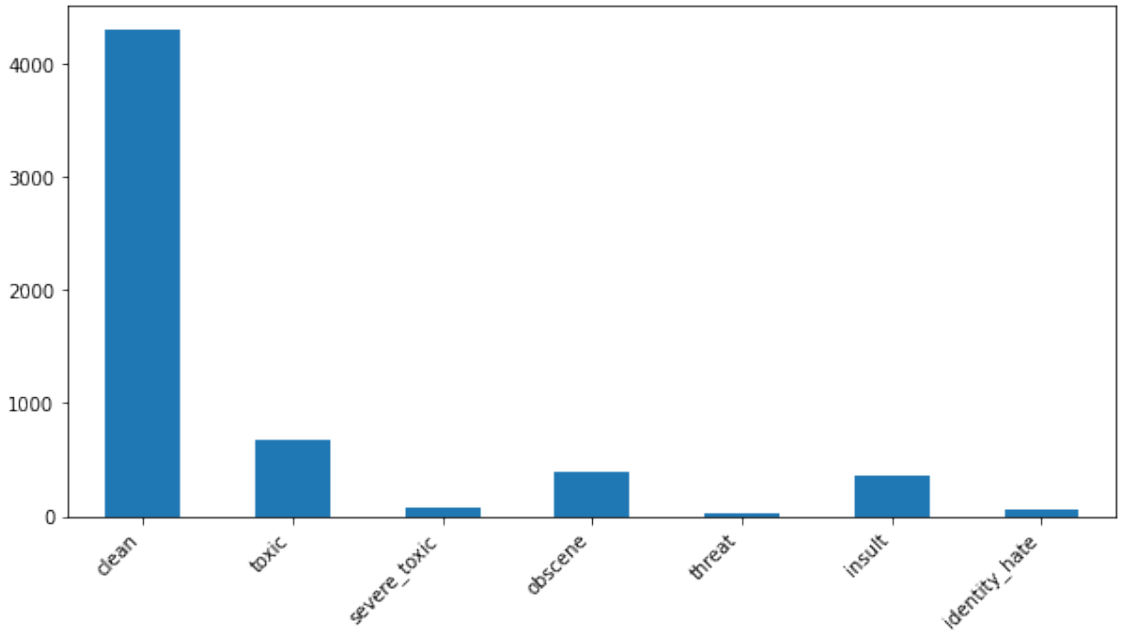


Figure 4.1: Class distribution of comments with a max length of 160 characters from the Wikipedia Toxic Comment dataset.

4.3 Sinch Data

The data from Sinch was extracted from their logs of sent SMS messages in the U.S. This data was only accessible via equipment provided by Sinch. Each data file contained messages from a specific time interval, and was provided in a CSV format. Each row had a total of four columns, separated by a semicolon, which contained:

- How many times this exact message was sent within the logged time interval
- The sender name
- The sender short code (number)
- The message's text content

For this project, we decided only to factor in the message text content as input to our models. Sinch initially provided a set of logs that was used in the starting phase of the

project. Later on during the project, we were provided with two additional sets of messages, containing more data. Sinch also provided logs of messages that were flagged as containing prohibited content by the current filter, based on regular expression.

As SMS messages are limited to 160 characters, this was the maximum length of the text content for each row. Messages that exceed this limit are split into multiple rows, as they are technically sent out as multiple SMS messages. However, since the data was not organized in chronological order, there was no way to efficiently identifying split messages and merging them, so each row was handled as a standalone message even though this is not always the case.

4.3.1 Exploratory Data Analysis

To get an overview of the dataset, we analyzed both the raw and annotated Sinch dataset. By exploring the characteristics of the dataset, we could later make informed decisions regarding feature selection, model selection, and model parameters.

Raw Data. As shown in Figure 4.2, the word count has two peaks at 12 words and 28 words respectively. The peak at 12 words was examined and was found to be due to many of the messages being verification codes and similar messages that often are in that word count range.

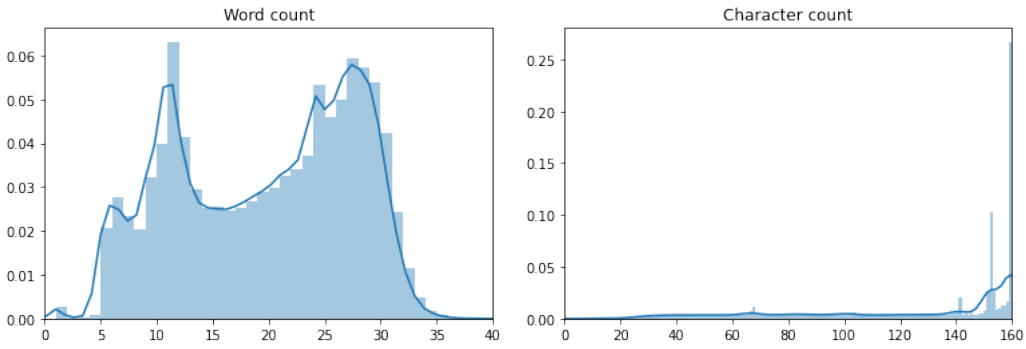


Figure 4.2: Word count and character count distribution of the raw data.

The character count graph shows that over 25% of the messages are exactly 160 characters, indicating that a large portion of the raw messages have been cut off. As previously mentioned, there was no good way to identify split messages given the raw data we were provided.

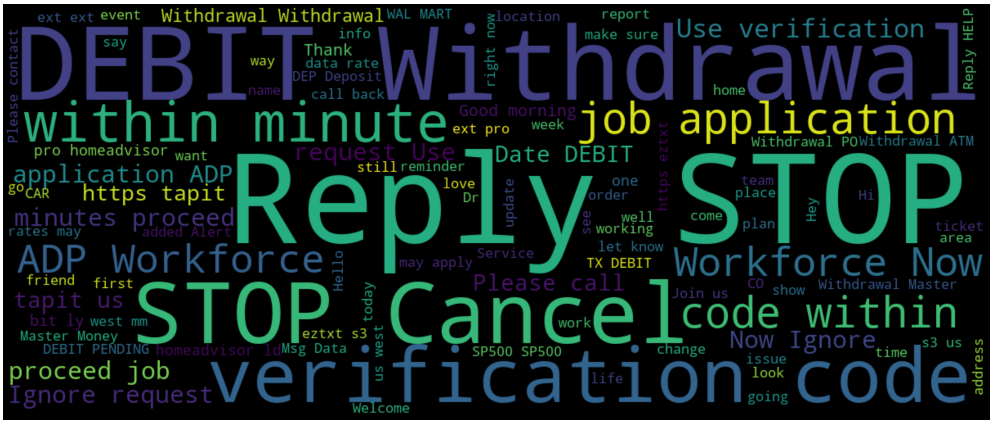


Figure 4.3: Word cloud of the 100 most common words and bigrams (a few sensitive words/bigrams have been removed).

The word cloud in Figure 4.3 gives a good high-level overview of the type of language being used in the messages. As can be seen the words used are typical of business-to-consumer SMS messages, and give a hint that using domain adaptation, e.g. utilizing prediction models trained on other datasets, might be problematic as the language used is very domain-specific.

Annotated Data. One key challenge with any classification task utilizing machine learning is getting enough training data for each class. For this dataset we manually annotated 4,862 of Sinch's messages and used these to train and evaluate our models. Our process for finding and annotating this data is explained in detail in Section 5.3 (Annotation) and 5.4 (Finding Data and Bootstrapping).

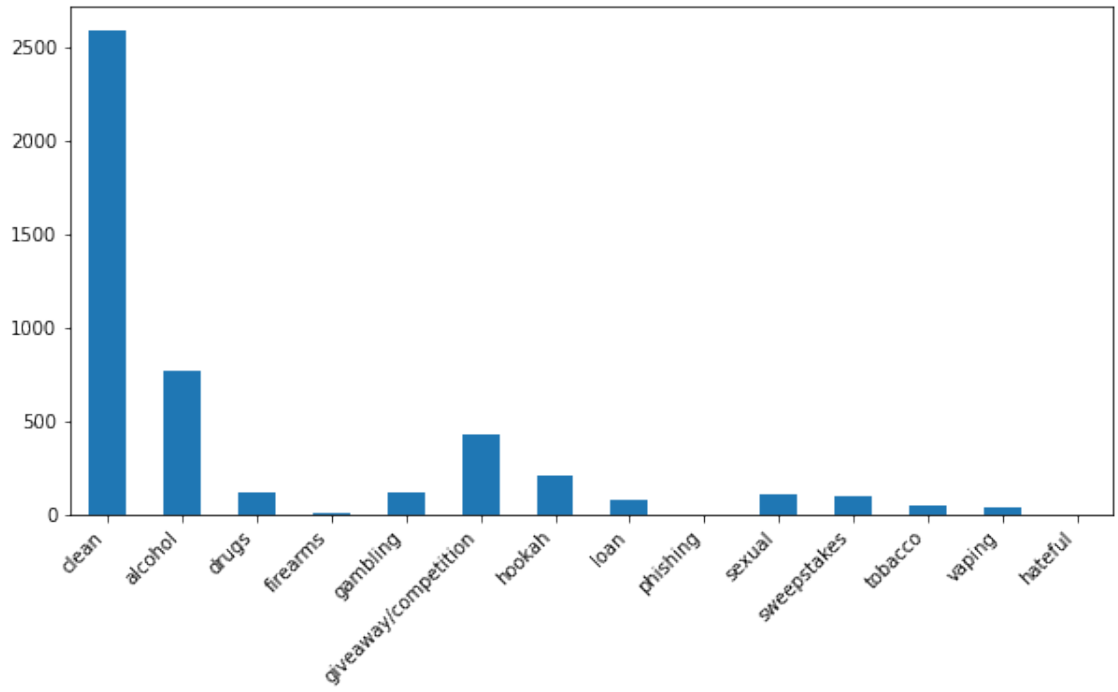


Figure 4.4: Class distribution of the annotated messages.

The class distribution in the annotated dataset is shown in Figure 4.4. As can be seen some categories, such as *alcohol* and *giveaway/competition*, have hundreds of examples. *Firearms* on the other hand has 12. This gives a clear indication that some categories will be very challenging for any machine learning model to correctly classify. Furthermore, we were not able to find any examples of *phishing* and *hateful* messages in the 4,862 labeled messages.

