MASTER'S THESIS 2021

Evaluation of Active Learning Strategies for Multi-Label Text Classification

Henric Zethraeus, Philip Horstmann

ISSN 1650-2884
LU-CS-EX: 2021-10

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2021-10

Evaluation of Active Learning Strategies for Multi-Label Text Classification

Henric Zethraeus, Philip Horstmann

Evaluation of Active Learning Strategies for Multi-Label Text Classification

Henric Zethraeus
mas15hze@student.lu.se

Philip Horstmann
mas15pho@student.lu.se

August 1, 2021

Master's thesis work carried out at Sinch AB.

Supervisors: Michael Truong, Michael.Truong@sinch.com
Pierre Nugues, Pierre.Nugues@cs.lth.se

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

Abstract

With increasing data flows from cloud communication services, unlabeled data has become abundant; however, labeled data remains scarce. Commonly in machine learning, annotators will label a portion of randomly sampled data for the model to be trained on, which is usually expensive, sub-optimal, and time consuming. Active learning is an approach in machine learning where the model decides which samples are to be labeled from a set of unlabeled instances. In this thesis, we initially investigate a suitable machine learning model for mutil-label classification of text messages, and then evaluate different active learning strategies. Our findings show that a logistic regression model with an active learning strategy based on a minimum confidence, average, non F1 macro score weighting created the best overall results with a fast learning rate and the highest F1 max score. This strategy queries samples with the lowest confidence, averaged over all class labels of each document, along with treating each class label independent of F1 performance.

Keywords: MSc, Machine Learning, Natural Language Processing, Active Learning, Labeling, Multi-Label Classification

Acknowledgements

We would like to thank Sinch for an interesting topic to our thesis and providing us with material to complete it as well as a welcoming office space. We extend a big thanks to our supervisor Michael Truong at Sinch, for constantly going the extra mile to help us in our project, either with general guidance or intricate programming problems. We would also like to thank our supervisor Pierre Nugues at the Department of Computer Science at LTH, who assisted us with great feedback and ideas throughout the thesis, always positive and solution oriented. Finally we are grateful to experts in the field Andrea Esuli and Fabrizio Sebastiani, both from the National Research Council of Italy, and Tivadar Danka developer of ModAL, for contributing with thoughtful insights to our results.

Contents

Chapter 1
Introduction

千日の勤学より一時の名匠
"Better than a thousand days of diligent study is one day with a great teacher."
– Japanese proverb

1.1 Background

The ability to harness the capabilities of data driven decision making is now widely accepted as central to the success of the tech industry. Tech giants such as Amazon, Google and Facebook all depend heavily on the continuous input and efficient utilization of data (Labrinidis and Jagadish, 2012; Chollet, 2018). This shift towards a larger throughput of data has been named *big data*. Big data differs from the traditional data analysis environment in that it focuses on flows of data instead of stocks and because it relies on data scientists instead of data analysts (Davenport et al., 2012).

However, data flows cannot be handled efficiently without initially developing models that are based on stocks of data. In "How is 'Big Data' different" in the MIT management Sloan Review 2012, the authors of the article write that stocks of data are still "useful for developing and refining the analytical models used on big data, once the models have been developed, they need to process continuing data streams quickly and accurately" (Davenport et al., 2012). Understanding how to develop a model from stocks is thus crucial to the management of future flows of data.

Sinch is a company that provides cloud communication solutions for its customers, mainly by offering efficient global distribution of SMS messages. Sinch is an international organization and has been expanding rapidly over the last years and thus, the volume of distributed text messages has increased as well. This increase in data flow could help Sinch gain a deeper understanding about market trends and customer communication habits, which in turn could help Sinch make informed decisions and to adapt to market change. However, because of the sheer volume of text messages and the continuous change in data, monitoring

Sinch's data is not a straightforward task.

Currently, Sinch distributes over 100 billion text messages every year, making it impossible to manually check and label every message. Moreover, the use of Application Service Providers as customers prevents Sinch from having direct contact with its end consumers and the messages they send. Sinch thus suffers from a common modern machine learning problem. As Settles (2009) puts it, "unlabeled data may be abundant or easily obtained, but labels are difficult, time-consuming, or expensive to obtain."

A common strategy for annotation of machine learning data sets is to randomly sample a suitable number of data instances for training. However, this suffers from two problems:

1. It is time consuming and expensive because the unlabeled instances may not be selected in the most efficient way, causing annotators to label more than required.

2. The data instances which the model needs to train on may not be picked and the already learned categories may be redundantly chosen, thus resulting in a poorly performing classifier.

The purpose of *Active learning* is to get around these problems. (Settles, 2009)

Active learning refers to a method where the *learner* (classifier), given a training set, may request *queries* on which additional instances the *oracle* or *teacher* (e.g. human annotator) should label from a set of unlabeled instances. The idea is that the model can achieve greater accuracy with fewer training examples by making informed decisions on which instances to label rather than randomly selecting them (Settles, 2009).

1.2 Goal & Limitations

The purpose of this master thesis project is to produce a suitable machine learning model for the multi-label classification of Sinch's text messages. By making use of data sets of previously distributed text messages, the goal is to create a model that can perform satisfactory on the initial data sets and will be able to interpret and adapt to the changing market.

More specifically, the project will focus on incorporating *active learning* into the model in order to reduce the amount of manual labeling required for new data, while still maintaining satisfactory accuracy and speed of the classification. Different theories within active learning propose different strategies of the method and our goal is to evaluate which strategy offers the best solution for Sinch; and if *active learning* is better than random sampling at all.

Our main limitation in the project is the time limit. The short span requires us to limit the size of the data sets so that labeling does not take up too much time. Moreover, the models selected must be relatively time efficient in both training and prediction which narrows the scope of possible machine learning models. Due to this, the project will not delve very deep into the finding of the perfect text classifier, but will instead settle for a model with satisfactory results so that focus can be put on evaluating active learning.

Another limit to the scope of this thesis relates to the data sets we chose to work with. It must be emphasized that the goal of this thesis was not to create an optimal machine learning algorithm for the classification of Sinch's text messages, but rather to investigate the possibility of using active learning as a technique for adapting to the changes in the communication data flow. Therefore, the data sets were not overly tampered with and preprocessed, but on

the contrary, consciously kept as close to the actual flow of data as possible so as to let the active learning algorithm make itself apparent in a realistic environment; hence, the imbalance of the data sets.

1.3 Mathematical Notation

Throughout this report some mathematical notation will be recurring. To ease further reading, Table 1.1 contains a summary of the notations used. The notations are inspired by those used in Zhang and Zhou (2013) to create conformity with the other literature in the field.

It should be noted that the split into test and training sets in the notations in Table 1.1 – $\mathcal{D}$ and $\mathcal{S}$ – will only be used in mathematical context. When executing the experiments, we used a k-fold iteration, so there is no permanent real distinction between train and test sets in practice. Therefore, when we refer to the data sets in general, it is to be understood as $\mathcal{D}$ and $\mathcal{S}$ combined.

Table 1.1: Mathematical notations used throughout the report along with their corresponding meanings.

Notation	Mathematical meaning
$\mathcal{X}$	d-dimensional feature space, $\mathbb{R}^d$
$\mathcal{Y}$	label space with q possible labels, $\{y_1, y_2, ..., y_q\}$
$\mathbf{x}$	d-dimensional feature vector $(x_1, x_2, ..., x_d)^\intercal$, $\mathbf{x} \in \mathcal{X}$
Y	set of labels associated with a feature vector $\mathbf{x}$, $(Y \subseteq \mathcal{Y})$
$\mathcal{D}$	multi-label training set $\{(\mathbf{x}_i, Y_i) \vert 1 \leq i \leq m\}$
$\mathcal{S}$	multi-label test set $\{(\mathbf{x}_i, Y_i) \vert 1 \leq i \leq p\}$
$g(\cdot)$	binary classifier $g : \mathcal{X} \rightarrow y$, where g returns prediction of $\mathbf{x}$ belonging to label y
$h(\cdot)$	multi-label classifier $h : \mathcal{X} \rightarrow 2^{\mathcal{Y}}$, where h returns the predicted labels for $\mathbf{x}$
$f(\cdot, \cdot)$	real valued function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$, $f(\mathbf{x}, y)$ returns the confidence of $\mathbf{x}$ being a member of class y
$t(\cdot)$	threshold function, $t : \mathcal{X} \rightarrow \mathbb{R}$, where $h(\mathbf{x}) = \{y \vert f(\mathbf{x}, y) > t(\mathbf{x}), y \in \mathcal{Y}\}$
$\mathbf{y}$	q-dimensional label vector denoting the proper labels for $\mathbf{x}$
$\hat{\mathbf{y}}$	q-dimensional label vector denoting the estimated labels for $\mathbf{x}$

Chapter 2

Data Sets

This chapter will cover the data sets that were used throughout the project. The sections will specifically go over the different data sets, what they were used for, their characteristics, and how they differ. A special attention will be put on a general understanding of the specific data that Sinch handles and provided to us, which is central to the project. Moreover, this chapter will also cover the labeling of the Sinch data, since it is needed to provide a comprehensive exploratory data analysis in the last sections of the chapter.

2.1 Sinch Data

Since Sinch mainly distributes SMS and similar formats (although not exclusively), this project will focus on a data set composed of short text messages. This type of data is, in many ways, convenient since the messages usually follow some form of template and are not very complex in their nature. To protect the privacy of all parties involved, personal and company specific information have been redacted from the sample messages in this report.

2.1.1 Sinch Set 1

The first Sinch data set – from now on referred to as S1 – is a subset of a collection of text messages distributed in the US market during three hours. In order to efficiently train and evaluate models in reasonable time, the data set was cut to 10k messages using random sampling so that the labeling process would not take up too much time. The primary use of this data set was to test and train the baseline model.

2.1.2 Sinch Set 2

The second Sinch data set (S2) is a set of messages distributed in 2019 and was used by Tran and Truong (2019)[1]. S2 is similar to S1, however, none of the messages are identical. One of the main differences between this data set and the first Sinch data set is that S2 uses named entity recognition, which means that some parts of the messages have been replaced by entities such as DIGITS or REDACTED. The primary use of S2 is to see how well the model trained on S1 can adapt to S2 using active learning.

2.1.3 Labeling

The labeling process is essentially part of the preprocessing of the Sinch data sets and will be further explained in the chapter 4 of this study. However, in order to perform a comprehensive exploratory data analysis, information on the labels of S1 and S2 is required.

13 labels in total were selected for the Sinch data sets. Each message belongs to *at least* one of the categories. None of the categories are mutually exclusive except for *Alert* and *Notification* (11 and 12, respectively). The labels Alert and Notification relate to the urgency, need for action or response, and anticipation of the text message. If a message is deemed to belong to *at least* two of the subcategories *urgent*, *action required*, and/or *not expected*, it is given the label *Alert*. However, if it only fulfills the requirements for one *or less* of the subcategories *urgent*, *action required*, and/or *not expected*, it is given the label *Notification*. See Table 2.1 for the full list of labels and short explanations of them.

Table 2.1: Labels used on the Sinch data sets.

Number	Label	Short explanation
0	**Banking**	Personal finances such as transaction or balance inquiries
1	**Transportation**	Transportation information such as flight schedule updates
2	**Two Factor Authentication**	Account login information such as a one time password
3	**Marketing**	Commercial content
4	**Finance**	Financial information such as stock market updates
5	**Social**	Messages referring to social, personal or spare time recreations
6	**Non-English**	Messages in other languages
7	**Job and Education**	Messages with information about jobs or education
8	**Service**	Information about services such as car reparations
9	**Security and Monitoring**	E.g. house alarms or GPS trackers
10	**Public Information**	Information distributed to certain areas such as earthquake warnings
11	**Alert**	If $\geq$ **2/3** of urgent, unexpected, and action required are satisfied
12	**Notification**	If $<$ **2/3** of urgent, unexpected, and action required are satisfied

To get a better understanding of how the messages are labeled and, in particular, how to differentiate between *Alert* and *Notification*, Table 2.2 shows four sample messages and their labels.

Take message 0 in Table 2.2 as an example. The message is providing a verification code for an account and is therefore labeled *Two Factor Authentication* (2FA). Also, although the message is probably *expected*, it does *require action* and is *urgent* – since the verification code will only last temporarily – therefore, it is labeled as an *Alert*. Message 1, on the other hand,

[1]Michael Truong was our supervisor at Sinch.

is classified as a *Notification*, because although this message *requires action* to login, it is not *unexpected*, since the customer most likely asked to retrieve login information, and contrary to message 0, it is not *urgent*, since a password does not have a time restriction the way a verification code does. Thus, for message 0, more than 2 out of 3 requirements are satisfied → *Alert*, for message 1, less than 2 out of 3 requirements are satisfied → *Notification*.

Similarly, both message 2 and 3 relate to banking. However, message 2 refers to a debit card being declined, which is probably both *unexpected* and *requires action* – if the customer wants to make the payment at COMPANY (it might even be *urgent*) – therefore, the message is labeled *Alert*. Message 3 is neither *unexpected*, *urgent* nor *action required* and is therefore labeled *Notification*.

Table 2.2: Sample messages from Sinch data with corresponding labels. The company names, customer names, and account information, have been exchanged for arbitrary entities.

ID	Message	Labels
0	A-123456 is your COMPANY verification code.	2FA; Alert
1	Account notification: The password for your COMPANY Account 123456	2FA; Notification
2	Debit Card 1234 A charge of $ 313.86 was declined at COMPANY on 10/15 02:06 AM ET due to NSF.	Banking; Alert
3	Checking Withdrawal posted for $ 192.84	Banking; Notification

2.2 Reuters Data Set

In order to get a better understanding of to what degree the performance of our models depends on their inherent capabilities or the properties of the Sinch data, a third reference data set was introduced for cross evaluation. For this purpose, the Reuters RCV1-v2 data set was chosen. V2 in RCV1-v2 stands for version 2, which is the version used in this thesis, but hereinafter the data set will only be referred to as RCV1.

RCV1 is composed of 800,000 news wire stories written between 1996 and 1997. The news wire stories vary in length between a couple of hundred words to several thousands and are written in English. Each story has been given *at least* one label in each of the following category sets: Topics, Industries, and Regions. For our comparison, only the set of 103 occurring Topic codes was used, and Industries and Regions were neglected.

The reason for choosing the RCV1 is that it does not suffer from the weaknesses of many other data sets, namely: few documents, limited documentation of categorization, incomplete category assignments, or limited access. Moreover, Lewis et al. (2004) have provided benchmark results for NLP research on the set, which could be used for comparison in this study. (Lewis et al., 2004)

2.3 Exploratory Data Analysis

The exploratory data analysis will delve deeper into the characteristics of the data sets by looking at the composition and lexical properties of the messages and the distribution of labels.

2.3.1 Label Distribution

In order to gain an understanding of the distribution of messages in the different labels, we performed an exploratory data analysis on the S1 and S2 data sets and created the two graphs that can be seen in Figure 2.1 below. The figure shows the number of messages in each category divided by the total number of messages in each data set so that they can be compared to each other. Note that this is a multi-label classification, meaning that each message can belong to more than one category. Therefore, the number of labels in sub-figure (a) is larger than the number of input vectors. The classes *Alert* and *Notification* on the other hand are mutually exclusive, so the number of labels per data set in figure (b) is equal to the number of input vectors.

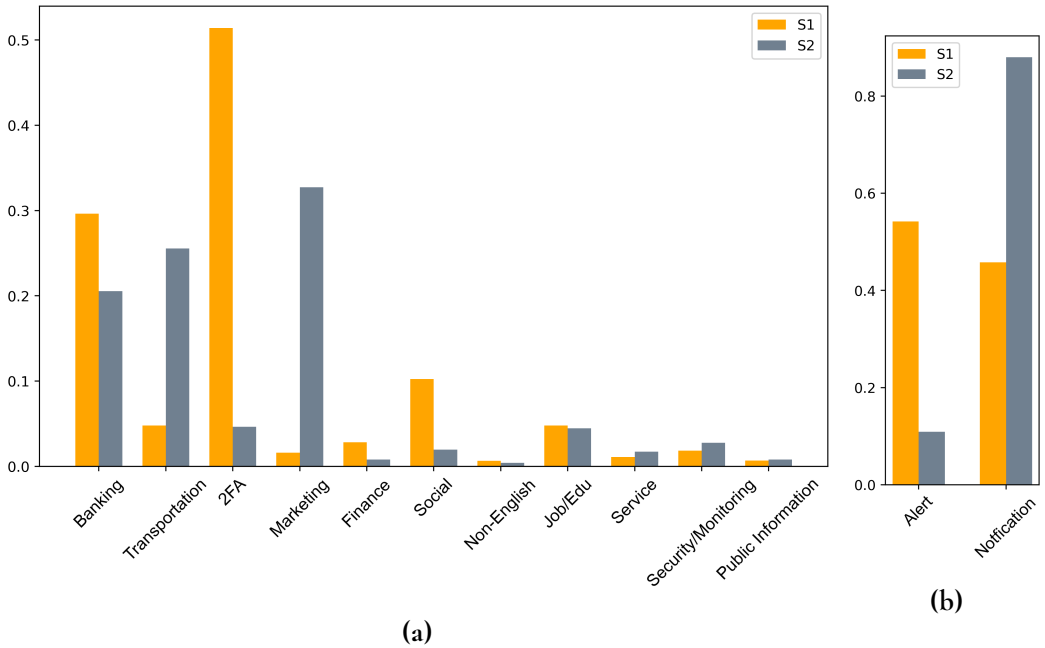


Figure 2.1: Labeled messages in each category per total messages in Sinch set 1 and Sinch set 2

One conclusion that can be drawn from Figure 2.1 is that both data sets are imbalanced. In S1, *Banking* and *2FA* are the most common classes while in S2 *Banking*, *Transportation*, and *Marketing* are the most prevalent. Furthermore, some categories such as *Non-English* and *Public Information* occur very rarely in both data sets. In sub-figure (b), it can also be seen that S2 has a higher volume of *Notification* messages than *Alerts*, while S1 is quite balanced among the two categories. One way of more accurately quantifying the class imbalance of data sets is to use the *Imbalance Ratio*.

2.3.2 Imbalance Ratio

In short, the imbalance ratio of a class label is the ratio between the number of negative samples and positive samples of that class label. In mathematical terms, the imbalance ratio can be explained as follows: If $\mathcal{Y}$ denotes the set of all class labels $\mathcal{Y} = \{y_1, y_2, ..., y_q\}$, and $\mathcal{X} \in \mathbb{R}^d$ denotes the space of possible d-dimensional input vectors, then each feature vector $x_i \in X$, is associated with a set of labels $Y_i \in \mathcal{Y}$ such that the multilabel training set can be described as $\mathcal{D} = \{(x_i, Y_i) | 1 \leq i \leq N\}$, for all N input vectors. For each class label $y_j \in \mathcal{Y}$ the set of positive and negative samples of that class label are denoted $\mathcal{D}_j^+ = \{(x_i, +1) \mid y_j \in Y_i, 1 \leq i \leq N\}$ and $\mathcal{D}_j^- = \{(x_i, -1) \mid y_j \notin Y_i, 1 \leq i \leq N\}$, respectively.

The *imbalance ratio* of one class label is then defined as:

$$ImR_j = max(\ |\mathcal{D}_j^+|,\ |\mathcal{D}_j^-|\ ) / min(\ |\mathcal{D}_j^+|,\ |\mathcal{D}_j^-|\ ) \tag{2.1}$$

To get the *average imbalance ratio* Equation 2.1 is just summed over and divided by all class labels as follows:

$$ImR = \frac{1}{q} \sum_{j=1}^{q} ImR_j \tag{2.2}$$

By using Equation 2.1, the imbalance ratios of the individual class labels of S1 and S2 could be calculated as seen in Table 2.3.

Table 2.3: Imbalance ratios for individual labels for S1, S2 and the two data sets combined. The labels have been abbreviated for formatting reasons, see Table 2.1 for full list of labels with explanations.

Data Set	Bank	Trans	2FA	Mark	Fin	Soc	N.Eng
S1	2.4	19.8	**1.1**	61.9	34.5	8.8	**155.3**
S2	11.2	8.8	52.8	6.6	311.5	127.2	**587.2**
S1+S2	2.7	8.3	1.6	8.5	43.6	11.7	**171.8**
Data Set	**Job**	**Ser**	**Mon**	**P.I.**	**Al**	**Not**	
S1	19.8	89.9	53.1	146.1	1.2	1.2	
S2	54.9	143.9	89.1	311.5	21.9	**1.8**	
S1 + S2	20.2	77.2	46.3	139.0	**1.4**	**1.4**	

Table 2.3 shows that there is a very strong imbalance in both data sets. The boldfaced numbers show the most and least imbalanced labels in each data set, and all three show that *Non-English* and *Notification* are the most and least imbalanced labels, respectively. These maximum and minimum values, along with the average imbalance calculated using Eq. 2.2 are compiled in Table 2.4 below. The table also shows the imbalance values for RCV1, which are presented in Zhang et al. (2020).

By comparing RCV1 to the Sinch data sets, it becomes even more apparent how imbalanced the Sinch data sets are. Moreover, Zhang et al. (2020) compared 13 other multi label data sets and concluded that the maximum label imbalance ranged from 3.0 to 50.0, and the average imbalance across the label space ranged from 2.1 to 17.9. As seen in Table 2.4, S1 and S2 are far more imbalanced in both aspects.

Table 2.4: Minimum, maximum and average imbalance ratios for S1, S2 and the two datasets combined.

Data set	min	max	avg
S1	1.1	155.3	45.8
S2	1.8	587.2	133.0
S1 + S2	1.4	171.8	41.1
RCV1	3.5	47.6	15.9

2.3.3 Lexical Properties

In this section, we looked at the lexical properties of the different data sets. This was done in order to see if there were any patterns that showed correlation between the lexical complexity of the data sets and their individual classes and the performance of the active learning.

First, we tested the word count of each data sets and their classes. In Table 2.5 below, the word count for S1, S2, S1 and S2 combined are shown. The table highlights both the highest word count and standard deviation as well as the lowest word count and standard deviation.

The classes with the highest and lowest word counts and standard deviations for S1 were *Service* (22.5 ± 5.5) and *Two Factor Authentication* (8.7 ± 5.2), respectively; for S2 they were *Public Information* (25.2 ± 23.8) and *Non-English* (10.2 ± 4.9), respectively; and for S1 and S2 combined, they were *Marketing* (22.5 ± 6.1) and *Two Factor Authentication* (9.0 ± 5.5).

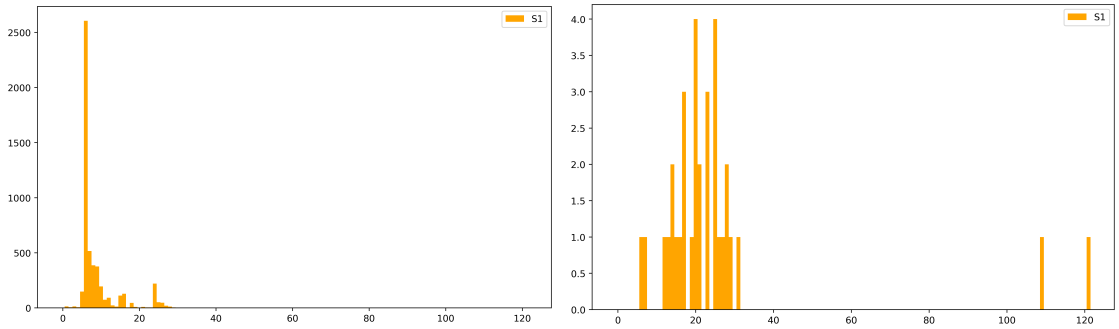
Table 2.5: Average word count per class label and their standard deviations for S1, S2, and S1+S2. The class with the highest word count per data set is boldfaced while the lowest word count per data set it italicized.

Data Set	Bank	Trans	2FA	Mark	Fin	Soc	N.Eng
S1	16.3 ±6.2	16.7 ±7.5	*8.7 ±5.2*	20.1 ±6.4	15.7 ±7.3	11.9 ±2.8	16.7 ±5.7
S2	18.6 ±6.1	18.7 ±6.1	16.7 ±7.5	22.8 ±6.0	17.9 ±6.4	18.6 ±10.2	*10.2 ±4.9*
S1+S2	16.8 ±6.3	18.1 ±6.6	*9.0 ±5.5*	**22.5 ±6.1**	16.0 ±7.1	12.4 ±4.2	15.6 ±5.9
Data Set	**Job**	**Ser**	**Mon**	**P.I.**	**Al**	**Not**	
S1	13.1 ±6.5	**22.5 ±5.5**	20.7 ±7.0	12.1 ±5.6	9.9 ±6.4	13.8 ±6.0	
S2	17.2 ±12.0	18.6 ±6.6	23.6 ±4.6	**25.2 ±23.8**	20.0 ±10.7	20.0 ±6.7	
S1+S2	14.2 ±8.6	21.2 ±6.0	21.9 ± 6.2	16.6 ±15.6	10.7 ±7.3	16.5 ±7.0	

What is immediately striking is that there is a big difference between the data sets. Apart from *Two Factor Authentication* which occurs as the lowest word count per message twice, the other categories are unique for each data set.

Figure 2.2 was created to illustrate the difference between the highest and lowest word counts per message out of the six aforementioned class word count distributions – namely *Two Factor Authentication* for S1 and *Public Information* for S2. What can be seen in Figure 2.2 is that there is a clear spike of word count per label at 6 words. This is because of a common template of verification messages that Sinch distributes which occur frequently in the data

set; the message template can be found as message with ID 0 in Table 2.2. Because of this common template and that there are no outliers in the distribution, the word count and standard deviation remains low. For *Public Information* in S2, on the other hand, there are a couple of outliers far away from the main distribution, as well as a broader distribution of word counts around 20 words per message, which drives up the word count and standard deviation of the class. It should also be noted that the volume of messages on the y-axis are very different, which illustrates the label imbalance that was discussed in the previous section.



(a) Two Factor Authentication in S1

(b) Public Information in S2

Figure 2.2: Distribution of word count per message for the lowest and highest out of the classes in data sets S1 and S2

Next, we calculated the lexical diversity of the class labels in the individual texts. Initially, the messages had to be divided into separate words and stems, which is done using tokenization. There are several methods for text tokenization that suite different applications and tools. For instance, DistilBERT – which was used later in this project – uses one, which is a separate method than the one used here. The tokenization used here was the one supplied by Kristopher Kyle specifically for *Lexical Diversity* analysis (Kyle, 2020). The tokenization used for subsequent classification is discussed in further detail in Section 3.4.2.

After tokenizing the messages, lemmatization is usually applied for calculating lexical diversity (Plisson et al., 2004). Lemmatization refers to the task of finding the normalized form of a word. For example words like *compute*, *computing*, and *computed* would all be normalized as *compute*. The lemmatization applied here is also the one supplied Kristopher Kyle (Kyle, 2020). After the lemmatization of the messages, their lexical diversity could then be calculated using the Text-Type Ration (TTR) defined below

$$TTR_q = \frac{number\ of\ types}{number\ of\ tokens}. \tag{2.3}$$

E.g. the sentence "We like active learning" would equal 4 tokens and 4 different types, thus generating a lexical diversity of 1. The addition of the sentence "We like dancing" to the set, would add 3 tokens, but only add 1 more type, thus generating a lexical diversity of $5/7 = 0.7$ The TTR scores calculated using Eq. 2.3 range from 1 to 0, where 1 denotes the highest possible lexical diversity and 0 the lowest.

For the computation of the TTR, the lemmatized messages of each class label q were added together to form one large set of words. Another approach would be to compute the TTR of each individual message and then take the average of them. However, since every message is composed of a few tokens of almost the same number of types, each message would

receive a TTR score close to one, and the average value would also be close to 1. Therefore, we decided to aggregate the words of the messages to compute the lexical diversity of the whole class. The results of TTR computations can be found in Table 2.6 below.

The table shows that *Non-English* has the highest lexical diversity in both of the data sets and the combined data set. *Alert* and *Notification* have the lowest scores, closely followed by *Two Factor Authentication*.

Table 2.6: Lexical diversity per label for data sets S1, S2, and S1+S2. The classes with the highest and lowest lexical diversity are boldfaced.

Data Set	Bank	Trans	2FA	Mark	Fin	Soc	N.Eng
S1	0.176	0.241	0.162	0.325	0.278	0.212	**0.409**
S2	0.203	0.185	0.279	0.164	0.459	0.361	**0.480**
S1+S2	0.171	0.197	0.161	0.198	0.269	0.208	**0.388**
Data Set	**Job**	**Ser**	**Mon**	**P.I.**	**Al**	**Not**	
S1	0.240	0.370	0.321	0.404	**0.161**	0.165	
S2	0.282	0.378	0.329	0.459	0.229	**0.129**	
S1+S2 tot	0.221	0.317	0.274	0.379	0.161	**0.152**	

As a final remark to the exploratory data analysis, it can be seen that there is a strong correlation between the most imbalanced classes and the classes with the most lexical diversity. This is a result of Sinch's distribution model. Since most messages follow a specific template and are distributed in high volumes, a higher volume of messages in a class will not result in a particularly high increase in types of tokens. Therefore, the higher the volume, the lower the TTR score.

Chapter 3
Theory

This chapter will present the theory and related works that we used to complete this thesis. The chapter will begin with a brief introduction to artificial intelligence and machine learning in order to later explain multi-label classification. The chapter will then continue by explaining the machine learning methods used and further move on to natural language processing and the theory related to that. It will end by explaining the theory behind active learning and scoring metrics used to evaluate the methods.

3.1 Artificial Intelligence

Artificial intelligence (AI) can vaguely be described as the field whose objective is to make computers "think." More concisely, the field can be defined as "the effort to automate intellectual tasks normally performed by humans" (Chollet, 2018, p. 4). Initially artificial intelligence was performed by hard-coding explicit rules into the computer, which is now referred to as *symbolic AI*. However, as the field progressed it was discovered that symbolic AI was insufficient at solving more complex and vaguely defined problems. As a consequence, machine learning and later deep learning were developed as sub-fields of AI (Chollet, 2018).

3.2 Machine Learning

The purpose of machine learning is to make computers learn how to perform tasks on their own, in contrast to symbolic AI, where the rules of the computers are stated explicitly (Chollet, 2018). In machine learning, the computer is *trained* on data instead of explicitly programmed. The difference between classical programming (and symbolic AI) and machine learning is illustrated in Figure 3.2 below. In symbolic AI and classical programming, the computer is fed rules and data and then provides answers as output. In machine learning, however, the inputs of the computer are data and answers (i.e. labeled data), and the output

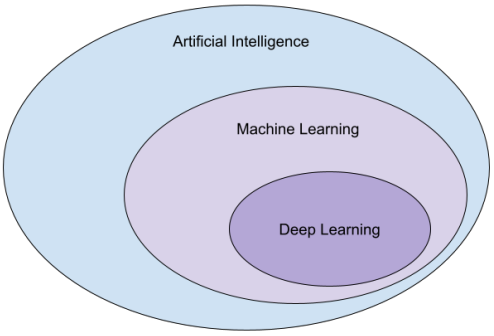


Figure 3.1: Figure of artificial intelligence with machine learning and deep learning as sub-fields

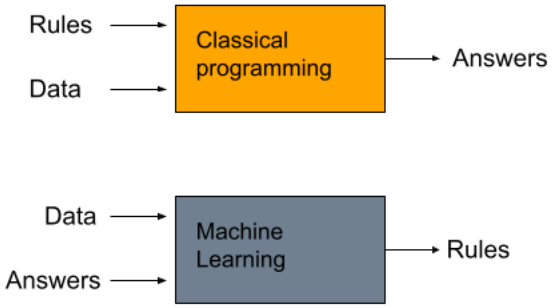


Figure 3.2: Comparison of classical programming and machine learning.

is rules. This is called the training phase of machine learning. The machine is given many solved samples of data and finds a statistical structure in the correct output from which it can create rules to automate the task. These rules that the machine learns during the training phase can then be applied to new (unlabeled) data, and thus provide answers without being explicitly taught how to do so (Chollet, 2018).

3.2.1 Single- and Multi-Label Classification

Classification refers to the task of assigning a set of one or more class labels $Y_i = \{y_1, y_2, ..., y_q\}$ to an input of feature vectors $\mathbf{x}_i \in \mathcal{X}$. If the feature vector $\mathbf{x}_i$ can be assigned to one and only one class label in the label space, the concept is called *single label classification*. The classification is done through a function

$$g : \mathcal{X} \rightarrow \mathcal{Y} : \mathbf{x} \mapsto y \tag{3.1}$$

If the label space only consists of one class label and is strictly binary, i.e. $\mathcal{Y} = y = \{0, 1\}$, it is called a *binary classifier* and the function g is then mathematically described as:

$$g : \mathcal{X} \to \{0, 1\} : \mathbf{x} \mapsto y \tag{3.2}$$

where 1 denotes member of y and 0 denotes *not* member of y (Gerniers and Saerens, 2018).

If, however, $\mathbf{x}$ can belong to a set of one *or more* class labels $Y_i = \{y_1, y_2, ..., y_q\}$, it is called *multi-label classification*. In a multi label classification problem, the computational requirements are increased exponentially, since every feature vector $\mathbf{x}_i$ can in theory be assigned to any combinations of labels $y \in Y$. Or as Gerniers and Saerens put it:

> The fact that multiple labels can be associated to one object constitutes a huge difference with respect to the traditional, single-label, classification task. Indeed, in the latter case, we only need to choose one label among a finite set of labels. The number of possible outcomes that can be given to an observation is thus simply the number of class labels present in the dataset. When the number of labels grows, the number of outputs will grow linearly. This means that single-label classification is scalable to applications with a huge number of class labels. This is not the case with multi-label classification. (Gerniers and Saerens, 2018)

The possible output of a multi-label classifier is thus $2^{\mathcal{Y}}$. If the multi-label classifier is denoted h the classifier can be defined as

$$h : \mathcal{X} \to 2^{\mathcal{Y}} : \mathbf{x} \mapsto Y \tag{3.3}$$

The function h is trained on a set of labeled instances:

$$\mathcal{D} = \{(\mathbf{x}_1, Y_1), (\mathbf{x}_2, Y_2), ..., (\mathbf{x}_m, Y_m)\} \tag{3.4}$$

This problem with exponentially increasing combinations of label sets becomes even more computationally difficult if there exists dependence between labels. Therefore, for most multi label classifiers, the model assumes independence between labels for efficiency. The most common method for addressing multi label classification problems is through *binary relevance*.