To better understand the class distributions, we also looked into the co-occurrence of classes as shown in Figure 4.5. For all messages of each of the classes on the ***Y***-axis, we calculated the fraction of those messages that also contained each of the classes on the ***X***-axis. This shows that for example out of all messages of class *hookah*, 63% of them also belong to the class *alcohol*. For the reverse, messages of class *alcohol* also belonging to the class *hookah*, the number is instead 17%. This graph helped us reason about how different classes might be combined to reduce the number of classes in an effort to simplify both the annotation and classification process.

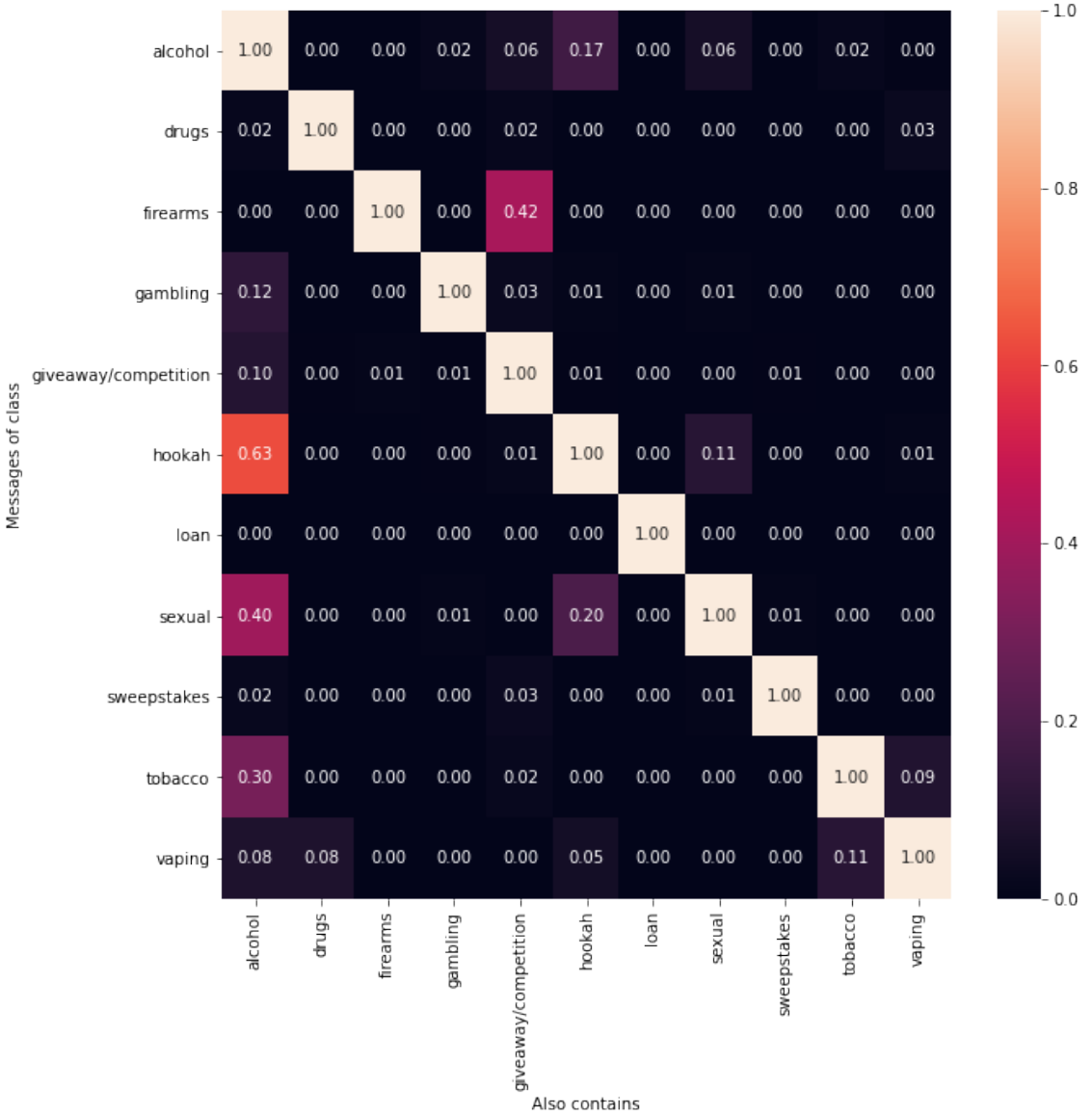


Figure 4.5: Heatmap showing co-occurrence of classes.

As one of the key use cases for Sinch is to classify a message as either *clean* or *dirty*, we also examined this binary distribution as shown in Figure 4.6. It is interesting to note that a large part of the messages we annotated were extracted from a RegEx currently used by Sinch to find *dirty* messages. As our annotated dataset consisted of 61% *clean* and 39% *dirty* messages, this indicates that the current solution captures many false positives. The true positives and false positives of the RegEx is further examined in Section 6.3.1.

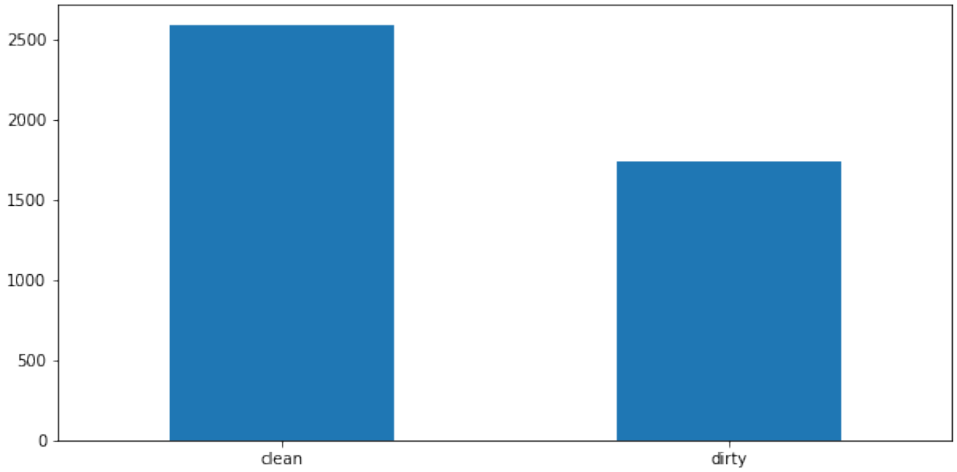


Figure 4.6: Binary class distribution of the annotated messages.

To get a better understanding of the language difference between the categories, we also generated the most correlated N-gram ($N = 1, 2, 3$) for each class. It should be noted that often the most correlated trigrams and bigrams were the name of the organization sending the SMS, e.g. the name of a bar in the *alcohol* category. A few examples of bigrams are shown in Table 4.1. Finally, we generated word clouds for each class to give a high-level overview of the language used, see Figure 4.7 for examples.

Giveaway/Competition	Tobacco	Sexual
enter win	cigar lodge	free entry
entered win	happy hour	grown sexy
chance win	cigar event	til 12

Table 4.1: Most correlated bigrams per class, a few examples.

Figure 4.7: Word clouds of the 100 most common words and bigrams, for *clean* and *alcohol* respectively.

Chapter 5
Methodology

In this chapter, we describe the methodology used for this project. The project has required multiple steps and processes, especially in regard to obtaining and handling data. Some of the processes have been performed in parallel to others, while some required the other processes to be completed beforehand. Initially we explored public datasets, with the objective to find pre-annotated classification datasets with language characteristics similar to that of Sinch's messages. We then set up testing pipelines for each model, and benchmarked them on the public datasets.

After this, we started to work with the Sinch dataset. Initially we explored different strategies to efficiently annotate the messages given our limited time frame. Our initial approach was based on the logs from Sinch's regular expression filter. With this initial data to work with, we transitioned into using bootstrapping to find further messages to annotate. The annotated dataset was continuously expanded, until we reached a point where the decision of setting a dataset baseline was made. This decision was based on the size of the annotated dataset, the benchmarked performance, and the remaining time of the project.

5.1 Pre-processing

For the AG News and Wikipedia Toxic Comments dataset, the only pre-processing step was to remove any entries that exceeded 160 characters in length.

By the nature of Sinch's customers' sending behaviour, a vast majority of the raw messages are duplicates. One typical example is verification codes. These duplicates were removed during pre-processing, with the exception of a few cases explained in Section 5.4. The removal was performed by comparing each message from the same sender. The filtering was based on the number of words shared between two messages, and the word order. It is worth noting that while this process was efficient and had good results, it was not perfect. A good example is that messages that all end with "*Reply STOP to stop receiving updates. Message rates may apply.*" might be considered duplicates even though they are unique. As this was a relatively rare

occurrence, we deemed this filtering approach to work well enough for our use case. Table 5.1 shows the reduction amount.

	Reduction
Logs A (1st set)	*98.8%
Logs B (2nd set)	99.5%
Logs C (3rd set)	99.7%
RegEx filter log	*95.9%

Table 5.1

*Not all duplicates were removed, as explained in Section 5.4.

5.2 Exploration

To get an overview of the raw data, we calculated the character counts, word counts, URL counts, average word lengths, capitalization statistics and more. Since all messages are short and relatively similar, most of the statistics were not very insightful. In the end, character counts and word counts were the two features that we found most relevant. We used a word cloud of the most commonly used words to help get an overview of the text content while not revealing any sensitive information.

For the annotated dataset, we calculated the class distributions, both for each individual class, and also after combining the class labels into a binary representation of either *clean* or *dirty*. For each class, we also calculated the most correlated N-gram ($N = 1, 2, 3$) and generated a word cloud.

5.3 Annotation

The public datasets, AG News and Wikipedia Toxic Comments, already had labels and therefore did not require any additional annotation. The first step of annotation of the Sinch dataset was to decide which labels to use. We did this after conversations with representatives at Sinch. They wanted to be able to identify messages according to the SHAFT guidelines, as well as some additional categories. After this discussion and getting an overview of the dataset, we decided to use the following labels during annotation:

Sexual	Hateful	Alcohol
Firearms	Tobacco	Drugs
Sweepstakes	Giveaway/competition	Gambling
Hookah	Vaping	Phishing
Loan		

We also had a *clean* label for any message that did not belong to any of the above categories, as well as an *uncertain* label. More specifically, messages promoting or attempting to sell items or services within these categories, or having a positive sentiment towards them, were classified within the appropriate category. However, messages that condemned or discouraged the use of items or services within the categories were tagged as *clean*. For example, advice to stop smoking was not categorized as *tobacco*, but as *clean*.

The *uncertain* label was used when the annotator did not feel confident in the label assignment for the particular message. It was also used when messages in other languages than English had not been successfully filtered out before being imported to the annotation tool.

As an annotation tool we used Doccano, which provides a web interface where a message is presented and the annotator can choose appropriate labels (Nakayama et al., 2018). Since this was a multi-label task, it was possible to choose multiple labels for the same message.

When exporting the data from the tool, it produced a JSON file with the messages and annotation data. We wrote a script to produce CSV files with the desired format. For multi-label, we used one-hot encoding. We also produced files for binary classification using this data, where the *clean* category remained, but all the other categories were merged to a *dirty* category.

We decided to only use one annotator to prioritize dataset size during the limited time of the project. Using multiple annotators, where every annotator works with different subsets of the dataset, would require roughly the same amount of time but could introduce inconsistencies as the interpretation between humans often vary for some messages.

On the other hand, using multiple annotators where everyone annotates the entire dataset is a better method to achieve high annotation quality, as any inconsistencies between annotators can be found and reconsidered. This approach would however require more man-hours for the same dataset size. Since we are not domain experts and the SHAFT guidelines are not clearly manifested, the annotator used the *uncertain* label frequently in an attempt to avoid individual interpretation errors for uncertain cases. These messages, which made up 24% of the total annotated messages, were not included in our final dataset and were instead left for future review.

5.4 Finding Data and Bootstrapping

Messages within the “SHAFT” or other categories listed previously are very rare among all the messages that Sinch’s customers send. Simply sampling and annotating the regular log data would therefore be a very inefficient way of gaining samples in the categories of interest. Instead, we used the log data from their current method of identifying unwanted messages, which is based on regular expressions, as a starting point. The regular expression script uses blacklisted keywords along with a blacklist and whitelist of senders, to determine if a message is unwanted or not. It does not provide any categorization, just a simple binary *dirty*/*clean* output.

After the annotation of the pre-processed RegEx logs, we had 3,117 labeled messages, where 2,198 did not have the *uncertain* label and would therefore be used for the first experimentation with models. The *uncertain* messages required closer inspection by a domain expert, and was consequently left for further work.

To expand our labeled dataset, we used bootstrapping after the initial annotation of

RegEx data. Bootstrapping in this context means that we use the current labeled data to train a model that we use to predict on new data. We used these predictions to find messages that are likely to be in our wanted categories. These messages were then annotated to expand our dataset (Cui et al., 2016).

To perform this kind of dataset bootstrapping, we used DistilBERT, for a number of reasons. First of all, it had shown good results in our early testing. We also hoped that utilizing the initial knowledge transformers models have from pre-training could lead to it finding a more diverse set of messages. Due to limited access to GPU resources during the time of bootstrapping, the larger models could not be used. We also saw DistilBERT as potentially a good middle ground between model size and the potential benefits of transfer learning.

We trained a DistilBERT binary classifier on the first version of the labeled RegEx data and then predicted the first version of the full logs (Logs A). 1,288 messages were predicted as *dirty*. These messages were then manually reviewed and annotated, resulting in a total dataset size of 3,333 messages after the uncertain entries were removed.

We later received another set of logs, Logs B, which had much more unlabeled data. Again, we used bootstrapping to expand our labeled dataset, but this time we set the threshold probability to 0.2 for a *dirty* message, instead of the previously used standard value of 0.5. In addition to expanding the dataset, we could also possibly reduce the number of false negatives using this data, as some of the messages in the 0.2-0.5 range could be falsely predicted to be clean with the standard threshold. We chose the threshold of 0.2 to get a good chance of finding false negatives without requiring a huge amount of time for annotation, as a large majority of the predictions were below 0.2. Out of all messages in Logs B, roughly 2% had a predicted dirty probability of over 0.2. We then reviewed and annotated the messages above this threshold, which resulted in a total dataset size of 4,638 messages after removing uncertain entries.

After receiving the final set of logs, Logs C, we created our own script to find messages containing certain keywords. The script targeted categories that currently had relatively few samples, and flagged a total of 436 messages. These messages were then manually annotated. The dataset size was now 5,029 messages, excluding uncertain entries.

At this point, we removed remaining exact duplicates that had been accidentally included in the input to the annotation tool. Because the logs consisted of multiple files, and we only initially removed duplicates on a per-file basis, some duplicates had been included when the same message occurred in multiple files (this problem only affected Logs A and RegEx filter logs). We also annotated some more data for a specific type of messages, news alerts, that had been generating many false positives during testing of a DistilBERT binary predictor.

After the removal of duplicates and the addition of some news alert messages, the dataset now contained 4,523 messages. This data was used in the comparison to regular expressions in Section 5.8. Since the comparison required us to further annotate some messages, we decided to include these in the final dataset used during training and benchmarking of the models. Since the comparison resulted in 339 additional annotated messages, the final dataset size became 4,862 messages without uncertain entries.

5.5 Merging and Exclusion of Categories

With the final dataset completed, we had no messages in the *hateful* and *phishing* categories and only 12 in the *firearms* category. For this reason, we decided to exclude these categories when evaluating the models and leave expansion of these categories as further work.

We also decided to merge some categories due to limited amounts of messages in some categories, and similarity between them. Tobacco, vaping, and hookah were all merged to a *TVH* category, as they all technically fall under the tobacco classification of SHAFT. We also merged sweepstakes, giveaways/competition and gambling into an *SGG* category. Sweepstakes are a form of giveaway and the sweepstakes label was only used when the message specifically mentioned sweepstakes or had the word *sweeps* or similar in a provided URL, as the annotator could not otherwise separate the two. Since a message can be ambiguous whether or not it required some form of paid entry to win a certain prize, it was sometimes hard to determine if it was gambling or giveaway/competition. Another argument for merging these categories is the semantic similarity, as all of these categories are about winning a prize.

5.6 Model Setup

To set up the models and perform training and evaluation, we used Python together with several libraries. To train the models, 80% of the data was used (70% training and 10% validation set for the deep learning models), and the remaining 20% was used for model evaluation. For the traditional machine learning algorithms, we used the implementations from the scikit-learn library, together with their default hyperparameters (Pedregosa et al., 2011). Scikit-learn's *TfidfVectorizer* was used to convert the raw messages to matrices of TF-IDF features. The parameters used with the *TfidfVectorizer* are shown in Table A.1 in the Appendix. For multi-label classification using these models, the problem was transformed into a number of independent binary tasks (one per label) using Scikit-multilearn's *BinaryRelevance* problem transformation.

The Flair library was used to train and evaluate the Flair model (Akbik et al., 2018). The parameters and embeddings used are shown in Table A.2 in the Appendix, and they were chosen based on recommended values from the Flair authors, as well as experimentation on the public datasets.

Lastly, we used the Huggingface's Transformers library together with the TensorFlow implementation and the ktrain library to set up, train, and evaluate the transformer models BERT, DistilBERT, and XLNet (Wolf et al., 2019). The Huggingface's Transformers library also provided the pre-trained model weights for the transformer models. These models are capable of performing multi-label classification using a single model. We used the default classification setup from the library, which has a sequence classification head (linear layer) on top of the pooled output.

The maximum sequence length for the transformer models was set to 32 tokens, as almost all messages were below this threshold, see Section 4.3.1 (Exploratory Data Analysis). The hyperparameters used for these models can be found in Table A.3 in the Appendix. These were chosen based on a combination of recommended values from their research papers and Github repositories, as well as some minor experimentation.

For the AG News dataset, we set up models for multi-class classification. For the Wikipedia Toxic Challenge and the Sinch data, we set up both multi-label and binary models, where the latter examined the case of *clean* vs. *dirty*.

To perform an extensive evaluation of the models on Sinch data, we did some further experimentation. For the case of multi-label using transformer-based models, we tried both the default setup of a single model, as well as having multiple independent binary models for each class. We also evaluated the multi-label performance of each model when the dataset consisted of only *dirty* messages, in addition to evaluating them on the entire dataset. This was done to evaluate if a potential solution should consist of a single multi-label classification step, or consist of a classification pipeline with an initial binary classifier, followed by a multi-label classifier for the predicted *dirty* messages.

5.7 Model Evaluation

The main metrics to evaluate the models were decided to be F1-score along with macro-averaging, after discussions with representatives at Sinch and our supervisor. F1-score is the harmonic mean of the precision and recall. These metrics have the following definitions:

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{5.1}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{5.2}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.3}$$

In classification tasks with multiple classes, Macro F1 score is a widely used metric. Occasionally, Micro F1 scores are also calculated. The Micro F1 score is calculated by counting the global true positives, false positives, and false negatives—regardless of class. This means that Micro F1 does not take class imbalance into account, which is why it can give a skewed performance estimate on imbalanced datasets. However, we have decided to include Micro F1 scores in the evaluation of multi-label models for additional insight.

Macro F1 on the other hand does take class imbalance into account, and is therefore often the preferred metric. There are two different definitions of Macro F1, yielding slightly different results. The first one is the "F1 of averages", where the harmonic mean is computed over the arithmetic means of precision and recall for each class. The second definition is the "averaged F1", where F1 scores are computed for each class and then averaged via arithmetic mean. Since the second definition has proven to be more robust (Opitz and Burst, 2019), and is also the standard adopted by the scikit-learn package, we have used the "averaged F1" definition in this project.

As mentioned in Section 5.6, 20% of the data was used for model evaluation, hereinafter referenced to as the test set. This data was not used while training the models, but used only to evaluate the models after training. To get the F1 scores, predictions on the test set were made, which were then fed together with the ground truth labels to scikit-learn metrics library (Pedregosa et al., 2011), which produced an output with multiple scores and metrics.

5.8 Comparison to Regular Expressions

To compare a machine learning solution to the existing regular expression solution at Sinch, we set up an experiment to compare certain metrics between these alternatives. Sinch representatives were particularly interested in the comparison of false positives and false negatives. Since the occurrence of SHAFT messages is very rare, we decided not to sample "real" data. Sampling 500 of these messages would likely give us less than 5 dirty messages, as the occurrence of dirty messages has been less than 1% according to our findings. While this could give us an understanding of the performance on clean messages and an estimation of false positives, it would likely not allow us to draw any certain conclusions regarding dirty messages and false negatives. This is because the performance of the model and RegEx on these dirty messages could vary significantly depending on the particular messages that were sampled, and not give a general understanding of its performance. To get an accurate estimation of false negatives, one would have to annotate a very large amount of randomly sampled messages. Given the limited scope and time frame of the project, we were unable to perform such experiments.