3.2.2 Binary Relevance

Binary relevance or *one versus rest* can simply be viewed as several single label classifiers stacked on top of each other. For each feature vector $\mathbf{x}_i \in X$, the classifier considers the possibility of $\mathbf{x}_i$ belonging to label y_j and then repeats the process for every class $y_j \in \mathcal{Y}$. In other words, the multi-label classifier h is constructed of q independent binary classifiers g so that each binary classifier determines the possibility of $\mathbf{x}_i$'s membership in y_j:

$$g_i : \mathcal{X} \to \{0, 1\} \tag{3.5}$$

Put together, the outputs of all classifiers g_i in h creates an output vector $\hat{y}$:

$$h : \mathcal{X} \to \{0, 1\}^q : x \mapsto \{g_1(\mathbf{x}), g_2(\mathbf{x}), ..., g_q(\mathbf{x})\} \tag{3.6}$$

Although binary relevance is a common way of solving the problem with multi-label classification, there are many other frequently used methods. Some of these are: decision trees, support vector machines and boosting. In the next section we will briefly describe the classifiers used in this project.

3.2.3 Classifiers

We have used five different machine learning classifiers in this project: logistic regression, Support Vector Classifier, Linear Support Vector Classifier, RandomForest and XGBoost. We briefly explain these five methods below. For the enthusiastic reader, we suggest to read the sources referenced.

Logistic Regression (LogReg)

Chollet (2018) states that the logistic regression model (LogReg) is commonly said to be the "hello world" of modern machine learning and explains that LogReg predates computing but because of its uncomplicated and adaptable nature it is still widely used today. In short, Schein and Ungar (2007) explain the model as follows. Given a set of predictors, $\boldsymbol{X}_n$, we can determine the probability of a binary output y_n by Equation 3.7 below:

$$P(Y_n = 1|\mathbf{x}_n) = \sigma(\mathbf{w} \cdot \mathbf{x}_n) \tag{3.7}$$

where $\mathbf{w}$ denotes the weight for each part of the vector $\mathbf{x}_n$ and σ is expressed by equation 3.8

$$\sigma(\theta) = \frac{1}{1 + exp(-\theta)} \tag{3.8}$$

mapping any real valued θ into the interval (0, 1).

For multi-label classification, a LogReg model is trained for each individual class y_j in $\boldsymbol{\mathcal{Y}}$ using binary relevance. These models together create the output vector $\hat{y}$.

Support Vector Classifier (SVC)

The aim of support vector classifiers (SVC), also known as support vector machines, is to classify samples by computing a decision border between samples that belong to different classes. Once a decision border has been computed a new sample can be classified by checking which side of the border it is located on.

The classifier computes the boundary border by first mapping the data to a new higher dimensional space, according to $\boldsymbol{\phi}$, the dimensions in the higher dimension is equal to the amount of attributes the classifier is to classify on. The classifier then seeks to create a hyperplane which is of one less dimension than the space. The hyperplane is then optimized to be as far away from the closest samples of different classes. Figure 3.3 shows how the SVC maps the samples in a higher dimension with the use of $\boldsymbol{\phi}$ and then creates the hyperplane in the feature space (Chollet, 2018).

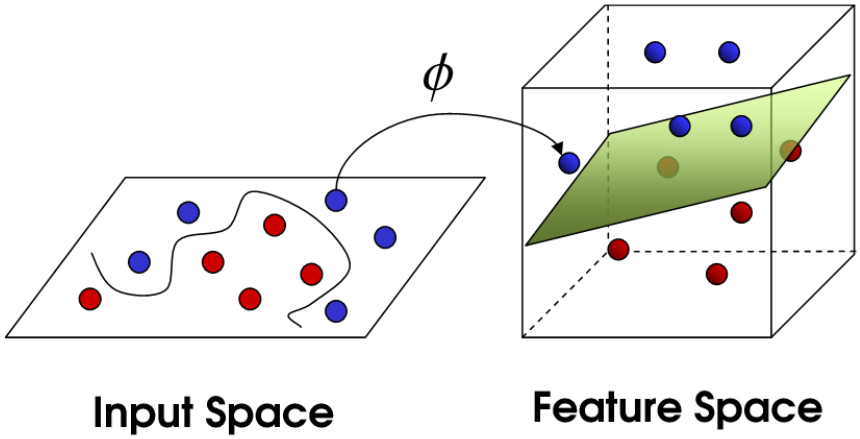


Figure 3.3: The kernel transformation. The figure shows the mapping of features from a two-dimensional plane to a three-dimensional space. (Commons, 2020)

Linear Support Vector Classifier (Linear SVC)

LinearSVC just like SVC seeks to find a boundary that best divides the classes, the difference is that it assumes that the classes are linearly separable. This means that $\phi(x_i) = \mathbf{x}_i$ and no higher dimensional space that handles linearly inseparable data is introduced (Chang et al., 2010).

RandomForest

This method creates a large amount of diverse decision trees, each tree determines an output of a predicted class given an instance of input data. By combining the predictions of these unique and diverse trees, the classifier can create a single consolidated prediction output, resulting in a more robust result than using the decision trees by their own (Breiman, 2001).

Figure 3.4 illustrates simplified how a classification of an instance is made with RandomForest.

XGBoost

XGBoost is a renowned method in the machine learning community and authors Chen and Guestrin (2016) declare that amongst the winners of Kaggle competitions[1] XGBoost is the most common (2015).

XGBoost stands for *eXtreme Gradient Boosting* and is a decision tree ensemble method. It benefits from boosting which is a method to add decision trees to correct errors from previous decision trees. It is optimized through tree-pruning, cache awareness, efficient handling of missing data, paralleled computing and more enhancements on the algorithm compared to predecessors. It will add decision trees sequentially based on the gradient descent of the

[1]Competitions on machine learning problems, more information on `https://www.kaggle.com/`

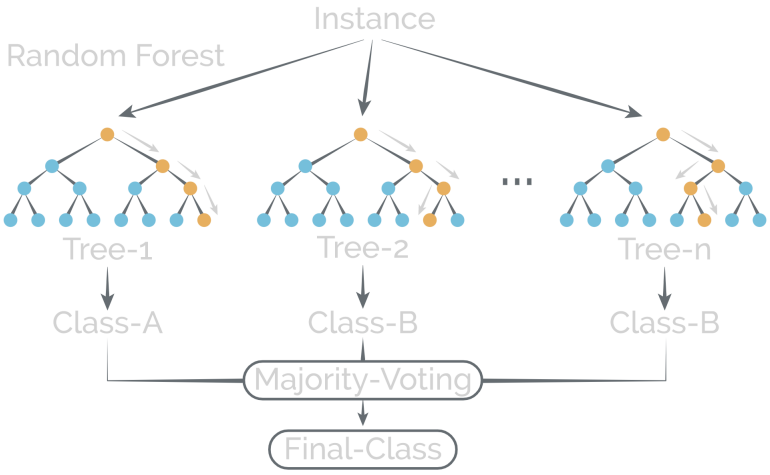


Figure 3.4: Example of a decision by RandomForest. (Koehrsen, 2017)

loss to minimize errors until no more models will further improve the structure. (Chen and Guestrin, 2016)

3.3 Deep Learning

Deep learning is a sub-field of machine learning (see Figure 3.1) which consists of *layers* of representation. Each layer aims to create more meaningful representations of the data until the model can make a conclusive prediction based on the input data. Thus, the deep in deep learning refers to the "depth" of the layers, in other words, how many layers the model consists of. Deep learning is usually carried out using *neural networks*, which are literal layers stacked on top of each other, with variations of connectivity between the layers (Chollet, 2018).

Each layer is connected by weighting the output of each layer to create the input of the next layer. When the model is *trained*, these weights are optimized to make better predictions. To do so, a loss function is required between the models prediction and the true label of the data. The loss function is then used as input to a specific optimizer function, which in turn optimizes the weights of the neural network.

3.4 Natural Language Processing

Natural language processing (NLP) is a cross-disciplinary field of artificial intelligence and linguistics that aims to translate human language to the computer, and vice versa. In her article, Liddy (2001) defines NLP as follows:

> Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications (Liddy, 2001, p. 2).

The complexity of the problem lies within the amount of information behind a simple sentence; the word choices, the order of the words, punctuation, etc. all adds up to a mountain of information that the receiver will interpret and take into consideration in the next interaction. If the receiver is a human this interpretation occurs instantly and almost seamlessly, the human has learnt this during years of interactions. The computer, on the other hand, must process the words into a suitable dictionary of its own.

3.4.1 Word Representation

Chollet (2018) explains two methods of retrieving a dictionary for the computer but with different approaches: One-Hot encoding and Word Embeddings. One-Hot encoding is done by assigning each word with a unique index, i, in a binary vector which is the size of the vocabulary. The word can now be represented by a vector which is filled with zeros except for element i in the vector which is a 1. The One-Hot word vectors are therefore sparse (mostly containing zeros), binary, high dimensional and hard coded, the other method, Word embeddings, is the opposite as shown in Figure 3.5.

Word embeddings, as Chollet explains, are not hard coded but trained on the available data. By creating an embedding matrix with dimensions $M \times N$, where M is the size of the vocabulary and N a chosen dimension (e.g. Google BERT uses 768 dimensions), each word is found in the M dimension being represented by a row in the Nth dimension. The weights are instantiated random and through backpropagation trained and adjusted with the model to reach the models objective, for example classifying text documents.

The result of the backpropagated word vectors are that geometric relationships between the vectors will relate to a semantic relationship between the words. Since the words now are symbolized as vectors in an N-dimesional space, the relations between the words can therefore also be explained via mathematical vector operations.

A common example is to show arithmetic similarity by using addition and subtraction operations on the word embedding vectors of king man and woman. $King - Man + Woman$ results in a vector that is very similar to the vector representing $Queen$. Similarly, the vector operations of $Berlin - Germany + France$ equals a vector close to the one representing $Paris$.

As with most machine learning fields, the more training data, the better the model will perform, therefore when data is not sufficient it is advisable to use word embeddings that have been precomputed on another task. (Chollet, 2018)

3.4.2 Tokenization

Tokenization is a method to break down the words into a vocabulary used in e.g. the above explained Word Embeddings. There are several different methods of tokenization and in this project we used WordPiece created by Wu et al. (2016).

The dictionary of WordPiece is initialized with all the characters available of the corpus that it is to be trained on. It then generates a new word unit by combining two word units from the current dictionary that increases the likelihood on the training data the most when added to a language model of choice. This process is repeated until the dictionary is full (normally between 8-32 thousand units in the dictionary).

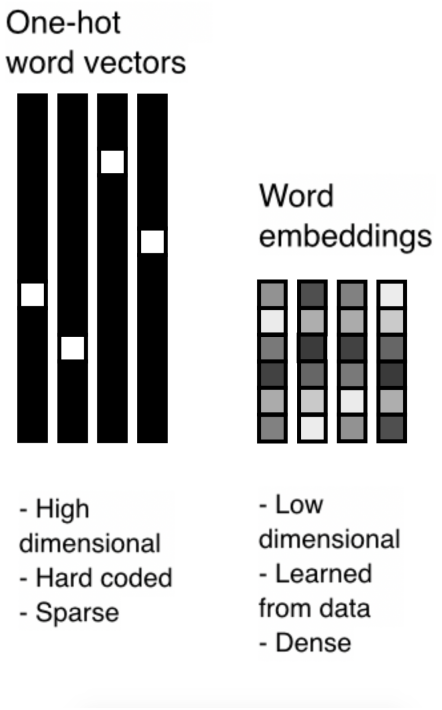


Figure 3.5: One hot encoding compared to word embedding.

When tokenizing on the pretrained WordPiece, it splits each word it is given into pieces until all pieces are part of the dictionary. In addition to the word units in the dictionary, it also contains all Latin characters and therefore a piece can in worst case be broken down into a single character (Wu et al., 2016).

As an example to illustrate how tokenization works we look at the sample message with id 0 from Table 2.2:

[A-123456 is your COMPANY verification code.]

When tokenized with the WordPiece method which is trained with the BERT model with a dictionary of size 30,000 tokens, the words are split up as shown in Figure 3.6. Two additional tokens are added in the beginning and the end, `[cls]` and `[sep]`, which are important to the BERT model and will be explained in Section 3.6. As seen, the number 123456 is not part of the vocabulary and is therefore split up into three parts namely; 123, ##45 and ##6. The ## symbols indicate that the word piece is connected to the word piece to the left of it.

3.5 Transformer

To create the vector representations of the text messages, we used DistilBERT, which utilizes transformers. The transformer introduced by Vaswani et al. (2017) received acclaim for its state-of-the-art results in certain NLP tasks the same year, while simultaneously requiring less training time than other models. The goal of the transformer is to translate a sequence of elements into another sequence of elements, based on how the transformer is trained the elements of input and output are chosen by the user. Commonly is to train it to take as input

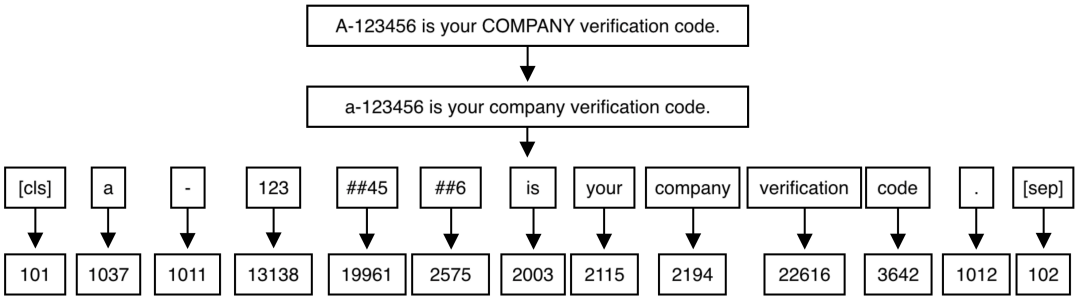


Figure 3.6: Sample visualization of what the tokenization of an input sequence looks like.

a sequence of words and produce a continuous representation of the sequence, as suggested by Allard (2019). As seen in Figure 3.7 is the architecture of the transformer model.

The transformers consists of encoder and decoder blocks that are intertwined as shown in Figure 3.7, the blocks are powered by *Attention* mechanisms. Below are the description of encoder and decoder as well as the Attention mechanism explained.

3.5.1 Encoder-Decoder

In Figure 3.7 of the architecture, the left side in the gray box is the part called the encoder and the right side in the gray box is the part called the decoder. The number of encoders and decoders stacked on one another within a transformer may be any given number N_x – e.g. the paper by Vaswani et al. (2017) used 6. A simplified view of the encoder and decoder and how they are connected is shown in Figure 3.8.

The encoder takes as input a word embedding of a word from a sentence that first is put through a self-attention layer. The encoder uses the self-attention layer to view the different words in the input sentence as it encodes the individual word. The output of this layer is then fed into a feed-forward neural network.

The encoder has now mapped the input sequence of word embeddings to a continuous representation denoted z. The decoder also consists of the Self-Attention and Feed Forward layers, the difference is that between the layers is an Encoder-Decoder Attention layer that helps the decoder focus on relevant parts of the sentence. The decoders object is to decode the representation z to an output sequence.

(Vaswani et al., 2017)

3.5.2 Attention

Attention helps the computer map what words are connected to each other by weighting them as more important to one another. An example could be if a computer is to understand the sentence *"The animal didn't eat the meat, because it was not hungry"*. It could be difficult for a computer to understand whether the word *"it"* refers to *"the meat"* or *"the animal"*. Here the attention method would put more weight on *"the animal"* when encoding the word *"it"*.