Instead of sampling 500 random messages, we let a trained binary DistilBERT model predict on thousands of new messages, and then we sampled 10 messages for every two-percent interval of the predicted dirty probability, meaning 10 messages in the interval 0.00-0.02, 10 messages in 0.02-0.04 and so on, adding up to 500 messages. This gave a uniform distribution of "dirty" probability among the messages. While this dataset does not represent a realistic scenario, this provides a difficult challenge with many "dirty" messages while not giving the ML model any apparent inherent advantage. DistilBERT was chosen as a model, for the same reasons mentioned in Section 5.4, and for the sake of consistency.

We manually annotated the 500 messages, and removed uncertain entries. Messages that were very similar but had not been removed by our duplicate detection script were also removed. This resulted in 339 annotated messages that could be used in the comparison. Finally, the regular expressions script was run on these messages. Messages that were flagged by the script were called "predicted positives" in our comparison, and messages that were not flagged were called "predicted negatives". For the comparison, the predicted positives and negatives of both the ML model and the RegEx script were compared to the true label of the message, which had been determined during annotation.

For a further comparison, we also ran the RegEx script on the same test set that we used to evaluate the machine learning models. Again, this is not representative of real data, but it is useful for a comparison of the two approaches.

A particular metric that can be estimated well for realistic data is the dirty precision of the RegEx script. Since we have annotated RegEx logs, and have the true positives and false positives for this data, the dirty precision can be easily calculated. This was included as a reference when evaluating the data for the comparison.

5.9 Prediction Time Benchmarking

Since Sinch sends approximately 1,000 messages per second, prediction time is a factor to consider. To benchmark the prediction times of the various models, we used trained binary predictors of each model and predicted samples from the Logs C dataset. We used `time`

module in Python to measure the time before and after the predictions, and the difference in time was then part of the output. Pre-processing required by each model, i.e. tokenization and vectorization, was included in the time span. The predictions were done in batches of 1 and 1,000 messages. The predictions were repeated 1,000 times with different messages and an average was calculated.

We performed the benchmarks, as well as the training of the large models, on a server at Sinch. It was equipped with an Nvidia RTX 2080 Ti 11 GB GPU, an Intel i7 9800X CPU and 64 GB RAM. The deep learning models used the GPU while training and predicting, while the traditional algorithms only used the CPU.

Chapter 6
Results

In this chapter, we present the results from the various models trained and evaluated on the three datasets. Each dataset is presented in a separate section. As our main metric, Macro F1 scores are presented for each case, while additional metrics are presented when appropriate. For the Sinch dataset, we have a more comprehensive results section, including the comparison to the current regular expressions solution as well as performance benchmarks for prediction time.

6.1 AG News

	Macro F1
Logistic Regression	0.82
Naive Bayes	0.83
SVM	0.80
Random Forest	0.75
Flair	0.86
DistilBERT	**0.88**
BERT	**0.88**
XLNet	**0.88**

Table 6.1: Macro F1 for each model on the AG News dataset.

The classification results for the multi-class AG News dataset are shown in Table 6.1. The deep learning based models outperformed the traditional machine learning models by quite a large margin on the modified AG News dataset, with all transformers-based models achieving a Macro F1 score of 0.88. Figure 6.1 shows the confusion matrix for the BERT model, showing that it struggled especially with separating the *world* and *sports* classes from each other.

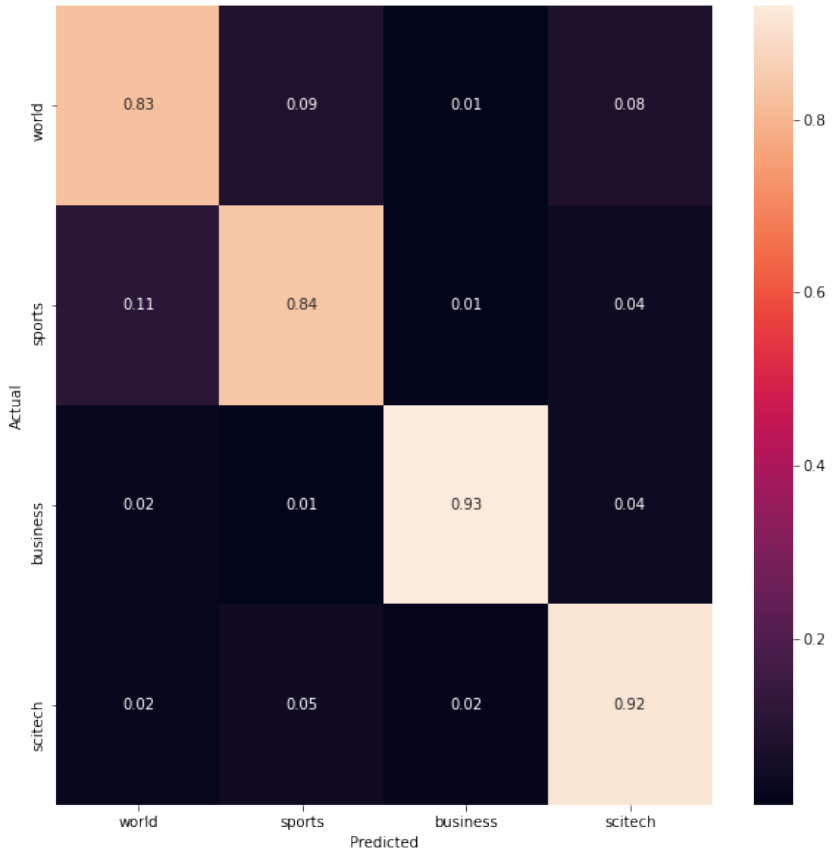


Figure 6.1: Confusion matrix for BERT on the AG News dataset.

6.2 Wikipedia Toxic Challenge

	Macro F1
Logistic Regression	0.70
Naive Bayes	0.78
SVM	0.84
Random Forest	0.85
Flair	0.86
DistilBERT	0.93
BERT	0.93
XLNet	**0.94**

Table 6.2: Macro F1 for each model on the binary Wikipedia Toxic Comment dataset.

The results for the binary Wikipedia Toxic Comments dataset are shown in Table 6.2. The transformers-based models again outperformed the other models, just as on the AG News dataset. This time, XLNet performs slightly better than both DistilBERT and BERT with a Macro F1 score of 0.94.

The results for the multi-label Wikipedia Toxic Comments dataset are shown in Table 6.3.

	Macro F1	Micro F1
Logistic Regression	0.24	0.43
Naive Bayes	0.23	0.45
SVM	0.44	0.66
Random Forest	0.36	0.66
Flair	0.35	0.67
DistilBERT	**0.48**	**0.83**
BERT	0.47	0.80
XLNet	0.45	0.80

Table 6.3: Macro and Micro F1 for each model on the multi-label Wikipedia Toxic Comment dataset.

As expected, the Macro F1 scores are much lower on the multi-label version of the dataset compared to the binary version. Again, the transformers based models perform the best, with DistilBERT achieving the highest Macro F1 and Micro F1 scores of 0.48 and 0.83 respectively.

	Toxic	Severe Toxic	Obscene	Threat	Insult	Identity Hate
Logistic Regression	0.48	0.07	0.55	0.00	0.34	0.00
Naive Bayes	0.54	0.00	0.56	0.00	0.26	0.00
SVM	0.71	0.25	0.78	0.00	0.56	**0.33**
Random Forest	0.70	0.07	0.75	0.00	0.65	0.00
Flair	0.78	0.00	0.62	0.00	0.69	0.00
DistilBERT	**0.90**	0.29	**0.87**	0.00	**0.82**	0.00
BERT	**0.90**	**0.33**	0.83	0.00	0.78	0.00
XLNet	**0.90**	0.20	0.81	0.00	0.79	0.00

Table 6.4: F1 score per class and model on the multi-label Wikipedia Toxic Comment dataset.

The per-class performance is shown in Table 6.4. Several models perform decently on the *toxic*, *obscene*, and *insult* classes, but they all struggle with *severe toxic*, *threat*, and *identity hate* classes. This corresponds directly with the number of samples of each class in the training dataset as shown in Figure 4.1. All models performed worse on the classes with fewer training samples. For the class with the fewest samples, *threat*, all models got an F1 score of 0.00.

6.3 Sinch Data

As shown in Table 6.5, Random Forest, DistilBERT, and BERT performed the best on the binary Sinch dataset, achieving a Macro F1 score of 0.89. Flair, which performed in the middle of the pack on the previous two datasets, performs the worst with a Macro F1 score of 0.80. For each model the performance on the *dirty* messages was considerably worse than the performance on the *clean* samples, with an average reduction in F1 score of -0.065 between *clean* and *dirty*. Out of the models with the highest Macro F1 scores, BERT had the highest F1 score for the *dirty* class, while DistilBERT had the highest recall for this class. Random Forest and BERT had the highest F1 scores for the *clean* class.

	Cln P	Cln R	Cln F1	Dty P	Dty R	Dty F1	Macro F1
Logistic Reg	0.83	**0.96**	0.89	**0.92**	0.72	0.81	0.85
Naive Bayes	0.83	0.92	0.87	0.85	0.73	0.79	0.83
SVM	0.88	0.93	0.90	0.88	0.81	0.84	0.87
Random Forest	0.88	**0.96**	**0.92**	**0.92**	0.80	0.86	**0.89**
Flair	0.85	0.82	0.84	0.75	0.78	0.77	0.80
DistilBERT	**0.90**	0.93	0.91	0.88	**0.84**	0.86	**0.89**
BERT	0.89	0.95	**0.92**	**0.92**	0.82	**0.87**	**0.89**
XLNet	0.88	0.90	0.89	0.84	0.82	0.83	0.86

Table 6.5: Precision, Recall, and F1 scores for each model on the binary Sinch dataset.

Cln = Clean, **Dty** = Dirty

For multi-label performance, we can make a general observation when comparing Table 6.6 and Table 6.7. We observe that the performance for every model is increased when excluding *clean* messages, consequently only training and predicting on *dirty* messages.