The attention functions by mapping vectors called query (q), key (k) and value (v) to an

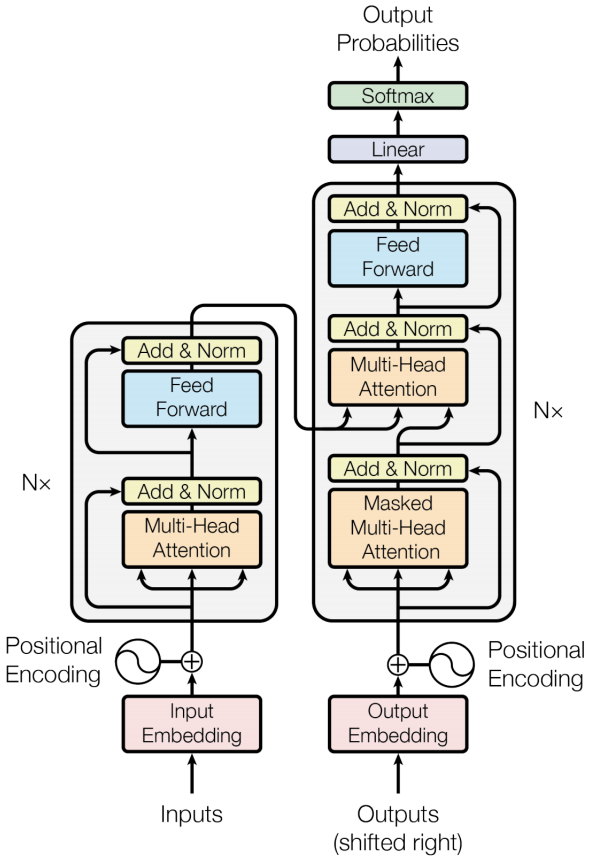


Figure 3.7: The architecture of the transformer. (Vaswani et al., 2017)

output vector. The output vector is calculated as a weighted sum of the values where the weight is dependent of the query with corresponding key. Vaswani et al. packs all the values into matrices Q, K and V to compute them in batches to be more effective. The attention is calculated by Equation 3.9, where we compute the dot product of Q and K and pass it to a softmax function. This term serves as a multiplier to the V matrix. The result is an output matrix.

$$Attention(Q, K, V) = softmax(\frac{Q \cdot K^T}{\sqrt{d_k}}) \times V \tag{3.9}$$

where d_k is the dimension of the key vector.

In the architecture they have used this function in three different ways:

Encoder-Decoder Attention In this layer, the keys and values come from the output from the encoder and the queries are provided from the previous decoder layer. This way the weighting to predict the next token is done of previously predicted tokens.

Self-Attention The self-attention layer, usually in the encoder, gets all of its inputs of keys, values and queries from the output of the previous layer, additionally each position in the encoder can attend to any position in the antecedent layer of the encoder.

Masked Self-Attention This layer is found in the decoder and is identical to the self-attention layer except that the tokens that have not been decoded are hidden (also called masked),

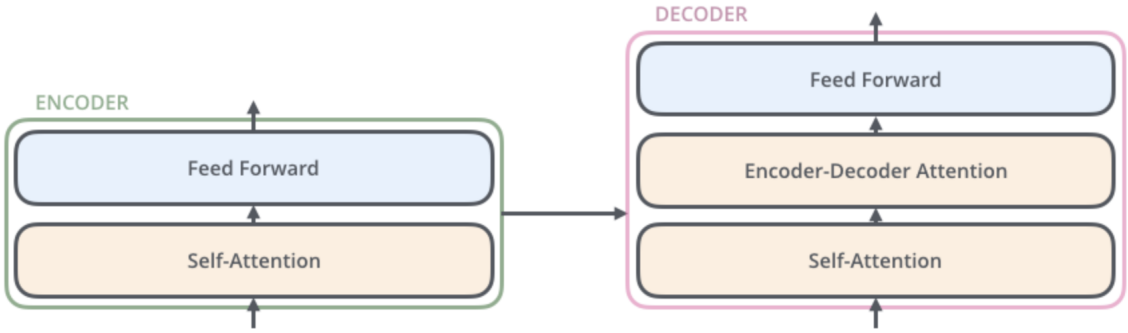


Figure 3.8: A simplified view of the encoder and decoder. After Alammar (2018).

making it only allowed to attend to all previously decoded positions. This is made to ensure that no illegal connections are used.

(Vaswani et al., 2017)

3.6 BERT

BERT, *Bidirectional Encoder Representations from Transformers*, is a language model developed by researchers at Google and presented by Devlin et al. (2019). The *Bidirectional* refers to that BERT unlike traditional language models that reads a sentence left-to-right or right-to-left (or combined) instead attends to the whole sentence. *Encoder Representations from Transformers* refers to the use of encoder blocks in a Transformer structure described in Section 3.5. BERT is pre-trained on a large amount of text ($\approx$ $3,300M$ words) using two methods: Masked Language Modeling (MLM) and Next Sequence Prediction(NSP).

Masked Language Modeling could be described as a fill in the blank test. Words (tokens) in the input sequence are *masked* from the model and the model has to attempt to predict the value of the word embedding of the masked token. It does this by analyzing the surrounding tokens that are not masked, which makes it understand the context of the sentence.

Next Sequence Prediction is used to train the model on sentence classification and how they are relational. The model analyzes two sentences and has to predict if the second sentence is consecutive to the first sentence or not.

BERT produces embeddings for all token inputs including the classification token. A classification embedding is produced from the two methods and is stored in the token called `[cls]`, see Figure 3.6 for clarification, and added in front of the other tokens. The embedding contains BERT's prediction of the sentiment of the sentence and it is especially interesting for this Master's thesis since it is used to determine the label of the message (Devlin et al., 2019).

As an example, we will look at the tokenized message **[A-123456 is your COMPANY verification code.]** used in section 3.4.2 in Figure 3.6. Figure 3.9 displays how the pretrained BERT model takes as input an entire sentence and produces as output embedding vectors of 768 dimension. The `[cls]` tokens' outputted embedding vector is highlighted in blue as its use will be further elaborated later.

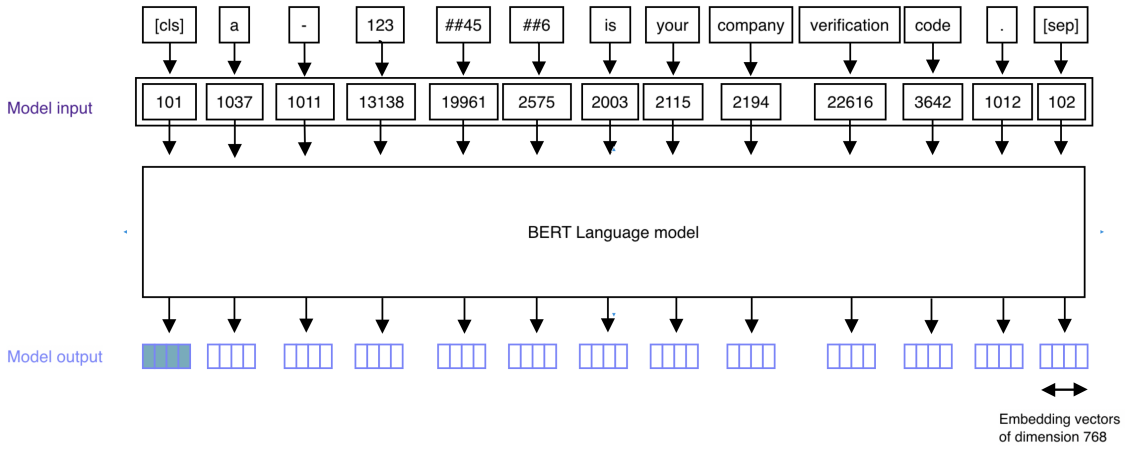


Figure 3.9: A simplified view of the BERT language model.

3.6.1 DistilBERT

DistilBERT is the language model used in the Master's thesis which is developed through the use of the BERT model. The authors use a knowledge distillation technique reducing the time and the parameters which reduces the models size while still retaining a very high accuracy. Knowledge distillation is composed of two participants, a teacher and a student, in this case BERT and DistilBERT respectively. Instead of solely training a model individually, the results from the teacher is available to the student and the student optimizes its own parameters to get as close results to the teacher as possible. (Sanh et al., 2020)

Table 3.1 shows a comparison between results from Sanh et al. paper of DistilBERT and BERT in two recognized NLP tasks of classifying the IMDb and the SQuAD data sets. Additionally there is a comparison of the number of parameters and the inference time of each model. For more information about the tasks, scores and evaluation we recommend you to read the Sanh et al. (2020) paper.

Table 3.1: Comparison of BERT and DistilBERT. After Sanh et al. (2020)

Model	**IMDb** (acc. %)	**SQuAD** (F1 %)	**Nbr of params.** (Millions)	**Inference time** (seconds)
BERT	93.46	88.5	110	668
DistilBERT	92.82	85.8	66	410

The key take away Sanh et al. (2020) want to present in the paper is that by training a student model effectively with knowledge distillation it is possible to make a language model 60% faster, 40% smaller while retaining 97% of its understanding capabilities compared to the BERT teacher (Sanh et al., 2020).

3.7 K-Means

K-means is a clustering method with purpose of finding clusters of similar data points and then group them into K groups. Unlike a classification problem where the classes are provided, clustering methods aim to provide the classes. The K-means method is comprised of the following steps :

1. Select an appropriate value of K, which is the amount of clusters.
2. Select K centroids in the space that are set to the center of each cluster.
3. Calculate the distance from every data point, $\mathbf{x}$, to the centroids in your space.
4. Each data point is assigned to the closest centroid.
5. Find the new centroid of the cluster by computing the mean of all data points of that cluster.
6. Iterate through steps 2, 3 and 4 until convergence and the centroids stop moving.

(Yildirim, 2020)

3.7.1 Elbow Method

To decide the best amount of clusters K on the data set, it is possible to parameterize K and then let the Elbow Method determine the amount. Introducing Within Cluster Sum of Squares (WCSS), in Eq. 3.10, as an evaluation metric that helps determine a point where the value of introducing more classes does not greatly decrease the WCSS score.

$$WCSS = \sum_{C_{j=1}}^{C_K} (\sum_{\mathbf{x}_i in C_j}^{\mathbf{x}_m} distance(\mathbf{x}_i, C_j)^2) \tag{3.10}$$

where C denotes the centroid, $\mathbf{x}$ the data point in each cluster, K the amount of clusters and m the individual amount of data points in each cluster j.

Plotting the WCSS versus the amount of clusters K in a graph we will get a non linear decreasing curve, hopefully including an “elbow” - a point in the curve with a significant angle. The “elbow” signifies that introducing more clusters does not greatly decrease the WCSS, suggesting the optimum K clusters has been reached (Syakur et al., 2018).

3.8 Active Learning

Active Learning is a sub-field of machine learning, where the learning algorithm can choose which data it wants to be trained on. In short, the active learning model is trained with a set of labeled instances after which it can decide whether or not additional unlabeled instances are informative or not, and thus, whether it should be trained on them or not. By utilizing active learning the need for manual labeling of data can be reduced, which as stated earlier is a time consuming and expensive part of machine learning. Moreover, the ranking of unlabeled

instances in terms of their informativeness can sift out the unnecessary training samples and may thus create an even better performing model with less data (Settles, 2009).

A conventional approach to selecting which data to label would be to randomly sample a subset of the unlabeled data set. However, by not making an informed decision on which data to be labeled, it is probable that some of the labeled data will be similar and thus the learning algorithm will just reinforce already learned information and not get the most out of the selected labeled data.

In active learning, the algorithm can instead choose to *query* the *oracle* on which specific data it wants labeled, and thereby gain more information per labeled instance. The main objective of active learning is thus to effectively rank the set of additional unlabeled examples in terms of the added information they would carry if labeled by the oracle and added to the training set of the learner (Esuli and Sebastiani, 2009). In active learning the computer or program is referred to as the *learner* while the *oracle* or *teacher* is the human (usually) who supplies the *learner* with its requested information.

In general, the process of active learning has the following structure:

1. The algorithm is trained on a labeled data set ($\boldsymbol{L}$) which is a subset of the complete data set.

2. The algorithm tries to predict results from an unlabeled data set ($\boldsymbol{u}$).

3. When predicting the labels of the unlabeled data set ($\boldsymbol{u}$), the algorithm ranks the predicted data set in terms of the added information they would provide if labeled.

4. The most valuable samples are manually labeled by the *oracle*.

5. The manually labeled samples are then put in the labeled data set ($\boldsymbol{L}$) and the algorithm gets trained once again.

This process is reiterated as many times as necessary in order to achieve a satisfactory level of accuracy.

To create an active learning program, two main decisions of the learner needs to be made:

1. In which *scenarios* should the learner be able to ask queries.

2. By which criteria should the algorithm determine the informativeness of unlabeled instances. This is called the *Query Strategy Framework*.

3.8.1 Scenarios

When determining which scenarios the learner should query there are three different methods: *member query synthesis*, *stream-based selective sampling*, and *pool-based selective sampling*.

In member query synthesis the learner can request any instance in the input space. The instance is usually synthetically produced *de novo* by the program. In other words, the program creates a synthetic instance that it believes it would gain a lot of information from having labeled. The problem with member query synthesis is that the queries can sometimes be difficult for a human annotator to annotate. In the case of text classification, the program can generate a document that carries no real meaning for a human (Settles, 2009).

In stream-based selective sampling, an instance is drawn from the actual distribution of unlabeled data, the algorithm then decides whether to ask for the instance to be labeled or not. The method is called *stream-based* because each instance is picked one at a time from the data source.

The final method is called *pool-based selective sampling*. This method is similar to stream-based selective sampling in that the query instances are drawn from the actual distribution rather than synthetically generated. The difference between the two methods is that in pool-based selective sampling the learner goes through the whole selection (or a subset) of unlabeled instances before determining which instances to query, rather than doing so sequentially (Settles, 2009).

3.8.2 Query Strategy Framework

The query strategy framework determines the criteria by which the learner decides which unlabeled instances it finds the most informative. Although Settles (2009) highlights several different methods, this study will limit the scope to variations of one framework, namely *uncertainty sampling*. As the name suggests, uncertainty sampling is based on the premise that the most informative samples are those the learner is the least certain about. In the case of binary classification problems, the learner would query the instances that are the closest to 0.5 (where 0 denotes *not member* and 1 denotes *member* of the class). This uncertainty sampling strategy is called *least confident* and in the case of more than one class, the method can be mathematically generalized as follows:

$$\mathbf{x}^*_{LC} = argmax \mathbf{1} - P_\theta(\hat{y}|x) \tag{3.11}$$

where $\hat{y}$ denotes the class that $\mathbf{x}$ most likely belongs to according to model θ. The equation can be interpreted as the expected loss or cost if the model was to mislabel the instance. Thus, the higher the loss, the more likely the model is to query the oracle for the true label of the instance.

However, the least confident strategy only considers the most probable class label of the instance. Another way of determining the uncertainty of an instance would be to also consider the next most likely class label of $\mathbf{x}$. Such a method can be expressed as follows:

$$\mathbf{x}^*_M = argmin P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x) \tag{3.12}$$

here $\hat{y}_1$ denotes the most probable class label and $\hat{y}_2$ denotes the second most probable class label, according to model θ. This method is called *margin sampling*. The method focuses on the models ability to differentiate between the two most likely classes. If it deems the two classes almost as likely being the label of $\mathbf{x}$, the model will query the oracle on the instance. The larger the margin between the two classes, the easier it is for the model to differentiate between the two class labels.

Margin sampling only takes the first two class labels into account and does not consider the probability of the rest. This may be sufficient for classification problems with smaller label sets, but for situations where the label sets are larger, a fully generalized method might be necessary. One such method is called *entropy*. In the case of entropy, the model considers the probability of $\mathbf{x}$ being a member of each of the different classes. If the model is uncertain about $\mathbf{x}$'s membership to several different classes, the entropy will increase and the learner

will query the oracle on the instance. The mathematical expression for entropy based uncertainty sampling is shown below:

$$\mathbf{x}_H^* = argmax \sum_i P_\theta(\hat{y}_i|\mathbf{x}) log P_\theta(\hat{y}_i|x) \tag{3.13}$$

3.8.3 Multi-Label Active Learning

Although the general theory of active learning is applicable on multi-label classification problems as well, most research has focused on single-label active learning (Esuli and Sebastiani, 2009). *Active Learning Strategies for Multi-Label Text Classification* by Esuli and Sebastiani (2009) presented one of the first published studies in the field. In their paper, the authors state that determining the confidence value for a single class label cannot be translated to a multi-label environment in a straightforward manner since multi-label classification generates multiple different confidence values per document. The problem thus becomes one of deciding how to value the total number of confidence values for each document.

One way of handling the multitude of class confidence values would be to separately label each *class label* as opposed to the set of class labels for a whole document. This method is called *local labeling*. By utilizing *local labeling* each class label is handled separately and the learner could ask the oracle if document $\mathbf{x}_i$, is a member of class y_j or not, but does not need to query the whole set of labels for a document.

The other method is called *global labeling* and requires the learner to query the full set of labels for a document. In global labeling the documents are labeled on the confidence of each document, rather than each class label. In contrast to local labeling, the learner in global labeling thus asks for the whole set of $\boldsymbol{Y}_i$ labels for document $\mathbf{x}_i$ and not just a specific class label y_j. In Esuli and Sebastiani (2009), the authors decided on using global labeling since it was deemed to be more time efficient in practice than local labeling.

After deciding to use global labeling, the authors created a system of three different dimensions to vary in order to rank the *informativeness* or *uncertainty* of a document. The three dimensions are: *evidence*, *class*, and *weight* and are briefly described below.

Evidence

The Evidence dimension concerns which type of evidence should be used as basis for the ranking. The first one is *MinConfidence* (C) which is based purely on uncertainty sampling where the idea is that the more uncertain the learner is about the label, the more valuable it is. In this evidence strategy, negative and positive uncertainty is valued equally. That is, the label uncertainties that are farthest from being classified as either a positive or a negative label are valued the highest. Expressed in mathematical terms: $|f(\mathbf{x}_i, y_j) - 0.5| = 0$ would theoretically be the most informative.