Table 6.7 shows that SVM performs the best on the "only dirty" multi-label dataset with a Macro F1 score of 0.92, closely followed by DistilBERT$_{MB}$ and Random Forest at 0.91 and 0.90 respectively. Flair again performs the worst with a Macro F1 score of 0.50. On a per-class basis the top performances are spread among six different models. It should be noted that Flair, DistilBERT$_S$, and XLNet$_S$ all got 0.00 in F1 scores on the *sexual* class, bringing their Macro F1 score down considerably. By utilizing multiple binary models (*MB*) instead of a single multi-label model (*S*) the Macro F1 scores improved for DistilBERT (**+0.15**), BERT (**+0.07**), and XLNet (**+0.11**), in large part thanks to improved performance on the *sexual* class.

	Sexual	Alcohol	Drugs	Loan	TVH	SGG	Macro	Micro
Logistic Reg	0.12	0.67	0.48	0.53	0.58	0.67	0.51	0.63
Naive Bayes	0.32	0.63	0.10	0.65	0.59	0.65	0.49	0.60
SVM	**0.77**	0.79	0.81	0.81	**0.92**	0.83	0.82	0.82
Random Forest	0.48	0.76	0.79	0.63	0.87	0.82	0.72	0.78
Flair	0.00	0.66	0.00	0.48	0.11	0.37	0.27	0.21
DistilBERT$_S$	0.00	0.85	0.69	0.67	0.79	0.86	0.64	0.81
BERT$_S$	0.00	**0.86**	0.80	0.76	0.79	**0.87**	0.68	0.83
XLNet$_S$	0.00	0.82	0.78	0.83	0.49	0.80	0.62	0.75
DistilBERT$_{MB}$	0.72	0.83	0.82	**0.84**	0.84	0.86	0.81	0.84
BERT$_{MB}$	0.75	**0.86**	**0.87**	0.82	0.84	0.84	**0.83**	**0.85**
XLNet$_{MB}$	0.57	0.82	0.70	0.76	0.70	0.83	0.73	0.79

Table 6.6: F1 scores for each class and model on the multi-label Sinch dataset, including clean messages.

S = Single model for multi-label, **MB** = Multiple binary models for multi-label

	Sexual	Alcohol	Drugs	Loan	TVH	SGG	Macro	Micro
Logistic Reg	0.25	0.90	0.71	0.67	0.62	0.93	0.68	0.85
Naive Bayes	0.42	0.89	0.84	0.74	0.77	0.91	0.76	0.86
SVM	**0.88**	0.93	0.91	0.91	0.93	**0.95**	**0.92**	**0.93**
Random Forest	**0.88**	0.93	0.88	0.84	**0.94**	0.93	0.90	0.92
Flair	0.00	0.83	0.48	0.61	0.26	0.81	0.50	0.72
DistilBERT$_S$	0.00	0.93	0.84	**1.00**	0.87	0.92	0.76	0.90
BERT$_S$	0.25	0.93	0.86	**1.00**	0.91	0.93	0.81	0.91
XLNet$_S$	0.00	0.93	0.86	0.93	0.59	0.89	0.70	0.86
DistilBERT$_{MB}$	0.75	**0.94**	**0.94**	0.98	0.90	0.94	0.91	**0.93**
BERT$_{MB}$	0.67	**0.94**	0.86	**1.00**	0.88	0.94	0.88	0.92
XLNet$_{MB}$	0.48	0.90	0.91	0.96	0.70	0.90	0.81	0.86

Table 6.7: F1 scores for each class and model on the "only dirty" multi-label Sinch dataset, excluding clean messages.

S = Single model for multi-label, **MB** = Multiple binary models for multi-label

6.3.1 Comparison to Regular Expressions

The results of the comparison between the RegEx filter and DistilBERT model on the dataset based on uniform distribution of dirty probabilities is shown in Table 6.8. Out of the 339 messages used in the comparison, 122 were actual positives (*dirty*) and 217 were actual negatives (*clean*), as determined by annotation. The binary DistilBERT model predicted roughly half of these messages as *clean* and half as *dirty*, which was the expected outcome as the messages were chosen with a uniform distribution of dirty probabilities. The removal of uncertain entries after annotation skewed the prediction to not be an exact 50/50 split. The regular expressions script on the other hand matched only 8 of the messages, out of which 4 were correct, meaning that it missed a total of 118 dirty messages.

When running the regular expressions script on the test set used for the ML model evaluation, the DistilBERT model outperformed the script on every metric, as seen in Table 6.9. The actual positives (*dirty*) were 392 and actual negatives (*clean*) were 581 in the test set.

As a reference, the real dirty precision of the regular expressions can be estimated from the true positives and false positives of the annotated RegEx log data. These were 1,002 and 1,195 respectively, resulting in a dirty precision of 0.46. Similar values can be seen in this experiment, where dirty precisions of 0.50 and 0.53 were measured from the experiment data and the test set respectively.

	DistilBERT	RegEx
Predicted Positives	173	8
Predicted Negatives	166	331
False Positives	97	**4**
False Positive Rate	0.45	**0.02**
False Negatives	**46**	118
False Negative Rate	**0.38**	0.97
Clean Precision	**0.72**	0.64
Clean Recall	0.62	**0.98**
Clean F1	0.63	**0.78**
Dirty Precision	0.44	**0.50**
Dirty Recall	**0.62**	0.03
Dirty F1	**0.52**	0.06
Macro F1	**0.57**	0.42

Table 6.8: Comparison of DistilBERT and Regex on the selected experiment data.

	DistilBERT	RegEx
Predicted Positives	373	451
Predicted Negatives	600	522
False Positives	**43**	213
False Positive Rate	**0.07**	0.37
False Negatives	**62**	154
False Negative Rate	**0.16**	0.39
Clean Precision	**0.90**	0.70
Clean Recall	**0.93**	0.63
Clean F1	**0.91**	0.67
Dirty Precision	**0.88**	0.53
Dirty Recall	**0.84**	0.61
Dirty F1	**0.86**	0.56
Macro F1	**0.89**	0.62

Table 6.9: Comparison of DistilBERT and Regex on the test set used for ML model evaluation.

6.4 Prediction Time Benchmarks

Figure 6.2 shows the average prediction time of the different models for predicting a single message, while Figure 6.3 shows the average prediction time for batches of 1,000 messages. The traditional machine learning models are orders of magnitude faster than the deep learning models at predicting single messages. Support vector machines, logistic regression, and naive Bayes all take less than 0.6 milliseconds to predict a single message, whereas XLNet takes almost 100 times longer at 55 milliseconds.

Looking at the time for batches of 1,000 messages show a different pattern. Here the deep learning models are roughly 2 to 3 times slower than most traditional models. It is also

worth noting that Random Forest seems to scale well with an increased batch size.

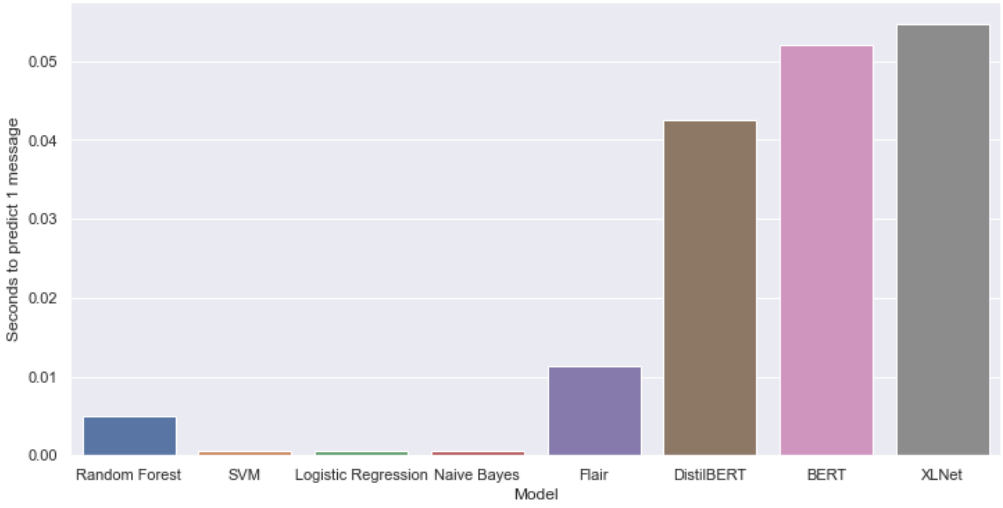


Figure 6.2: Average prediction time with batches of a single message.

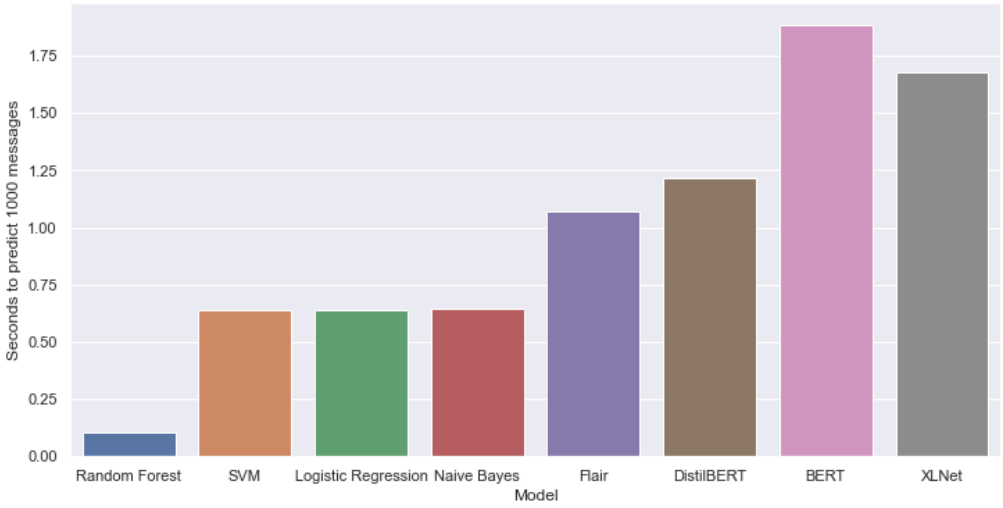


Figure 6.3: Average prediction time with batches of 1,000 messages.

Chapter 7

Discussion

In this chapter, we discuss the results and potential shortcomings of our methodology. We attempt to explain certain results in regard to the theory behind the models as well as properties of the datasets, such as class imbalance, short text length and specific domain language.