An alternative evidence dimension is *Positive MaxScore* (P) where the highest confidence is valued as most important. The reasoning behind this is that in most multi-label classification problems positive samples are scarce, while negative samples are common. Therefore, positive labels should in general be more informative for the learner than negative samples. Thus, if the uncertainty value of the learner is close to 1 it should query the oracle for the label. In mathematical terms: $f(\mathbf{x}_i, y_i) = 1$ would theoretically be the most informative.

Class

The class dimensions is about, given the evidence selected, deciding how to determine a single uncertainty value for each document. Esuli and Sebastiani (2009) brings up three dimensions, but we will focus on two, namely: *Average* (A) and *Max* (M).

The class dimension is intuitively straightforward. When *Average* is used, the document is assigned the average value of the confidence scores of each class label. That is, for document $\mathbf{x}_i$, the classifier values are $\hat{Y}_i = y_1, y_2, ..., y_q$ and the confidence score for that document is $c = (y_1 + y_2 + ... + y_q)/q$. Whereas, in the case of *Max* only the highest value of $\hat{Y}_i = y_1, y_2, ..., y_q$ is counted. That is $c = y_j$ where y_j is the most informative class label according to the evidence dimension.

Weight

Weighting is the third dimension and the choice here is between *Weighting* (W) or *NoWeighting* (N). *NoWeighting* implies that all labels should be treated equally while weighting will promote classes that score poorly on their individual F1-score, the F-score is described in section 3.9.

The learner's certainty score of a samples is in this case the product of the probability of membership to a class label multiplied by the F1-macro score of that individual class label. This skews the priority of the learner towards the poorer performing classes. If a class is scoring zero on its F1-score no distinction will be made between how probabilities are scoring, therefore a buffer constant is inserted to protect from this.

3.9 Evaluation

To evaluate results and compare methods we mainly use F_β-score and Exact Match Ratio (EMR).

3.9.1 F-score

F-score or F_β-score is a metric to evaluate the performance of a model and has several benefits compared to just calculating the accuracy of the model. The accuracy does not differentiate if the error was a type 1 error, predicting a false sample as true, or a type 2 error, predicting a true sample as false. In cases where the dataset is unbalanced for example only containing 10% positive labels, a classifier that only predicts every sample as negative will get an 90 % accuracy.

F_β-score is relative to the *precision* and the *recall*, shown in Equation 3.16, which respectively considers the two types of errors in their equations. Precision and recall are expressed and defined in terms of the following abbreviations:

- TP (true positives): all samples correctly predicted as positive.
- FP (false positives): all samples wrongly predicted as positive.
- TN (true negatives): all samples correctly predicted as negative.
- FN (false negatives):all samples wrongly predicted as negative.

Precision is a fraction of the amount of correctly predicted positive samples compared to all predicted positive samples, shown in Equation 3.14.

$$Precision = \frac{TP}{TP + FP} \tag{3.14}$$

Recall is a fraction of the amount of correctly predicted positive samples compared to all samples that are positive, shown in Equation 3.15.

$$Recall = \frac{TP}{TP + FN} \tag{3.15}$$

By combining precision and recall we get the F_β score as shown in Equation 3.16.

$$F_\beta(TP, FP, FN) = \frac{(\beta^2 + 1) \times Precision \times Recall}{(\beta^2 \times Precision) + Recall} \tag{3.16}$$

The β is a number set by the user to adjust the balance of the metric if either precision or recall is more important for the model. In our case, we use $\beta = 1$ which inserted in Equation 3.16 gives (Gerniers and Saerens, 2018):

$$F_1(TP, FP, FN) = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{3.17}$$

Macro and Micro F-score

When computing the F-score for multi-label classification, we need to address the problem that there is a difference between calculating the F-score before or after summing the different labels' precision and recall. We do this by introducing F^{macro} and F^{micro}.

Macro , by summing each class F-score and dividing by the number of classes the macro F-score gives us an average F-score per class as seen in equation 3.18

$$F_1^{macro} = \frac{1}{q} \times \sum_{i=1}^{q} F1(TP_i, FP_i, FN_i) \tag{3.18}$$

Where q denotes class.

Micro however computes a "global" F-score by first counting all TP, FP and FN from all classes. From equation 3.17 we get:

$$F_1^{micro} = F_1(\sum_{i=1}^{q} TP_i, \sum_{i=1}^{q} FP_i, \sum_{i=1}^{q} FN_i) \tag{3.19}$$

3.9.2 Exact Match Ratio

The second scoring metric we use is Exact Match Ratio (EMR) addresses the problem that multi-label classification evaluation has since predicted samples may be partially correct or

incorrect. Exact Match Ratio only considers a sample prediction to be correct if every class label ($Y_i = \{y_1, y_2, ..., y_q\}$) of the samples has been correctly predicted by the classifier.

$$EMR = \frac{1}{p} \sum_{i=1}^{p} e(x_i) \tag{3.20}$$

Where $e(x_i)$ is defined by Eq. 3.21.

$$e(x_i) = \begin{cases} 0 & h(x_i) \notin Y_i \\ 1 & h(x_i) \in Y_i \end{cases} \tag{3.21}$$

Where $h(\cdot)$ is the prediction of the sample and p is the amount of samples in the test set. Sorower (2010) debates that this evaluation might be to harsh in punishing the model for slight mistakes, especially if there are many labels.

Threshold

In the results chapter we refer to two types of exact match ratio, i.e. EMR1 and EMR2. They are calculated the same way using Eq. 3.20. The difference is how we have predicted the labels it is to evaluate. In EMR1, we let the classifiers predict labels with their default threshold and insert that in Eq. 3.20. In EMR2 however, we let the classifier return a probability for each label, a threshold, t, is parametrized from 0.1 to 0.9 classifying a label as one if its probability is above the threshold t and zero otherwise. We then search for the threshold t that returns the highest EMR score and also display the threshold that gave that score.

Chapter 4

Approach

In this chapter, we present our approach and methodology of the thesis work and how it was carried out. We also explain the experiments that were executed and what results were hoped to be produced. The results are presented in the next chapter.

4.1 Labeling

Initially we wanted to see if the labeling process could be expedited and simplified by mathematically clustering the messages before picking the class labels and starting the annotation. To do this, we sampled a small subset out of the original S1 data set to work with. After using DistilBERT to embed the messages, we performed the Elbow method with K-means, to see if there was an optimal number of clusters that the method suggested we should use as classes. The K was parameterized from 1 to 20 classes and then plotted in a graph versus the $WCSS$ as outcome. Unfortunately, the results from the Elbow method alone were not satisfactory and after discussion with experts at Sinch, we decided to settle for a more or less subjectively chosen number of classes.

We then performed K-means on a small subset of the data set, clustering the messages into the number of classes decided. The goal of this was to make the algorithm sort the messages into rough groups which would ease the work of labeling. By reading messages in the clusters created by the K-means algorithm, we figured out a label name for that class; the label names were deemed appropriate by Sinch.

We manually labeled 10K messages into the multi-label classes which resulted in the data set called S1. Later on we further labeled 4K messages, sampled from another time period, using the same class labels as for S1, which resulted in the S2 data set.

4.2 Model choice

The selection of a classification model was made with criteria from both LTH and Sinch in mind. Both parties wanted robustness and accuracy while Sinch was more interested in the speed of training and classification. We created and tested five different models, as mentioned in the theory section: LogReg, SVC, LinearSVC, Random Forest, and XGBoost. The model choice was performed using the S1 set, and before training the models, the messages were embedded using DistilBERT. DistilBERT takes each each message and transforms it into a 768-dimensional vector. Thus, each model takes an input vector of 768 dimensions and outputs a vector of 13 dimensions - corresponding to our labels.

To mitigate the risk of making a biased choice of model based on incidental factors such as favorable train test splits or special characteristics of the S1 data set, two measures were applied: k-folding and parallel testing:

K-folding refers to the process of reiterating the model K times, shifting the train and test split so that all data instances gets used as both train and test sets. After K iterations, the average result is calculated and evaluated on the performance according to Chollet (2018).

Parallel testing was carried out with the RCV1 which was deemed similar to the Sinch data set in that it is also a multi-label text classification set. It has the advantage that it is widely known and tested and contains a lot of labeled data.

The step wise process of the classification is listed below:

- Preprocessing of the data sets, e.g. cleaning up the data from special symbols.
- Feeding messages to DistilBERT which produces a sentiment vector of 768 dimensions.
- Splitting the data into train and test size in an 80% / 20% proportion respectively.
- Training one out of the five models on the dataset.
- Gathering evaluation metrics from the results of the model being tested and inserting in table.

For clarification: the classifier model only takes the classification tokens vector not the vectors from the remaining word tokens.

The classifier classifies the message based on the information in the classification token `[cls]`, translated to a vector $\mathbf{x}_i$, and outputs a vector of length 13, referred to as Label Probability Vector, (LPV). Each dimension in the LPV is representing the probability of the sample corresponding to a label. In other words, for each $\mathbf{x}_i \in \mathcal{D}$, the LPV is calculated by combining the output of 13 binary classifier functions $f(\mathbf{x}_i, y_j)$ - one for each $y_j \in \mathcal{Y}$ - that calculate the probability of each sample being a member of y_j.

The probability ranges from 0 to 1, where values in the LPV close to 1 equates that the model is certain that the message is *a member* of class y_j. Values close to 0 indicate that the model is certain that the sample is *not a member* of the class y_j. Finally, a value around 0.5 means that the model is uncertain if the sample is or is not a member of y_j.

After being put through a threshold function $t(LPV)$, the model then produces its prediction vector $\hat{\mathbf{y}}_i = (y_1, y_2, ..., y_q)$, in which each position is given a value of (1,0). The evaluation of the loss of the model is then based on the difference between the prediction vector $\hat{\mathbf{y}}_i$ and the proper label vector $\mathbf{y}_i$ of each feature vector $\mathbf{x}_i$.

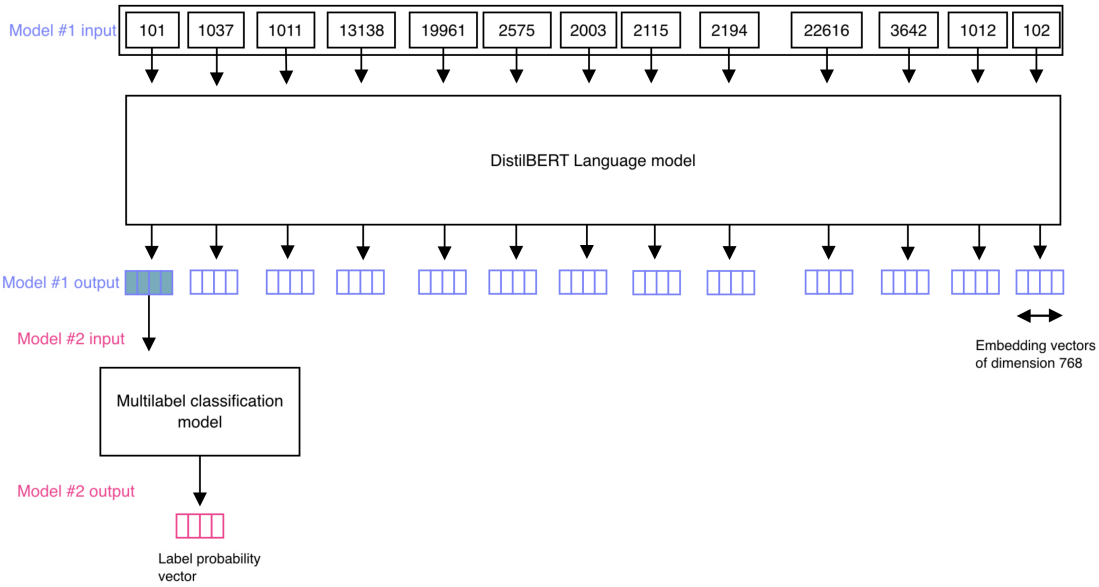


Figure 4.1: A simplified view of the architecture behind classifying a sample.

The evaluation metrics were collected and calculated and the choice of model are presented in the Results chapter, section 5.2.

Before continuing to perform the Active Learning on the S2 data set, the chosen model had its hyper-parameters optimized by iterating through the S1 data set on different combinations of settings.

4.3 Active Learning

To evaluate different Active Learning strategies, we created a flexible testing framework, where we could easily change parameters to gather results from them and compare them in graphs. Moreover, the strategy variations were inspired by the work of Esuli and Sebastiani (2009). We also let one model query random samples to be fitted on, working as a baseline reference to which the other combinations were compared.

For the Scenarios, we chose *pool based selective sampling*, because that is how Sinch currently evaluate their data. For the Query Strategy Framework, we chose to focus on *uncertainty sampling*, adapted to multi-label classification. The different active learning strategies were composed of variations of the three dimensions of multi-label active learning mentioned in the theory Section 3.8.3: Evidence, Class, and Weight. These dimensions have two variations each resulting in eight possible combinations (2^3).

The variations for each dimension are: *MinConfidence* (C), *Positive MaxScore* (P) for Evidence; *Average* (A) and *Max* (M) for Class; and *Weighting* (W) and *NoWeighting* (N) for Weight.

Together with the Random query selection, the complete number of combinations are: CMN, CAN, PMN, PAN, CMW, CAW, PMW, PAW, and RAND.

To test the effect of batching on the speed and accuracy, we also tested five different batch sizes. These batch sizes denote the number of queries selected from the pool before each training iteration. Note that this is not to be confused with the batch size of the machine learning algorithms. The query batch sizes that each combination was tested on were: 1, 5, 10, 25, and 50. The experiment process is illustrated in Figure 4.2 and described in the next paragraph.

The classifier was initially trained on the S1 data set of 10,000 samples, then the model starts the active learning process on the new S2 data set of 4,000 samples. First, a test subset of 800 samples is taken from the S2 data set, leaving the pool for query selection at 3,200 instances. For each iteration of the experiments, the test split is then circulated using the k-fold iteration. The classifier makes a prediction of the unlabeled samples in the pool and thus creates a Label Probability Vector for each message. Next, the active learning algorithm ranks the Label Probability Vectors of the samples in the selection pool, according to the selected strategy combination. The model then queries as many of the highest ranked samples as the batch size specifies, the samples "get annotated" by the oracle, and are then used to retrain the model. This process is repeated until there are no more samples in the pool. We use a software library called ModAL, from Danka and Horvath (2018), which implements the active learning function *teach*, this function presents the newly acquired samples and labels and refits the model on them.

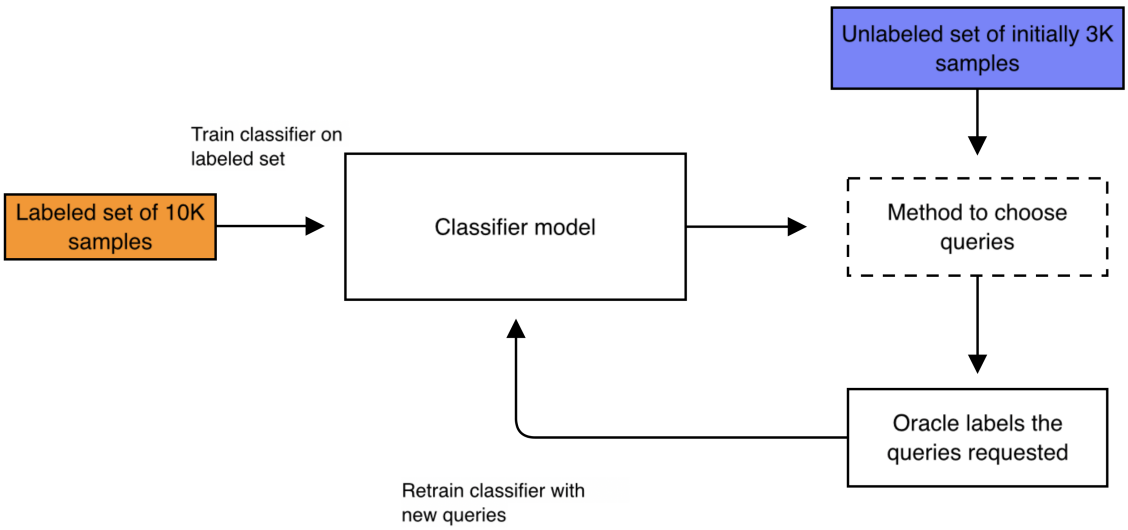


Figure 4.2: A simplified view of the architecture of how the iteration of active learning is done. The dotted lines surrounding the method to choose queries symbolizes that the method is exchanged depending on the active learning strategy combination.

4.4 Manual inspection

Both during the training of the model on the data set of 10K samples and later the active learning model of 4K samples, we gathered incorrectly labeled samples and put them into a spread sheet for manual inspection. The mislabeled samples could then be used to get a more detailed understanding of what the model was doing wrong, and which areas were of most concern.

For the active learner models, we also saved all samples that were queried and in which order. This let us get a view of what the active learners were querying about and helped us understand what might be improved.

Chapter 5

Results

In this chapter, we will present the results gained from the experiments introduced in the approach chapter by displaying them in plots and tables. The results will be briefly analysed and commented while a more comprehensive discussion will follow in the next chapter. Initially we will present the results of our labeling process, followed by the choice of the model, the active learning, and finally the manual inspection of the queries posed by the learner.

5.1 Labeling

In Figure 5.1, the result from the Elbow method is displayed. As seen, there is no clear "elbow" and therefore it was concluded that it was not enough to base the choice of the number of classes on this experiment alone.

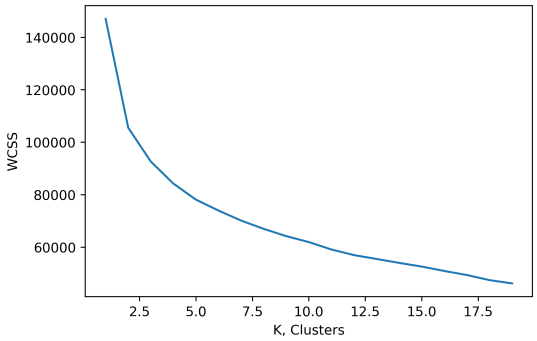


Figure 5.1: Result of cluster exploration with the Elbow method.

With discussion with experts at Sinch, 13 labels were decided upon to label the data set. The labels have already been presented and can be viewed in Table 2.1.

5.2 Choice of model

The experiments for testing the different models on S1 were carried out as described in section 4.2 and the results can be found in Table 5.1 below.

Time 1 and Time 2 denote the training and prediction time, respectively, in seconds. Furthermore EMR1, EMR2, F1 Macro and F1 Micro are collectively referred to as the performance metrics, while Time 1 and Time 2 are grouped together as the time categories. The boldfaced numbers represent the best results in each category.

Table 5.1: Performance of the different models on the Sinch data set 1 of 10K samples (with train/test split set to 80/20). The performance measurements are: EMR1 (Exact Match Ratio default threshold), EMR2 (Exact Match Ratio with threshold as t), F1 Macro, F1 Micro, Time 1 (training time), and Time 2 (testing time), both in seconds.

Method	EMR1	EMR2	F1 Macro	F1 Micro	Time 1	Time 2
LogReg	0.926	0.926 ($t = 0.5$)	0.928	0.973	**3.9**	0.05
SVC	0.907	0.918 ($t = 0.45$)	0.908	0.969	303.9	11.3
LinearSVC	**0.948**	-	**0.945**	**0.980**	27.6	**0.021**
Random Forest	0.928	0.932 ($t = 0.35$)	0.913	0.976	35.9	0.4
XGBoost	0.939	**0.939** ($t = 0.5$)	0.937	0.977	389	0.28

The results suggest that LinearSVC is the best method in terms of most categories with best results. Surprisingly, LinearSVC even has better results than regular SVC. The reason why LinearSVC does not have any EMR2 results is because there is no straightforward way of getting the probability scores of each label, thus there is no easy way of regulating the threshold.

The results acquired from the parallel testing of the RCV1 can be found in Table 5.2 below. The table shows the results from the different models after training it on 12,000 and testing it on 3,000

Table 5.2: Performance of the different models on the Reuters data set limited to 15K samples (with train/test split set to 80/20). The performance measurements are: EMR1 (Exact Match Ratio default threshold), EMR2 (Exact Match Ratio with threshold as t), F1 Macro, F1 Micro, Time 1 (training time), and Time 2 (prediction time). The unit of Time 1 and Time 2 are seconds.

Method	EMR1	EMR2	F1 Macro	F1 Micro	Time 1	Time 2
LogReg	0.370	0.370 ($t = 0.5$)	0.374	0.736	**3.3**	**0.13**
SVC	0.354	**0.459** ($t = 0.5$)	0.523	**0.784**	4241.7	145.1
LinearSVC	**0.442**	-	**0.541**	0.782	228.3	0.69
Random Forest	0.299	0.380 ($t = 0.35$)	0.296	0.725	307.9	2.9
XGBoost	0.434	0.447 ($t = 0.26$)	0.441	0.778	4837	6.6

In Table 5.2, once again LinearSVC is performing best in EMR1 and F1 Macro while only lacking 0.2 percent units in F1 Micro compared to the best in the category, SVC. LogReg is

scoring worse in the four performance metrics and especially in F1 Macro, though it performs best in terms of time efficiency. SVC performs best on overall performance metrics. We can see that the time for training has increased greatly of all classifiers in Table 5.2 compared to 5.1.

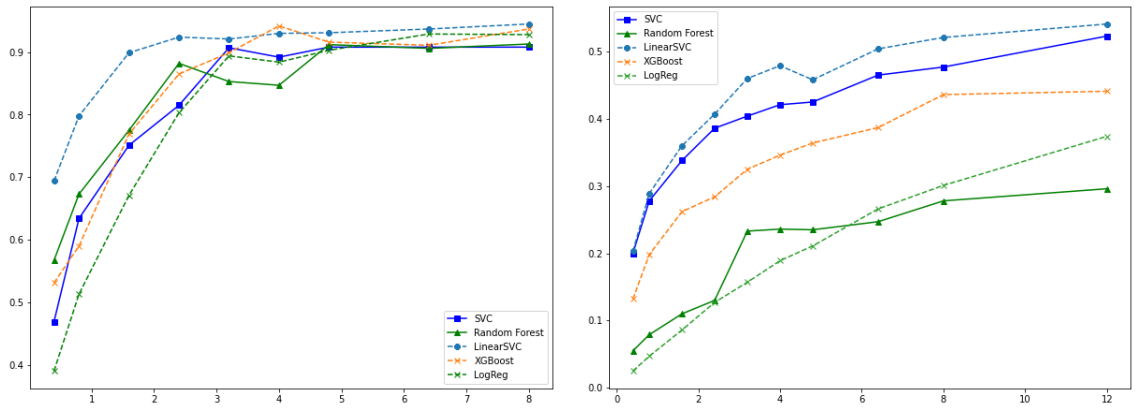
For comparison, we let the LogReg be evaluated on the full set of 800K samples of RCV1 set - with a train/test split of 80/20 - and compared our results to Lewis et al.'s 2004 benchmark results of SVC.1 and k-NN, in Table 5.3. Lewis et al. refer to their Support Vector Classifier as SVC.1 since the kernel function that maps the points into higher dimensions is of the first degree, which is equal to using a LinearSVC.

Table 5.3: Comparison of our LogReg classifier to the bench marked SVC.1 and k-NN classifiers on the Reuters data set. After Lewis et al. (2004)

Method	F1 Macro	F1 Micro
LogReg	0.647	0.822
SVC.1	0.607	0.816
k-NN	0.549	0.765

In Table 5.3, we can see that our classifier is performing on the same level as the bench marked results.

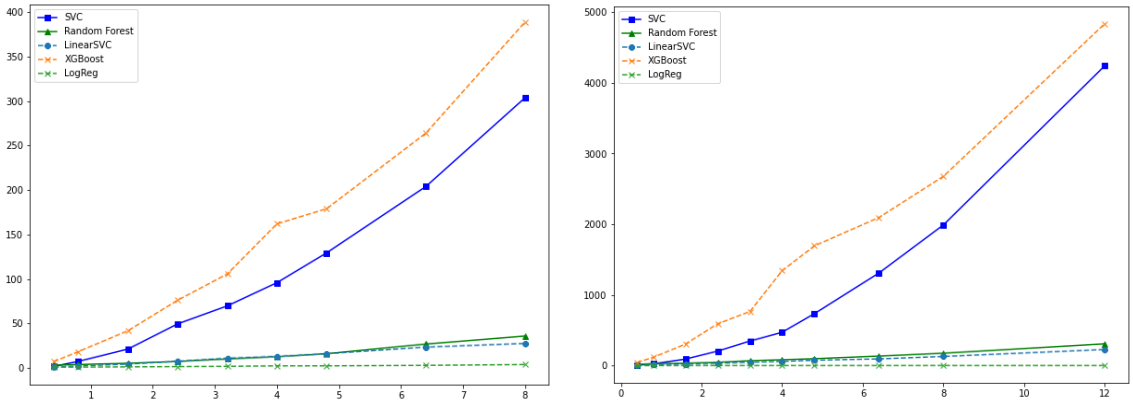
In Figures 5.2 and 5.3 below, the learning rates and training times for the two data sets S1 and RCV1 are shown. In Figure 5.2 (a), we see that the Sinch S1 models all have converged to a stable number after approximately 6,000 samples. Moreover, it can be seen that none of the models have converged after 12,000 training samples sub-figure (b). Out of the 5 models, LogReg seems to have the steepest curve, indicating that the performance of the model could be increased the most by training on more samples.



(a) Sinch Set 1 **(b)** Reuters.

Figure 5.2: Learning curves of Sinch data set 1 and Reuters data set for different models. The *y*-axis denotes the F1 macro score and the *x*-axis denotes the number of training samples in thousands. Note that the axes are not fix.

The training times of the models on Sinch S1 and Reuters RCV1 are shown in Figure 5.3. In both sub-figures we see that XGBoost and SVC are exponentially increasing when



(a) Sinch Set 1 **(b)** Reuters

Figure 5.3: Training times of Sinch set 1 and Reuters data set for different models. The *y*-axis denotes the training times in seconds and the *x*-axis denotes the number of training samples in thousands.

introduced to more training samples. Of the three remaining methods in Figures 5.3, LogReg is the quickest one when introduced to more training samples.

Finally, we decided to move on with LogReg as the model for testing active learning on. The reason was that it was the fastest model with satisfactory performance results. The decision will be explained in further detail in the discussion chapter.

5.3 Active Learning

All eight different active learning strategies have their F1 Macro and EMR score plotted over samples queried shown in Figures A.1 and A.2, respectively, placed in Appendix A for convenience. The total of 16 graphs show the various methods' individual scores from the testing of the five different batch sizes.

We can see in both Figures A.1 and A.2 that using a batch size of one gave the best performance in every graph except one of them (PMN). Therefore, we used batch size one as the main comparison between the eight methods and the random query learner was used as reference, when comparing their F1 Macro and EMR score in Figures 5.4 and 5.5, respectively.

It is difficult to draw decisive conclusions from both plots in Figures 5.4 and 5.5, since the plotlines are very volatile. The best performing strategy at one point will have recessed compared to other strategies for larger training sets. A few points can be made about the graphs though, disregarding the Positive MaxScore plot-lines. They all seem to moderately converge after around 1500 samples – the *MinConfidence* (C) plot-lines converge even earlier. Random (RAND) also outputs the same values at convergence.

Weighting (W) seems to significantly improve strategies using *Positive MaxScore* (P).

The first point in each plot-line is at 50 samples queried and here Random (RAND) is outperforming all other strategies in both F1 and EMR score.

As said it is hard to summarize the graphs given the volatility, with our supervisor we decided that 500, 1000 and 1500 are three interesting sample sizes to look into. We summa-

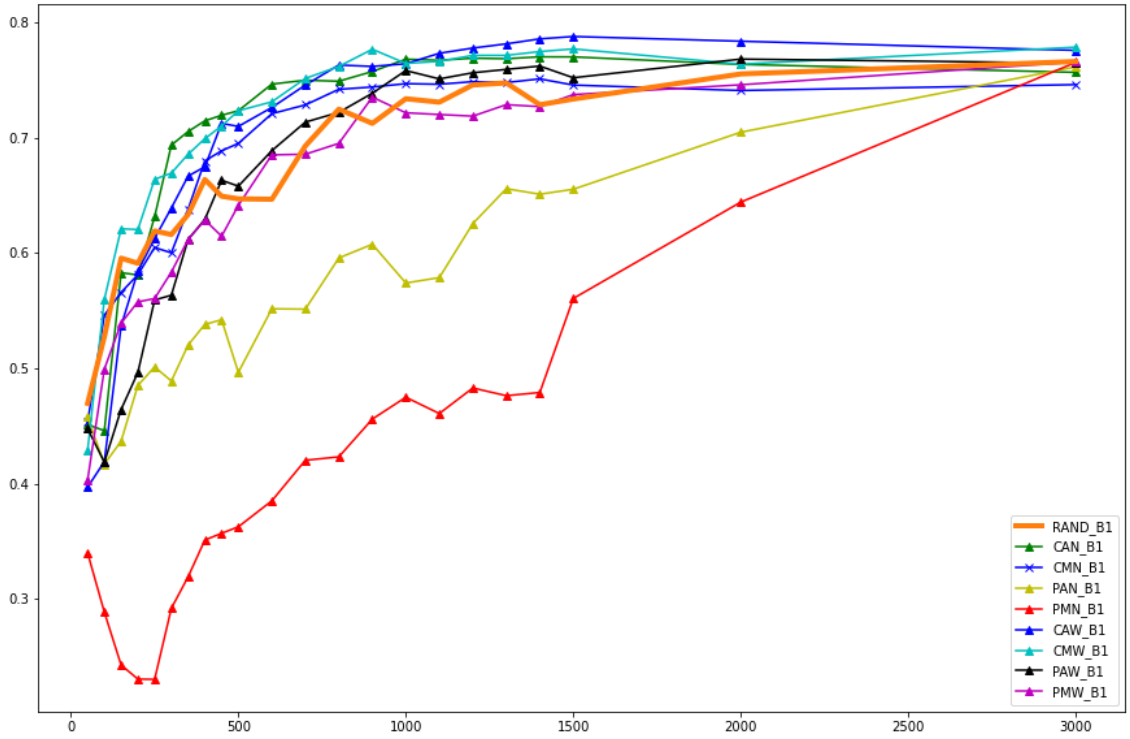


Figure 5.4: Macro F1 score for batch size one.

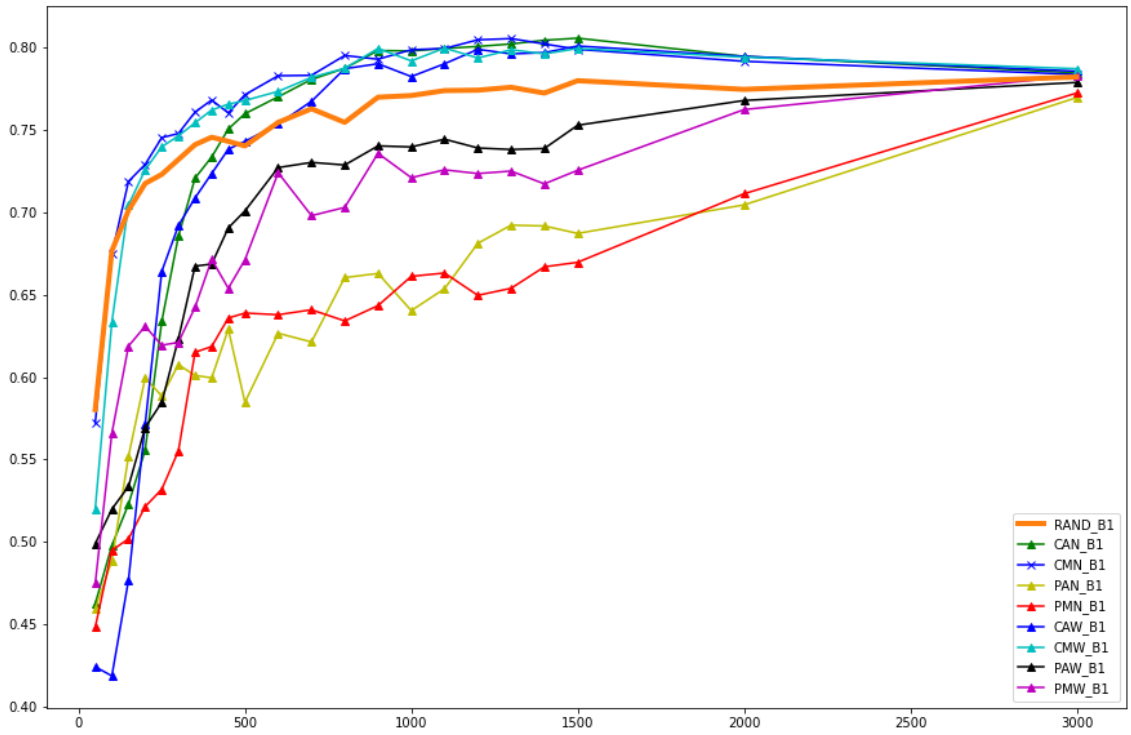


Figure 5.5: Exact Match Ratio score for batch size one.

rized them into table 5.4 showing F1 macro, EMR and area under F1 respectively EMR in a range from 500-1000 samples. The area metric can be used to complement the F1 and EMR

score since it reduces the variation of the metrics by averaging the scores over an interval. The boldfaced numbers represent the best results in each category.