7.1 Finding and Annotating Data

One of the key challenges with the Sinch dataset is that it is very imbalanced. As shown in Section 4.3.1 (Exploratory Data Analysis), a vast majority of all messages sent are clean, and out of the remaining messages the class distribution is also imbalanced.

Out of the thousands of messages that were classified using DistilBERT for the comparison to RegEx in Section 5.8, roughly 0.8% were predicted to be dirty. This can give us a rough estimate of the prevalence of dirty messages in the message logs when duplicates are excluded. This estimate is however based entirely on the prediction performance of the model, and as such should not be considered entirely accurate. The 0.8% figure does however align well with our experience from manually looking through the message logs without duplicates.

This presented a challenge in how to efficiently annotate enough messages of each category to train our models. If we randomly annotated messages we would, on average, have to annotate 1,000 messages to get 8 *dirty* messages. This is not a feasible method, especially considering the limited timeline and scope of the project. To get around this we used a bootstrapping method based on Regular Expressions as mentioned in Section 5.4 (Finding Data and Bootstrapping). Using this method, the dataset we annotated contained 40% *dirty* messages, making the annotating process much more efficient. It is however important to note that this biases the annotated dataset towards messages that the RegEx script could find. However, during multiple steps of the bootstrapping process, many messages that would not have been flagged by the RegEx script were annotated, making this bias less severe.

7.2 Data Imbalance

The class imbalance creates challenges on a model level. If the training dataset is heavily imbalanced the model will have to take this into account to perform well. Due to the bootstrapping technique we used to find training data, the training dataset was much more balanced than the raw data. This instead creates a challenge in making sure that the trained model generalizes well to the raw, more unbalanced dataset, it will later be used on.

Usherwood and Smit (2019) discuss the pros and cons of deep transfer learning versus traditional machine learning models in low-shot NLP classification tasks. The authors concluded that if there is a pre-trained language model available for the task, the deep learning methods perform better than the traditional methods. Our results on the other hand show that SVMs perform on par, if not slightly better, than the deep learning models on the Sinch dataset. Usherwood and Smit emphasise the importance of a pre-trained language model for the specific task, and we discuss some of the extra challenges Sinch's messages might produce in this regard in Section 7.4 (Domain Language). The authors end by saying that there is still much work to be done in obtaining high-quality classifiers for low-shot classification, which is something we also experienced.

7.3 Message Length and Split Messages

By their nature, SMS (*Short* Message Service) messages are short in length. In our dataset the maximum length was 160 characters, the standard maximum length of an SMS message. Longer messages, that are technically sent as multiple messages behind the scenes, were split into multiple messages in the raw dataset. Approximately a quarter of all messages had been cut into multiple parts, and as previously mentioned we did not have any efficient way of pairing them back together.

This creates two big challenges for our models in both that context is lost between the parts, and also in that one part of the message might be considered *clean* while the other part is *dirty*. It is reasonable to assume this loss of context between parts hurts the deep learning models more than the traditional models as they are better at understanding context. Our models are still able to get decent results, but working with split messages creates unnecessary complexity. To use any of these models in a production environment Sinch should make sure the models have access to the full message content.

7.4 Domain Language

When using pre-trained models, one has to consider the corpus that they have been pre-trained on, and how that might differ from the task-specific corpus. All deep learning models we used have been trained on external corpora, often the English Wikipedia corpus.

As shown in Section 4.3.1 (Exploratory Data Analysis), the language used in Sinch's messages is very specific to business-to-consumer SMS messaging. Furthermore, all messages are short. Due to the character limit, there are also heavy use of abbreviations, which is not as common in e.g. the Wikipedia corpus. This likely played a role in the performance of the complex deep learning models. They performed better than the traditional models on both

the AG News and Wikipedia Toxic Comments datasets, where the language used is closer to the natural language in the corpora used during pre-training. However, since the language is quite different in the Sinch dataset, the acquired knowledge from the pre-training corpora could likely not be applied to the same extent. Not only is the occurrence of rare and specific words, abbreviations, and codes common in business-to-consumer text messages, but the sentence structure is often different as well. Many business-to-consumer text messages do not even have full sentences, and the occurrence of multiple full sentences is even rarer.

There is also a difference in how the different models handle out-of-vocabulary words. As the occurrence of abbreviations, unusual words, and codes are rather common in Sinch's messages the handling of these can be assumed to have a noticeable effect on prediction performance. In the case of the traditional, TF-IDF based models, out-of-vocabulary words are simply ignored. Theoretically, the deep learning models are able to take these into account using character-delimited and subword-delimited embeddings. However, further pre-training on in-domain data might be needed to see significant improvements from this capability.

Much of the latest research in the field of NLP has been focused on using deep learning approaches to better understand context and long-range dependencies between clauses and sentences. It is worth considering to which degree these advances can be utilized for this specific task. Our traditional models, using only TF-IDF-based features, produced results that are on par or slightly better than the deep learning models, indicating that the deep learning models are not being utilized to their full extent. It should also be noted that the traditional machine learning models are able to produce good results using only TF-IDF features that do not contain information about word order (except for local order in the case of bigrams and trigrams), indicating that advanced approaches that take context and word order into account might not be necessary.

Using transformer models and embeddings pre-trained on large amounts of Sinch's own data might however produce better results, but one has to weigh the increased complexity and computational cost of such an approach. Even though Sinch could potentially use a large number of past messages for the pre-training, the dataset might not be large and diverse enough to considerably improve the models.

7.5 Classification Performance

The AG News and Wikipedia Toxic Comment datasets were used to evaluate and compare different models and setups. The AG News dataset is multi-class and as such not directly comparable to the Sinch dataset. Wikipedia Toxic Comments however was used both in a binary and multi-label setting, and the class imbalance is relatively similar to that of the Sinch dataset. As such it is a suitable dataset to benchmark against.

For both datasets, the transformers-based models consistently outperformed the other models by a relatively large margin. This is not surprising as they are the current state-of-the-art models for similar NLP tasks. The classification performance on the Sinch dataset however paints a different picture. On the binary dataset, random forest achieves an F1 score of 0.89, on par with DistilBERT and BERT. On the multi-label dataset, SVM achieves the highest Macro F1 score, slightly outperforming DistilBERT$_{MB}$. As previously discussed in both Section 7.2 (Data Imbalance) and Section 7.4 (Domain Language), a large contributing factor to the decreased performance of the deep learning based models is believed to be the

unique language characteristics of Sinch's messages, with messages being short and containing unusual words, abbreviations, and sentence structures.

Another aspect to take into account is classification performance in relation to dataset size. In our experiments, we had an annotated dataset of 4,862 Sinch messages, which is rather small in the context of deep learning. This is especially apparent in the multi-label classification task, where some classes (e.g. *firearms*) only contained 12 example messages, making us decide to remove that as a category to classify. As visualized in Figure 7.1, deep learning models often scale better with increased training dataset size, and often start outperforming traditional machine learning models once the training set is large enough in size. If effort is put into annotating many more messages in an attempt to create more accurate models, it is reasonable to assume that deep learning will be better at utilizing this additional training data. Deep learning can also utilize unsupervised pre-training on vast amounts of past messages as discussed in Section 7.4 (Domain Language).

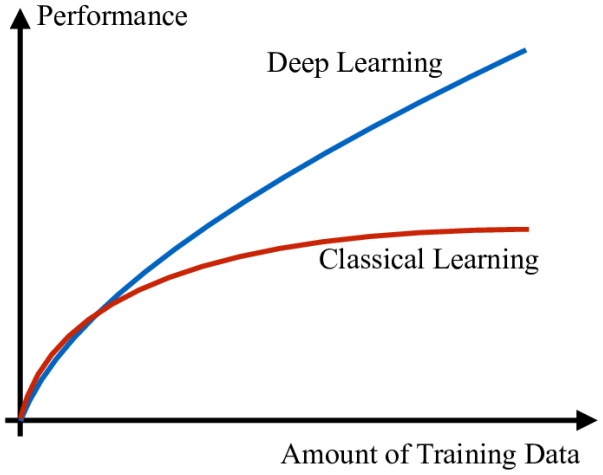


Figure 7.1: Prediction performance as a function of amount of training data available for deep and classical machine learning. After Zappone et al. (2019).

One more factor to consider is hyperparameter setup. We have not thoroughly experimented with different hyperparameters for the models. For the traditional machine learning models, we used the default hyperparameters. For the deep learning models, we started with a baseline of default or recommended hyperparameters and performed some minor experimentation and tweaking. However, to optimize the performance of each model, more tweaking can be done.

As a side note, the dataset used in the comparison of DistilBERT to RegEx, as described in Section 5.8, can also be used to gain an understanding of how reliable the predictions are in different intervals of "dirty probability", for specifically the DistilBERT model. We can observe that it makes the most wrong predictions when it predicts 0.6-0.7 in dirty probability, see Figure A.1 in the Appendix. This could be taken into account in the event of preparing the model for use in a production pipeline, as one can consider the dirty probability to set custom thresholds for blocking messages as well as flagging certain messages for manual review. However, as these results are specific to the DistilBERT model, we cannot draw any general conclusions that apply to all models.

The idea of setting custom thresholds can be further developed to improve the performance of the models. To potentially further improve on our results, one could create a function to optimize the performance of the models, by finding the ideal threshold for each class and model. However, one factor to consider when doing so is the importance of precision and recall for the use case, and which one to prioritize, as tweaking the threshold often means a trade-off between these two.

7.6 Prediction Time

Sinch sends millions of messages each hour on behalf of their customers. Therefore, any filtering implementation has to take prediction time into account if it is to be used in real-time in a production setting. As shown in Figure 6.2 and Figure 6.3, there is a large difference in prediction time between models and batch sizes. The traditional machine learning models are generally orders of magnitude faster than the deep learning approaches to predict single messages. This is expected as they are less complex and have less overhead to setup.

When looking at the prediction times for batches of 1,000 messages, the difference in prediction time is much smaller, likely due to the fact that any overhead during startup has a smaller impact. Larger batch sizes also allow the transformer-based models to better utilize the parallel processing capabilities of the GPU.

These numbers should be considered when deciding on how these models can be used in a production environment. If Sinch can batch multiple messages together before prediction, the difference is small. If they have to be predicted one-by-one, the prediction time can be a major factor in deciding which models are viable.