Table 5.4: Table of results for the eight different methods using active learning compared to randomly training the model(rand). Boldfaced numbers represent the best result in each category

Metric (interval)	Rand	CMN	CAN	PMN	PAN	CMW	CAW	PMW	PAW
F1 (500)	0.647	0.695	**0.724**	0.363	0.497	**0.724**	0.710	0.641	0.658
F1 (1000)	0.734	0.747	**0.768**	0.475	0.574	0.765	0.764	0.722	0.758
F1 (1500)	0.734	0.746	0.770	0.561	0.655	0.777	**0.788**	0.738	0.752
EMR (500)	0.740	**0.771**	0.760	0.638	0.585	0.768	0.743	0.671	0.700
EMR (1000)	0.771	**0.798**	**0.798**	0.661	0.640	0.792	0.782	0.721	0.739
EMR (1500)	0.780	0.799	**0.810**	0.670	0.687	0.799	0.801	0.725	0.753
F1 Area (500-1000)	346.6	365.9	375.1	210.4	285.0	**376.8**	373.6	348.8	357.3
EMR Area (500-1000)	379.7	**393.8**	391.4	320.4	318.9	392.1	386.1	356.1	364.8

Table 5.5: Table of results for the eight different methods using active learning compared to randomly training the model(rand). Boldfaced numbers represent the best result in each category

Metric (interval)	Rand	CMN	CAN	CMW	CAW	PAW
F1 (500)	0.647	0.695	**0.724**	**0.724**	0.710	0.658
F1 (1000)	0.734	0.747	**0.768**	0.765	0.764	0.758
F1 (1500)	0.734	0.746	0.770	0.777	**0.788**	0.752
EMR (500)	0.740	**0.771**	0.760	0.768	0.743	0.700
EMR (1000)	0.771	**0.798**	**0.798**	0.792	0.782	0.739
EMR (1500)	0.780	0.799	**0.810**	0.799	0.801	0.753
F1 Area (500-1000)	346.6	365.9	375.1	**376.8**	373.6	357.3
EMR Area (500-1000)	379.7	**393.8**	391.4	392.1	386.1	364.8

To tie the results to the exploratory data analysis, we also recorded the results for each class. The results are shown in Table 5.6, which specifically show the F1 Macro start, min, max and increase values for the different labels. The classes with the highest increase are *Public Information* (69.7 %), *Job/Edu* (69.5 %), and *Service* (59.4 %), while the lowest increases are *Notification* (1.2 %), *Two Factor Authentication* (6.1 %), and *Alert* (13.3 %).

5.4 Manual inspection

The manual inspection was done by letting the Exact Match Ratio save incorrect predictions on both the S1 and S2 data set. The saved samples were thus composed of all messages that had at least one of their class labels incorrectly predicted. Apart from the imperfect predictions by the classifiers, some of the incorrect predictions depended on one or more of following reasons:

Table 5.6: F1 Macro start value, minimum value, maximum value and increase (= max-start) for the individual classes of CAN. The boldfaced numbers represent the best and the worst in each category.

Data Set	Bank	Trans	2FA	Mark	Fin	Soc	N.Eng
Start	0.777	0.635	0.823	0.722	0.270	0.054	0.268
Min	0.777	0.453	0.817	0.722	**0.0**	0.054	0.208
Max	0.958	**0.963**	0.884	0.905	0.720	**0.442**	0.831
Increase	0.181	0.328	**0.061**	0.183	0.450	0.388	0.562
Data Set	**Job**	**Ser**	**Mon**	**P.I.**	**Al**	**Not**	
Start	0.069	0.033	0.734	**0.0**	0.528	**0.951**	
Min	0.069	0.033	0.647	**0.0**	0.528	**0.951**	
Max	0.765	0.627	0.932	0.697	0.66	**0.963**	
Increase	0.695	0.594	0.198	**0.697**	0.133	0.012	

- Samples were wrongly labeled by the manual labeling. There are two types of errors here, namely:
  - Mistake made by the person labeling.
  - Subjective error. That is, making inconsistent choices of labels on similar samples.
- Labels that are closely related and use similar words, such as banking and finance, get mixed up.
- Specific details, such as a company name in the message, may have made the person labeling choose a label, while the detail is to subtle for the classifier to understand.
- Messages not in English have sometimes been labeled Non-English as well as their intended label, e.g. Non-English, Service and Notification. This happened when the person labeling understood either the message (often Spanish) or recognized the pattern of the message.

Furthermore in our initial experiments of active learning, on querying a single sample, we manually inspected some of the queried samples. We found that in the beginning the learner would query relatively similar samples consecutively.

Chapter 6
Discussion

In this chapter, the results from the experiments will be discussed; that is, both the experiments chosen and the output from them. We will evaluate the conclusions and patterns that can be found in our data as well as the flaws of our experiments. Furthermore, future work that could complement our experiments will be proposed, and our results will be concluded in the last section.

6.1 Labels

When the Elbow method gave no conclusive results we decided to make observations and create labels as we saw fit, rather than relying on clustering algorithms alone. Although the elbow method did not show any conclusive results (see Figure 5.1), using K-means to optimize the clustering of messages after deciding on an approximate number of classes around 10-15 helped us with the labeling of the messages. However, our subjective opinions were necessary as a fully mathematical approach to clustering could easily have ignored less frequently occurring labels such as *Non-English* and *Public Information*.

The choice of exploring the Elbow method was made because we were familiar with it. A more comprehensive survey of similar methods could have produced more accurate results. For example, Rousseeuw (1987) proposes the Silhouette method for similar purposes. This method could have been used as an alternative to the Elbow method or as a validation method for the amount of labels chosen.

In hindsight, the number of labels were not ideal. Especially the addition of *Alert* and *Notification* were possibly a little too vague for our classifiers to distinguish between. What can be found in Table 5.6 is that the F1 score of *Notification* and *Alert* did not increase significantly with active learning. The reason why we initially chose to add *Alert* and *Notification* to the class labels were to separate the urgent and less expected messages from the more expected and unimportant ones; notably, a highly subjective task. As explained in the Data Set chapter, we wanted messages such as e.g. *"your card has been declined"* to be treated differently than e.g.

"this is your receipt from XXX store." However, the subtle differences and subjective nature of the classes showed that even we had a difficult time distinguishing between them in some cases. It is therefore not very surprising that the classifiers had the same issues.

If more time had been allocated to the task of annotation, additional labels could have been introduced. E.g. when annotating the S2 data set, potential classes such as *Retailer*, *Order Status*, *Bar/Restaurant*, and others frequently occurred. However, because of the time constraint, we did not want complicate matters by introducing more labels to the classifier and therefore decided to stick to the original labels of the S1 data set. The labels of S1 were derived from the characteristics of a data set that was allocated at a specific time in a specific region, and may therefore not have been fully representative of other Sinch data sets. The introduction of other classes and other data sets could therefore have improved the labels in two ways: 1. by being more suited for other data sets, and 2. by evening out the imbalance of the label frequencies.

6.2 Model choice

After carrying out the experiments and evaluating the performance results of the five different models, we finally settled for LogReg as the model to go on with. The choice was not completely unambiguous since LogReg was outperformed by other classifiers in several of the measured parameters. However, to understand our decision, the goal of the model choice must be emphasized. That is, the goal was to find a model that performed satisfactory and was time efficient enough to be used to make multiple iterations of active learning training on. For this purpose, we found that LogReg was a good model. That being said, most of the other models could have been used for the same purpose based solely on performance metrics such as EMR and F1 Macro, but the main reason why LogReg was finally chosen was because of its time efficiency. The selection process is discussed in more detail below.

As mentioned, Time 1 is the time it takes for training the model. This is an important factor in choosing the model for active learning since the model will be trained on queried samples in several thousand training iterations for each active learning strategy combination. In the Time 1 column in 5.1 we see that LogReg is the fastest classifier. To further evaluate Time 1 we plotted the data into Figure 5.3 where the difference in training time between the classifiers is even more apparent. We can see that the training times of SVC and XGBoost are exponentially increasing with increased samples. Their recorded training times in 5.1 are 303.9 s and 389 s, respectively, far exceeding those of LogReg (3.9 s), LinearSVC (27.6 s), and Random Forest (35.9 s). The long training time of XGBoost surprising since sources claimed that the algorithm was optimized and scalable (Chen and Guestrin, 2016). The speed of XGBoost that sources express is likely viewed in relation to more complex models, though, and a comparison with an elementary model such as LogReg on a simple task may therefore not be fair.

Time 2 denotes the time to predict the labels of the test set: $h(\mathbf{x}_i)$, for all $\mathbf{x}_i \in \mathcal{S}$. That is, the time to predict the probability function of each label $f(\mathbf{x}_i, y_j)$ for each label $y_j \in \mathcal{Y}$ and then applying the threshold function, t, to acquire the prediction vector $\hat{\mathbf{y}}_i$. Time 2 is of value because it relates to the probability calculation that is later required for ranking the queries in active learning. This prediction, which gives us the aforementioned Label Probability Vector LPV, is the product of calculating $f(\mathbf{x}_i, y_j)$ for each label y_j.

A mistake was made in the recording of Time 2, since we recorded the prediction time of $h(\mathbf{x}_i)$, rather than the probability prediction of f. However, since h is obtained by adding threshold function t to each sample, this threshold function represents nothing more than a threshold probability applied on each LPV produced. It can thus be assumed to be linear in relation to feature test size $\mathbf{x}_i \in \mathcal{S}$ and therefore almost identical for all of our explored classifiers. Thus, the Time 2 recorded can be assumed to be directly correlated with the probability prediction times that we *de facto* sought after.

The results for Time 2, show that LinearSVC is the fastest at 0.021 s, followed by LogReg (0.05 s), XGBoost (0.28), and Random Forest (0.4 s). SVC has a far slower prediction time of 11.3 s.

On performance metrics the classifiers displayed fairly similar results. All had an exact match ratio of above 90 % with default threshold setting as well as optimized threshold. The best performing classifier in EMR was LinearSVC at 94.8%. Even after optimizing the threshold for the other classifiers, the highest EMR 2 score (XGBoost at 93.9%) was still lower than that of LinearSVC. Similarly for F1 Macro and F1 Micro, LinearSVC performed best at 94.5 % and 98.0 %, respectively. What is interesting to note is that LinearSVC performed better than SVC on all performance metrics. The reason for this may be that a non-linear kernel for the SVC overfitted the classification, because of the simplicity of the data set. If more time had been given, this is something we could have investigated further.

6.2.1 Sinch and Reuters comparison

The parallel testing of the models on the RCV1 set was performed in order to tell if the results discovered on the S1 data set were only circumstantial or if the same conclusions could be reached on another data set as well.

In the Reuters data set we originally had 800K samples but we only compared the five models using a randomly selected subset of 15,000 samples. This was because some of the models were slow and exponentially increasing in training time so training on the full RCV1 set would have been too time consuming. Since we were mainly focused comparing the models with one another, we chose to limit our training set to 12,000, and test on 3,000 documents.

By comparing the RCV1 results in Table 5.2 the S1 results in Table 5.1, several inferences can be made. First, we can see that the classifiers' performances relative to one another were fairly similar to those of S1. By looking at Table 5.2 we see that LinearSVC has the best results in EMR1 (44.2 %) and F1 Macro (54.1%), while SVC this time outperformed LinearSVC in both EMR2 (45.9 %) and F1 Micro (78.4 %). This may be because of RCV1 is more difficult to classify because of the length of the documents and variation compared to S1. Due to this, the more advanced nature of SVC compared to LinearSVC might have played a role.

The training and prediction times also show similar results to those of S1. However, on RCV1, LogReg performed far better than the other classifiers with Time 1 and Time 2 being 3.3 s and 0.13 s, respectively. By looking at Figure 5.3 (b) we see that RCV1 plot is almost identical to sub-figure (b).

Second, we see worse results in every category of RCV1 compared to the S1 performance results in Table 5.1. There are several reasons for this. One is that there are 103 labels in the RCV1 data set. With only 15K randomly selected samples there is a chance that there are some labels that barely get trained on. As discussed in the theory chapter, the complexity of multi-

label text classification does not increase linearly but rather exponentially with a growing number of classes. The theoretical amount of combinations of labels to classify one document in RCV1 is $2^{103} \approx 10^{31}$. S1 on the other hand, has only $2^{13} \approx 8000$ combinations. Moreover, the articles of the RCV1 set are much longer and more complicated than the messages of S1. This also complicates the classification. And lastly, most messages in the S1 data set follow some form of template, which is repeated on other messages in the data set and it is therefore easier for the classifier to find patterns in S1 than in RCV1. For these reasons, the classification of RCV1 are more complicated than S1. Regardless, however, the relative performance of the classifiers proved to be mostly consistent on both of the data sets, which was the goal of the comparison.

When looking at the results, the training time for XGBoost and SVC proved to be too long with regards to the little, if any, performance benefits they added. The final evaluation therefore stood between LinearSVC, Random Forest, and LogReg. Random Forest, though, was not a good option since it was outperformed in almost every performance category. LinearSVC on the other hand did outperform LogReg and the other models in most categories. The problem with LinearSVC was retrieving probability from the model on each class. This was an issue we discovered after we had performed the experiments for choosing the models. We also discovered that the lack of a simple way of retrieving probability from LinearSVC was a problem for the software library ModAL used for Active Learning, since its implementation relies on classifiers with probability integrated in their code. A way of getting around this problem was tested but could probably have been more thoroughly explored. In the end we opted to rule out LinearSVC as a nominee to the model choice because of this factor.

With these results in mind, we finally concluded that LogReg was the best option. LogReg did not score the highest of the models tested in any of the four performance metrics, but in Table 5.1 it was only missing around 0.02 marks in each score and in Table 5.2 missing in a range from 5 to 17 marks to the best of the category. Since LinearSVC was ruled out, LogReg was scoring good enough in the performance metrics and best in the time categories, furthermore it is a simple model to understand and use and was hence chosen as our model to evaluate active learning on.

In Table 5.3, we compare the results of LogReg on the entire RCV1 data set with 800,000 samples with the authors (Lewis et al., 2004) models of SVC.1 and k-NN classifiers. We can see that LogReg is performing better in both F1 Macro and Micro. We believe that the LogReg is performing better than the bench marked results due to better language models and larger pre-computations. We used a model based on BERT which is pre-trained on 3,300 million words, which was not yet introduced in 2004. Our supervisor also proposed the reason could be that work in the field has resulted in e.g. better float precision and enhanced optimizers making models more precise.

In the bench mark performed by Lewis et al. (2004), the authors train their models on 23k documents indicating that the models should converge around that number. We could have trained our classifiers on the same number of samples and thereby probably achieved convergence. However, since the RCV1 comparisons were mainly performed to see if the conclusions of the classification of the S1 set were true for another data set as well, the exact number of training samples was not deemed to be the most important factor.

As a final remark to our choice of model, we can conclude that all our models performed extremely well on the S1 data set. This is assumed to be because the data set is imbalanced and labels such as *Two factor authentication* and *Banking* are highly represented in both training

and test sets as seen in the Exploratory Data Analysis in section 2.3.

6.3 Active learning

The results from active learning experiments indicate that a batch size of one with MinConfidence Average and NoWeighting (CAN) is the best combination for our active learner model. Furthermore any combination containing the MinConfidence as evidence dimension seems to outperform random query selection and Positive MaxScore in Table 5.4. In this section we will discuss the experiments and their results.

6.3.1 Batch Sizes

The main reason for testing different batch sizes is that with the high volume of messages that Sinch needs to classify every day, the model cannot be retrained on one sample for every iteration. Thus, a review on the effects of batching queries was sought after. Another reason was that when manually inspecting the query samples by our learner, we found that in our initial experiments with a batch size of one, the learner had queried similar samples several times in a row. This created a concern that the learning rate of teaching one sample at a time was not sufficient for training the classifier. We were hoping that a larger batch size could bring the model to teach itself a full set of samples it was uncertain about so as to eliminate the need for querying for similar samples multiple times.

Results in Figures in A.1 and in A.2 show that using a batch size of one gave the best performance in almost every Active Learning strategy. The reason we believe a batch of one gave better results is that the model will be updated more frequently. A frequent update of the model results in that the queried messages are always the most informative according to the current updated classifier. Using a larger batch size, only the first queried sample of the batch is certain to be the most informative sample, the rest of the samples in the batch might be rated less important if the classifier would be able to rank all samples after being trained on the first sample. The result corresponds with the opinion of an expert in the field who we were in contact with - Fabrizio Sebastiani - quoted several times in this paper for coauthoring the paper *Active Learning Strategies for Multi-Label Text Classification* 2009. He said that "the smaller the batch size, the better the system should be" (F. Sebastiani 2021, personal communication, 21 January).

The results we found were therefore in line with the common opinion in the field; larger batch sizes lead to worse performing active learning classification. However, in our opinions, the increasing of batch sizes *severely* deteriorate the results and capabilities of the active learning classifier as can be seen in Figures A.1 and A.2. Therefore, although its impossible for Sinch to retrain its classifier one query at a time, excessive increase in batch size should be avoided.

For further work on the batching of queries, measures to distinguish and more optimally combine samples in batches should be explored. One method we though about implementing was *edit distance*, to combine messages that are diverse, in terms of lexical content.

6.3.2 Active learning strategy comparison

The general pattern that can be observed in Figures 5.4 and 5.5 is that the random learner is scoring best in both evaluation metrics initially (at 50 samples) and then performs similar or equal to the active learning strategies after around 2000 samples. The superiority of the active learning strategies is most prominent around 500-1500 samples.