7.7 Comparison to Regular Expressions

7.7.1 Empirical Results

The experiment to compare the DistilBERT binary predictor to the regular expressions filter does have some drawbacks. First of all, after the removal of uncertain entries the distribution of dirty probabilities is not perfectly uniform. It should still be somewhat evenly distributed, under the assumption that uncertain entries are also somewhat uniformly distributed in this dataset. There were 173 predicted *dirty* and 166 predicted *clean* messages after removal of uncertain entries, which is a small indication that this is the case.

The experiment itself with a dataset from a uniform distribution of dirty probabilities certainly does not represent a realistic scenario. Out of the thousands of unique messages sampled from Logs C, 98.9% were predicted with a dirty probability of less than 0.2. 0.6% were predicted with a probability above 0.8, and consequently the remaining 0.5% were in the interval 0.2-0.8.

Since the RegEx script only flagged 8 messages as dirty, the false positive and false negative rates assume extreme values. As mentioned in Section 5.8, one would have to sample real data to effectively measure the actual values for these metrics. However, for the false negative rate, one would need to annotate a huge amount of randomly sampled data to get statistically reliable estimates as the occurrence of dirty messages is very rare. The remarkably high false

negative rate for the RegEx script resulted in a very low recall for the *dirty* class, 0.03, which in turn resulted in a very low F1 score for dirty messages, 0.06. This is the main reason why the Macro F1 score for the RegEx script is lower than the DistilBERT model for the selected experiment dataset.

The results from the test set also have their limitations. Once again, this dataset is not representative of the raw Sinch dataset as it is based on a bootstrapping method as discussed in Section 5.4. Instead, this can be used as an additional baseline to compare each model against. The DistilBERT model is specifically trained on similar data, so its dominant scores should not come as a surprise. However, the dataset used is heavily based on the RegEx logs, which explains the decent performance of the script, in particular when compared to the experiment dataset.

When comparing the machine learning models to the regular expressions approach, the machine learning models outperform the RegEx filter in both experiments in regard to Macro F1 as shown in Table 6.8 and Table 6.9. On the test set, DistilBERT outperforms the RegEx approach on every metric, achieving a Macro F1 of 0.89, compared to 0.62 for the RegEx model. On the dataset with uniformly distributed dirty probabilities, the results are less conclusive. As previously mentioned, with only 8 messages flagged as *dirty* by the RegEx filter, the metrics assume extreme values. One of the more interesting findings is the disparity in the number of positive predictions, with DistilBERT predicting 51% as *dirty* and RegEx only predicting 2%. Looking at Macro F1 DistilBERT outperforms the RegEx approach with a Macro F1 score of 0.57 to 0.42 on this dataset.

7.7.2 General Comparison

Aside from the empirical results, one should also consider the fundamental differences between ML models and the RegEx script. One advantage of the machine learning models is that most of them are probabilistic, or can be transformed into probabilistic classifiers. As such, message prediction probability can be taken into account, e.g. messages with a dirty probability in a certain range can automatically be sent to manual review. This also enables Sinch to easily tune the machine learning models by adjusting the thresholds for classification, allowing them to make the trade-off between e.g. precision and recall. This is something our supervisors at Sinch emphasised the importance of for their use case. For the RegEx model, the classification is binary, with no simple way of producing probabilities.

The current RegEx approach is also unable to perform multi-label classification, meaning that it cannot provide any data insights like an ML model can. However, this could technically be solved by splitting the RegEx script into multiple scripts, each with a matching pattern corresponding to a category.

Another aspect to consider is the prediction time, as this could be a crucial factor when processing messages in real-time, as discussed in Section 7.6. The RegEx solution is extremely fast and can easily process over 1,000 messages per second, while some transformer-based models could potentially struggle to keep up with this rate, at least with the current hardware configuration.

The ability to backtrack and explain the results of a classification should also be considered. This is particularly important when working with false classifications and attempting to reduce them. With a RegEx script, one can easily explain why a certain message has been flagged or not. If any part of the message matches the RegEx pattern, it will be flagged and

vice versa. For ML models, explaining a classification is harder. However, there are explanation techniques that can help with this task, such as LIME (Ribeiro et al., 2016).

The workflow for maintaining and improving the classification performance of the two approaches is also a significant difference. With a RegEx script, one can make changes to the pattern and directly see results. However, the changes could introduce new false classifications. To handle specific cases, one would need an increased number of conditionals, adding to the complexity and worsening the maintainability of the script. For ML models, the performance is improved by expanding the dataset by annotation. However, the process of resolving specific false classifications is more complex and can involve exploring and tweaking the dataset.

7.8 Annotation Ambiguity

An important source of error to consider is the ambiguity for some messages, as it is sometimes not clear which category it should belong to. For the most part, this has been temporarily solved by using the *uncertain* label, so that a domain expert can review these messages in the future. However, there are still some messages that could be wrongly labeled, as a result of ambiguity in the language used in the messages as well as in the SHAFT guidelines and the information we received from Sinch prior to labeling. There is also a factor of human error, as an ambiguous message could be labeled differently by the same person on two different occasions.

To get a rough estimate of the impact of the aforementioned factor, we took advantage of the fact that some duplicates had been included in the annotation process, as mentioned in Section 5.4. At this stage of annotation, there were 637 excess duplicates (excluding the first occurrence) of the messages. Out of these messages, 20 had differing labels compared to the original. This means that 3.1% of the times the annotator labeled a message that he had seen before, he ended up picking differing labels compared to the original message. When considering the binary case of *clean* and *dirty*, only 11 or 1.7% of the duplicates had been given a contradictory label.

As mentioned in Section 5.3, we only used one annotator to prioritize dataset size given our limited time frame. Ideally, several annotators should be used for increased data quality, and to get a better understanding of what kind of messages are more likely to be mislabeled. When using multiple annotators, one can get a measurement of the agreement between annotators by using Krippendorff's alpha (Krippendorff, 2013). This measurement could then be used to draw further conclusions of the reliability of the annotated data.

7.9 Classification Pipeline

As we can clearly observe that the models performed much better when only using *dirty* messages for the multi-label case, we propose using a pipeline approach consisting of an initial binary *clean*/*dirty* classifier, and then a multi-label classifier for only the predicted *dirty* messages. This approach breaks the task into two separate, simpler tasks and each of them can be optimized independently using different hyperparameters or models. This separation also makes sense from a business perspective as a binary filter can be used in the message sending

pipeline, corresponding to the current regular expressions solution, whereas the multi-label classifier would be used for further insight. The multi-label classification would also not be as performance-critical in terms of both accuracy and prediction time.

Since the categories *hateful*, *firearms*, and *phishing* are currently not handled, these would have to be dealt with separately until sufficient data within these categories has been collected. Until then, the parts of the RegEx script that attempts to identify these messages could still be used in conjunction with the binary classifier.

Chapter 8
Further Work

In this chapter, we discuss the potential further work that could be carried out to expand and improve on this project. We take the shortcomings of our project into account, as well as unexplored areas.

8.1 Taking Metadata Into Account

In our dataset, we had access to three metadata parameters: the number of times the same exact message was sent within a specific time interval, the sender name, and the sender short-code. We did not take any of these parameters into account (except while removing duplicate messages during pre-processing). Including this data might improve the classification in some cases, especially in regard to *phishing* as these messages are crafted to be as similar as possible to legitimate messages. It would also be interesting to take a sender's past sending behaviour into account, with the hypothesis being that a sender who has sent questionable messages in the past should be more closely scrutinized.

8.2 More Robust Annotating Process

One of the fundamental truths in any machine learning task is that a model will only be as good as the data used to train it. As such there should be further work to get high quality annotated data at a larger scale, as discussed in Section 7.8.

One challenge was that the guidelines for how a message should be classified were not clearly defined. This led to 24% of the annotated messages being labeled as uncertain, and thus not included in the training dataset. One way to get around this would be to use annotators who are well versed in the various guidelines. Another approach would be to have multiple annotators annotate each message and then apply only the labels used by a majority of the

annotators. This process would also reduce the likelihood of mislabeled messages due to human error.

As shown in Section 4.3.1, on exploratory data analysis, the class distribution of the labeled messages is imbalanced. While the *alcohol* class has over 750 annotated examples, *firearms* has only 12. For a model to be able to classify messages reliably, more training examples are needed. The process explained in Section 5.4 (Finding Data and Bootstrapping) could be used to specifically target the underrepresented classes. While this process is efficient, improvements or alternative methods that reduce the RegEx bias should also be investigated.

8.3 Data Augmentation

An approach that should be considered is data augmentation. By automatically creating altered versions of each annotated message, more training data can be artificially created, which in turn can help the model's generalization performance. This is particularly useful when the training dataset is small, such as in our case. While augmenting the messages, it is crucial that the messages are altered enough to help improve generalization performance, while not being altered enough to change the label of the message or to create nonsensical messages. A common approach for this is back-translation. Recent studies have shown that performing data augmentation on unlabeled data can significantly improve semi-supervised learning performance in the NLP domain (Xie et al., 2019).

8.4 Pre-training on Domain Data

As discussed in Section 7.4 (Domain Language), the language used in Sinch's messages is very specific to business-to-consumer SMS messaging. As such, using models and embeddings pre-trained on corpora with different language characteristics (e.g. BERT, Flair) will not perform optimally. As the pre-trained deep learning models outperformed the traditional models on both the AG News and Wikipedia Toxic Comments dataset, but not on the Sinch dataset, it is reasonable to assume that the language characteristics in Sinch's messages is hard for these models to understand. Given that Sinch has logs of many millions of past messages, it would be interesting to look into further pre-training these models on this dataset. This approach has been shown to boost text classification performance with BERT, even with small domain-specific datasets (Sun et al., 2019).

Chapter 9
Conclusions

In this thesis, we explored the possibility of using machine learning to reliably classify short text messages. We also compare our machine learning approaches to Sinch's current regular expression filter.

One of the key takeaways is that the language used in Sinch's messages is very domain-specific. This is likely an important factor in explaining why state-of-the-art deep learning models pre-trained on external corpora do not perform better than certain traditional machine learning models, such as support vector machines using simple TF-IDF features, on this dataset. This is in contrast to the results for both the AG News and Wikipedia Toxic Comments datasets, where the deep learning models outperformed the traditional models by a large margin.

We also found that splitting the prediction pipeline into a separate binary and multi-label classification step improved prediction accuracy. This pipeline structure also affords Sinch increased flexibility in tailoring the solution to fit their needs, as both tasks can be optimized independently.

Another factor to consider is that the machine learning models are more informative as they output predicted probabilities unlike the RegEx filter's binary prediction. They also allow multi-label prediction with decent accuracy. On the other hand, regular expressions are easy to create, update, and it is straightforward to understand why a certain message is flagged. With a machine learning solution there is more overhead, but we found the predictions to be more accurate.

In this project, we have trained machine learning models that outperform Sinch's current regular expression filter at most metrics. While this shows great promise for the machine learning approach, there is still further work that should be performed before using these models in production. As with any machine learning task, annotating high quality data at a larger scale is crucial for improved classification performance. There should also be efforts put into reducing the RegEx bias in the annotated dataset. Lastly, efforts should be put into achieving more accurate classification performance estimates, especially in regard to false negatives.

In summary, our machine learning approach shows great promise, but there is still room for further improvement. While regular expressions are hard to improve, machine learning models will keep improving given advances in model architecture and better annotated datasets.

References

Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., and Vollgraf, R. (2019). FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, Minneapolis, Minnesota. Association for Computational Linguistics.

Akbik, A., Blythe, D., and Vollgraf, R. (2018). Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. P. (2011). Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398.

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

Cui, Y., Zhou, F., Lin, Y., and Belongie, S. (2016). Fine-grained categorization and dataset bootstrapping using deep metric learning with humans in the loop.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

D'Sa, A. G., Illina, I., and Fohr, D. (2019). Towards non-toxic landscapes: Automatic toxic comment detection using dnn.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`. Checked 2020-04-10.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

Johnson, M. (2009). How the statistical revolution changes (computational) linguistics. In *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*, pages 3–11, Athens, Greece. Association for Computational Linguistics.

Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21.

Joseph, S., Sedimo, K., Kaniwa, F., Hlomani, H., and Letsholo, K. (2016). Natural language processing: A review. *Natural Language Processing: A Review*, 6:207–210.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, USA, 1st edition.

Kaggle (2017). *Toxic Comment Classification Challenge*. By: Jigsaw/Conversation AI. `https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge`. Checked 2020-03-27.

Krippendorff, K. (2013). *Content Analysis. An Introduction to Its Methodology*. Sage Publications, 3rd edition edition.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, USA.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119, Red Hook, NY, USA. Curran Associates Inc.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill.

Nakayama, H., Kubo, T., Kamura, J., Taniguchi, Y., and Liang, X. (2018). doccano: Text annotation tool for human. Software available from https://github.com/doccano/doccano.

Opitz, J. and Burst, S. (2019). Macro f1 and macro f1.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Rajaraman, A. and Ullman, J. D. (2011). *Data Mining*, page 1–17. Cambridge University Press.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA. Association for Computing Machinery.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *NeurIPS EMC2 Workshop*.

Sun, C., Qiu, X., Xu, Y., and Huang, X. (2019). How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583.

Usherwood, P. and Smit, S. (2019). Low-shot classification: A comparison of classical and deep transfer machine learning approaches.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., and Brew, J. (2019). Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Xie, Q., Dai, Z., Hovy, E. H., Luong, M., and Le, Q. V. (2019). Unsupervised data augmentation. *CoRR*, abs/1904.12848.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J. G., Salakhutdinov, R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237.

Young, T., Hazarika, D., Poria, S., and Cambria, E. (2017). Recent trends in deep learning based natural language processing. *CoRR*, abs/1708.02709.

Zappone, A., Di Renzo, M., and Debbah, m. (2019). Wireless networks design in the era of deep learning: Model-based, ai-based, or both? *IEEE Transactions on Communications*, PP:1–1.

Appendices

Appendix A

Parameter	Value
ngram_range	(1,3)
stop_words	"english"
min_df	5
norm	"l2"
sublinear_tf	True

Table A.1: Parameters used for the *TfidfVectorizer*.

Classifier Parameters	
learning_rate	0.1
max_epochs	40
Document Embeddings	
type	DocumentRNNEmbeddings
hidden_size	512
reproject_words	True
reproject_words_dimension	256
Word Embeddings	
	WordEmbeddings('glove')
	FlairEmbeddings('news-forward-fast')
	FlairEmbeddings('news-backward-fast')

Table A.2: Hyperparameters used for the Flair model.

	DistilBERT	BERT	XLNet
Learning rate	3e-5	3e-5	2e-5
Batch size	32	32	12
Epochs	4	4	4

Table A.3: Hyperparameters used for the transformer models.

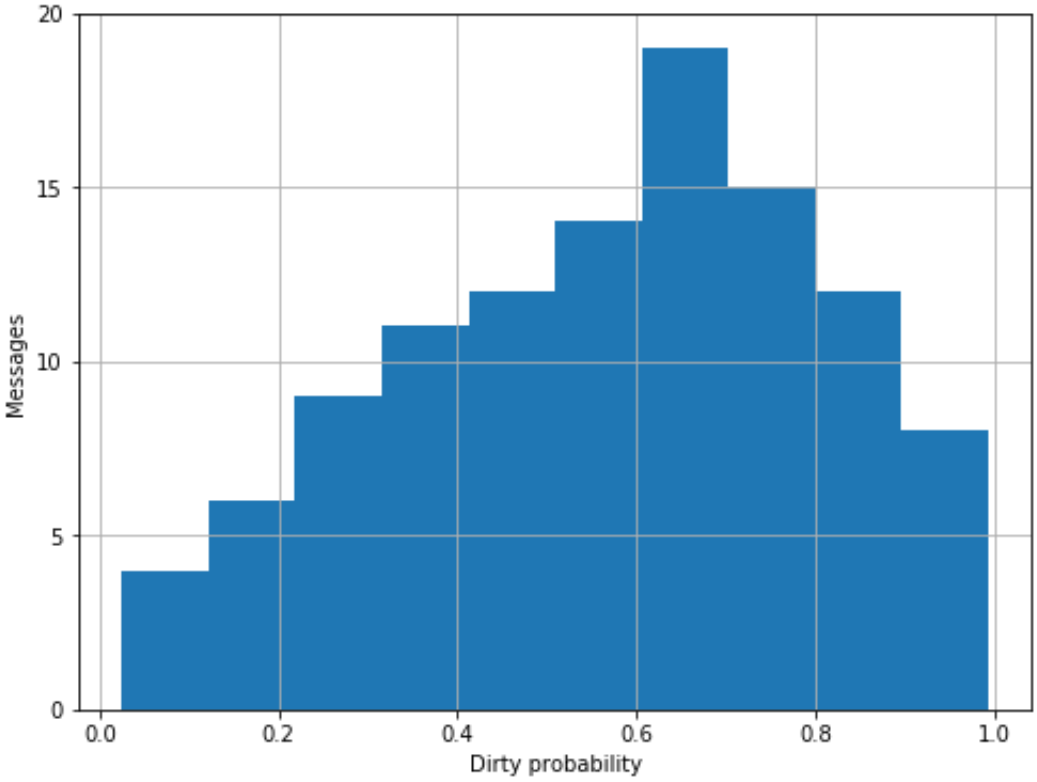


Figure A.1: Wrongly predicted messages of the DistilBERT model on the dataset used (339 messages) for comparison to regular expressions. Over 0.5 in probability is a false positive, and below 0.5 is a false negative.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2020-06-12

EXAMENSARBETE Classification of Short Text Messages using Machine Learning
STUDENTER Alexander Goobar, Daniel Regefalk
HANDLEDARE Pierre Nugues (LTH), Jianhua Cao (Sinch), Michael Truong (Sinch)
EXAMINATOR Jacek Malec (LTH)

Identifiering av oönskade meddelanden med maskininlärning

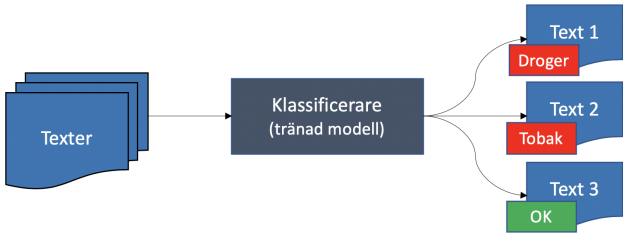
POPULÄRVETENSKAPLIG SAMMANFATTNING **Alexander Goobar, Daniel Regefalk**

Intresset för användning av maskininlärning för automatisk textklassificering har vuxit i takt med de stora framstegen som gjorts inom området. Detta arbete undersöker vilka modeller som passar bäst för att klassificera mycket korta texter, exempelvis för identifiering av oönskade SMS.

Maskininlärning har under senare år fått en allt större roll både inom forskning samt för kommersiellt bruk. Det finns olika typer av maskininlärning, där djup maskininlärning med neurala nätverk på senare tid hamnat i fokus. Det finns även mer klassiska algoritmer, som i större grad bygger på traditionell statistik. Inom djup maskininlärning för språkbehandling har det skett stora genombrott på kort tid, där mer komplexa modeller kan tränas och lära sig generella språkegenskaper på stora textmassor, och sedan finjusteras för det önskade användningsområdet.

I detta examensarbete har vi utvärderat olika metoder för automatisk klassificering av korta texter, både med klassiska algoritmer och toppmoderna djupinlärningsmodeller. Som data för vår undersökning har vi använt tre dataset. Två av dessa är publika och innehåller nyhetsartiklar samt kommentarer från Wikipedia. Vi använde även data från företaget Sinch (där arbetet utfördes) som bestod av B2C SMS, d.v.s. SMS som skickas från olika företag till konsumenter. Mer specifikt undersökte vi hur modellerna kan identifiera meddelanden med oönskat innehåll, t.ex. försäljning av tobak eller alkohol. För alla dataset användes endast texter med färre än 160 karaktärer, vilket är begräsningen för ett enstaka SMS.

För varje dataset tränades modellerna först på 80% av den tillgängliga datan. När träningen var klar utvärderades modellernas prestanda mot resterande 20%, så att de testades mot för modellerna tidigare obekant data.



Resultaten visar att de moderna djupinlärningsmodellerna presterar bra på alla dataset. På de två publika dataseten presterar de i särklass bäst, medan vissa traditionella algoritmer presterar likvärdigt på SMS-datan. Detta förmodas bero på att de generella språkegenskaperna från grundinlärningen hos de moderna modellerna inte kan appliceras i samma utsträckning för B2C SMS, där språkbruket avviker med exempelvis förkortningar och sifferkoder. Alla modeller uppvisade en förbättring gentemot den nuvarande lösningen för blockering av oönskade meddelanden på Sinch, som baseras på nyckelord.