We believe active learning strategies are outperformed by random querying in the beginning of training because a majority of the active learners are sampling from the worst labels and or the worst sample of the entire data set. The strategies using MinConfidence are querying about the samples with the worst possible confidence. Naturally, during their first queries they will receive outliers that might actually worsen the classification on the test set, because they are not representative of the test set as a whole.

Random sampling, however, is trained on 50 randomly queried samples from the query pool, averaged over 5 K-folds which decreases its variance if it chooses an outlier at random. Therefore it is more likely to pick a fair representation of the validation set in the beginning. We also find that after 2000 samples, the advantage of active learning is decreasing in relation to random sampling. We believe this is due to the diminishing returns of selecting the most important samples. As the query pool shrinks and the training set increases, the chances of random and active learning picking similar samples increase.

To try to validate our ideas, we contacted Tivadar Danka, developer of ModAL as to why active learning is outperforming random sampling at certain intervals.

> "[As far as I know] there is no general theory. The point where an active learning method outperforms the random sampling depends on the set of course (...) Too few samples are not enough to cause any improvement (even if they provide the theoretically optimal improvement), but a large enough sample of new data can boost the scores even if they are selected suboptimally. So, active learning is useful when you have the capacity to label some data, but definitely cannot label all." (T. Danka 2021, personal communication, 25 January)

Additionally we contacted Andrea Esuli, also coauthor of *Active Learning Strategies for Multi-Label Text Classification* (2009), who also confirms the above statement. Esuli elaborated and explained that in our data set, evaluation values at around 500 samples queried seems to be of interest as a trade off point showing how good an active learning strategy is. After 500 samples, the classifier has trained on a large enough set of samples to gain the most out of picking informative samples yet not to large set to end up in a zone in which all strategies perform the same because they are forced to select the same samples due to lack of alternatives. (A. Esuli 2021, personal communication, 25 January)

Since the S2 data set is limited to 4000 samples in total the random sampler will eventually choose equally informative samples as the active learning strategies. For Sinch though, the amount of samples in the unlabeled pool ($\boldsymbol{L}$) far exceeds the amount possible to annotate. Therefore, we believe annotating samples at random will not realistically converge with the performance of uncertainty sampling on larger data sets.

When comparing the active learning strategies to each other, Table 5.4 show that MinConfidence (C) is performing better than Positive MaxScore (P) regardless of the combination of class and weight dimensions.

The reason we explored Positive MaxScore (P) is that positive samples are typically more informative than negative ones in a supervised learning task. Using this strategy the classifier

will certainly increase the influx of positive samples (Esuli and Sebastiani, 2009). The manual inspection demonstrated that the strategy was indeed getting positive samples, the problem is that it is already fairly convinced of the label on that sample and will therefore not gain any additional knowledge.

The weight factor, Weighting (W), had a beneficial impact on the Positive MaxScore. This encourages the learner to query positive samples from labels performing bad, though we were hoping that it would have greater impact on the final results.

The best performing strategy using Positive MaxScore (P) is in combination with Average (A) and Weighting (W). This strategy is promoting samples with a lot of positive information and promoting labels that are scoring poorly. Interestingly when looking at Figures 5.4 and 5.5 the PAW and PMW are scoring better in the F1 Macro and then worse in the EMR compared to the MinConfidence strategies. We believe that this is because the Weighting factor in the PAW and PMW are optimizing the weak classes by choosing samples from them. However, while the optimizing of these classes will increase the macro F1 score as it regards all labels as equal, the more strict EMR score requires all classes to be correctly predicted for a good score. As weighted P strategies focus more on the weak positive samples, it may be too skewed towards picking the less frequent samples and consequently not making a lot of perfect predictions on frequently occurring message templates.

The observations suggest that it is more valuable for the classifier to query for samples it is uncertain about, we suggest that these indications are enough to confidently disregard Positive MaxScore as the evidence dimension of choice.

Regarding the class dimensions, it is hard to draw any clear conclusion, especially because of the volatility of the curves in Figures 5.4 and 5.5.

If we compare the strategies to each other in regards to each other based on the class dimension only. That is, comparing CAN to CMN, and PAN to PMN etc. We can see that in that for every strategy in Figure 5.4, the class dimension *Average* (A) outperforms *Max* (M). One exception would be CAW and CMW where CMW is actually performing better in earlier training iterations, but CAW performs better after 1000 samples. These observations are confirmed in Table 5.4. In Figure 5.5, we find that the opposite is true for some of the strategies - e.g. CMN ourperforms CAN and CMW outperforms CAW. Thus, it is difficult to make a clear decision about what is the best option for class dimension. But, it seems as though *Max* is a better option if stricter prediction precision is required, which is what EMR1 symbolizes.

We follow the same structure for the *Weighting* dimension. Here, we find that for *Positive MaxScore* strategies, *Weighting* exceed *Non-Weighting* in all cases. No similar conclusions can be drawn about the *MinConfidence*. In Figures 5.4 and 5.5, CAN seems better than CAW for both F1 macro and EMR, while CMN performs worse than CMW on F1 Macro, but better on EMR.

Finally, three strategy combinations have emerged as candidates for the best active learning strategy: CAW, CMN, and CAN. All three of them scored well in terms of overall performance, but they are individually exceeding one another in certain intervals in either F1 macro or EMR.

CAW scores best in one of the results in Table 5.4: F1 (1500). By looking at Figure 5.4, it appears stable and persistently has the highest results in F1 macro after 1000 samples. It performs relatively well in EMR as well, but is in general slower than CMN and CAN.

CMN achieved the best results in three places in Table 5.4: EMR (500), EMR (1000), and

EMR Area (500-1000). In particular, the results show that CMN is able to quickly increase the exact match ratio of its predictions. In F1 macro, it does not perform as well. After 1000 samples, it is only the fourth best strategy, and ends up being the worst after all samples are trained on.

Lastly, CAN proves to be a little more versatile than CAW and CMN. CAN scores the best in F1 (500), F1 (1000) and EMR (1000), and EMR (1500). Thereby, making it the most favorable option judging from Table 5.4. Figure 5.4 show that it learns quickly and performed best, especially around 500 samples. Regarding EMR in Figure 5.5, it is notably slower than CMN and RAND, but after 1000 samples it is among the best options.

In short, it is not clear which option is the superior one. If stable performance is important, CAW is the best option. If quick and training and exact predictions are important, CMN is better. If more versatility, quick F1 macro increase and stable performance is necessary, CAN would be the choice. Because CAN proved to be the best option judging from Table 5.4 and because of it doing well in both F1 macro and EMR, CAN would be our final choice of model.

Weighting constant

In our strategies using *Weighting* (W) we had to add a constant that served as a buffer when the F1 score of a label was zero. This is especially critical in a strategy such as MinConfidence Max Weighting (CMW). If a class label has zero in F1 macro score, CMW will randomly query any label of that class since there is no distinction between the actual confidence of the samples since they are all set to zero.

The constant added was set arbitrarily to 0.001, since we believed such a small number relatively to the other F1 scores would not affect any other weighting than the weight for the label with F1 score zero. Another possibility would have been to use a LaPlace smoother as used in the report by Esuli and Sebastiani (2009).

Evaluation metrics

The main performance metrics used were F1 macro, F1 micro, EMR1 and EMR2. Most emphasis was put on F1 macro and EMR2 since we chose to implement the optimization of the threshold in our experiments. Moreover, the micro score turned out to be less valuable for us than initially expected. The decision to focus more on F1 macro was mainly due to the preference of our supervisor and conformity with other research papers in which F1 macro seemed more prominent.

The problem with measuring active learning is that in comparison to regular classification optimization, the accuracy at convergence is not important. Therefore, much more emphasis is put on analysing the plots to see how fast the models are trained. However, because many of the active learning strategies were difficult to discern from one another in our Figures, we wanted to measure exact numbers to get more specific data on performance. But, this created a problem of choosing which particular points to look at. Our choice to measure the metrics on 500, 1000, and 1500 samples stemmed from our discussions with experts. Still, the numbers in Table 5.4 might be misleading since the performance exhibit a lot of volatility. Therefore, we introduced the area score.

The area score metric used in table 5.4 is an complement to the other metrics when the

results are as volatile as ours were. It removes the bias of choosing one single point to compare our different strategy methods as it averages the result over more training iterations and thus balances out if a strategy exhibits anomalies in the results of the point chosen. Moreover, the area metric allows for specifically choosing an interval to measure the performance of. In our case, we believed 500-1000 samples to be the most interesting interval to look at for active learning purposes. Although we believe the metric provided us with some insight, we have not found any other author using it in peer-reviewed articles and therefore consider it of less importance when evaluating the strategies.

6.3.3 Individual Label Performance

After deciding the optimal strategy - CAN - we compared the individual labels' macro F1 increase. In Table 6.1 the six different classes of varying lexical diversity and imbalance from S2 are compiled. This was done using the data from Table 2.3, Table 2.6 and table 5.6.

Table 6.1: Extracted labels F1 macro increase of CAN in S2 in comparison to imbalance ratio and lexical diversity.

	2FA	**Mark.**	**Not.**
Imbalance ratio	52.8	6.6	1.8
Lexical Diversity	0.279	0.164	0.129
F1 Increase	0.061	0.183	0.012
	Fin.	**N.English**	**P.I.**
Imbalance ratio	311.5	587.2	311.5
Lexical Diversity	0.459	0.480	0.459
F1 Increase	0.450	0.562	0.697

In the top half of Table 6.1 we can see that the three labels with low imbalance ratio and lexical diversity do not improve remarkably in F1 macro. The lower half of Table 6.1 demonstrate the opposite attributes. A large lexical diversity and imbalance ratio seem to indicate a large increase the individual F1 macro. 2FA is an exception to this, because despite having a high imbalance and being relatively lexically diverse, it still does not show a significant F1 increase. Most likely this is a consequence of 2FA having a very low imbalance and lexical diversity in S1. Thus, the total variation of the two data sets combined show a low imbalance rather than a high (see Table 2.6 and 2.3). Moreover, the frequent presence of 2FA messages consequently created a well fitted model for S1. Therefore, there was less room for improvement when training on S2.

For a classifier, it is easier to make predictions where the lexical diversity is low, for example if several messages follow similar formats. In those cases, the active selection of training samples is not as important. Or as our supervisor Pierre Nugues put it about a label with low lexical diversity and imbalance ratio: "After the model has seen a message once, it has basically seen all of them [of that label]." (P. Nugues 2021, personal communication, 27 January)

This shows that active learning is a strategy that most favorably should be applied in imbalanced and complex data sets.

6.4 Data Sets

During this project, have used three different data sets: S1, S2, and RCV1. From the beginning, we wanted the Sinch data sets to be fairly imbalanced, in order to see how well active learning could select samples. Our reasoning was that in a perfectly balanced data set, with little variation of class complexity, random sampling should generate a relatively good training set. With an imbalanced and less uniform class distribution though, it should be more important to make informed decisions about which samples to train on. This reasoning was supported by our conclusions in the previous section. The data sets were also rather representative of Sinch's data flow, where the active learning is supposed to be implemented.

However, the Sinch data sets might have proven to be *too* imbalanced. In Table 2.4 we find that the difference in imbalance ratios for RCV1 and the Sinch sets are very high. Moreover, S1 and S2 are more imbalanced in total, and for every class, than any of the 13 data sets which Zhang et al. (2020) investigated in his paper *Towards Class Imbalance Aware Multi-Label Learning*. The high presence of labels like *Two Factor Authentication*, *Banking*, *Alert* and *Notification*, in relation to less frequently occurring classes such as *Non-English* and *Public Information* may have hindered us from making robust conclusions on the performance of active learning on the different classes.

The imbalance could have been reduced by e.g more preprocessing. Other approaches are discussed in e.g. Zhang et al. (2020), where utilizing correlations between classes is presented as one way of ameliorating the problems of class imbalance. However, we finally opted for continuing in line with our primary intuition of investigating the imbalanced data sets more or less ass they were. And, although there would have been benefits with reducing imbalance, their characteristics lead us to make some conclusions, such as those in the previous section.

6.5 Further work

The most important aspect to further build our thesis would be to expand the tests and ensure more reliable statistical proof of our conclusions. The small size of the data sets and their imbalance created an interesting environment for a case study, but the results can hardly be generalized. For future work, more experiments will need to be carried on larger and more balanced data sets to validate our results. One way of expanding the data sets would be to use data from different time intervals to better represent the Sinch data as a whole. Another way to validate our results would be to run the same tests on the Reuters data set.

Also, the model we used - LogReg - is basic and not likely to be used by Sinch in practise. It would therefore be interesting to see if our conclusions regarding active learning can be translated to a more advanced classifier.

Possibly our most interesting conclusions regarded the connection between label imbalance, lexical diversity, and F1 macro increase using active learning. Intuitively it seems reasonable that it is more important to pick the right samples when the data set is imbalanced and complex, than when it is uniform and simple. It would be interesting to test those connections in more detail - and specifically to look at the performance of individual labels as it might provide insight as to under what conditions active learning is most favorably utilized.

We would like to have analyzed what would happen if we let samples that the model is most certain about be directly annotated as the predicted class labels and added to the

training set. This field, called *semi-supervised learning*, is related, but in some sense the opposite of active learning. 1. because the model picks the most confident classes and 2. because the model assumes that its predictions are correct, thus eliminating the need of an *oracle*. The strategy is somewhat similar to the evidence dimension *Positive MacScore*, and thus overfitting the model is a big risk. What might be interesting is to see what a combination of semi-supervised learning and uncertainty sampling active learning would accomplish. Intuitively it would be an interesting combination: querying the *oracle* on the *least* confident predictions and assuming certainty of class membership on the *most* confident predictions.

Another possibility would be to explore how active learning could aid in learning a newly introduced label in the training set. In reality this is a situation that is likely to occur at Sinch, especially with their rapid expansion and new customers that send different types of messages. Here, maybe a different choice of strategy combination would perform and adapt better to incorporate the new labels. When labeling the second data set we actually introduced more labels, preparing to explore this situation but with the time limitation we were not able to go through with the testing.

Finally, the trade off between time efficiency and accuracy should be further investigated. Our study has shown that active learning does lead to more accurate classification. However, it is more time consuming. Before incorporating it in commercial applications, it must be explored whether the gain in precision and accuracy is worth the loss in time. Active learning is inherently less time efficient than random sampling. This is especially true for smaller batch sizes, since more iterations have to be completed by the learner.

6.6 Conclusion

This thesis has evaluated possibilities of improving multi-label classification of short text messages using active learning. The first part of the project concerned finding a suitable classifier which was both accurate and fast in making predictions. After training the models on S1 and parallel testing them on RCV1, we finally settled for LogReg. In the tests on S1, LogReg was the overall quickest of the options and achieved good performance results in F1 macro (0.928) and exact match ratio (0.926).

The active learning experiments concerned the testing of uncertainty sampling in combinations of two variations in three dimensions: *MinConfidence* (C), *Positive MaxScore* (P), *Average* (A), *Max* (M), *Weighting* (W), and *NoWeighting* (N). The comparisons were done through experiments for each of the eight combinations and a random sampler. In addition to the combinations different dimensions, each strategy was also tested on five different query batch sizes: 1, 5, 10, 25, and 50.

Our findings show that actively choosing samples outperforms random sampling for every *MinConfidence* strategy. Thus, proving that making informed decisions about which samples to label will substantially improve time efficiency and performance of the training of a classifier. Our results also show that batching queried samples decreases F1 macro and EMR performance. Moreover, the testing of individual labels indicate a correlation between the imbalance and lexical diversity of class labels and increase of F1 macro in training.

Concerning the best active learning strategy, our results indicate that *MinConfidence*(C), *Average*(A) and *NoWeighting*(N) is the best combination on the data set. CAN performs well on different metrics and gets the best overall results, as seen in Table 5.4. We want to be

cautious though and not appoint a single strategy as the best, because the strategies perform well on different metrics and may display different results on another data set.

As a final remark, it must be reasserted that our findings are not decisive in deciding the optimal active learning strategy for multi-label classification of short text messages. The data sets that were used were too imbalanced and small to create any general conclusions. Further work with larger and more balanced data sets need to be carried out in order to derive more nuanced and robust conclusions on the matter.

References

Alammar, J. (2018). The Illustrated Transformer [Blog post]. https://jalammar.github.io/illustrated-transformer/. [Online; accessed 2021-02-05].

Allard, M. (2019). What is a transformer?[blog post]. https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04. [Online; accessed 4-March-2021].

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

Chang, Y.-W., Hsieh, C.-J., Chang, K.-W., Ringgaard, M., and Lin, C.-J. (2010). Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*, 11(4).

Chen, T. and Guestrin, C. (2016). Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Chollet, F. (2018). *Deep Learning with Python*. Manning Publications.

Commons, W. (2020). File:kernel yontemi ile veriyi daha fazla dimensiyonlu uzaya tasima islemi.png — wikimedia commons, the free media repository. [Online; accessed 14-February-2021].

Danka, T. and Horvath, P. (2018). modAL: A modular active learning framework for Python. available on arXiv at https://arxiv.org/abs/1805.00979.

Davenport, T. H., Barth, P., and Bean, R. (2012). How 'big data' is different.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

Esuli, A. and Sebastiani, F. (2009). Active learning strategies for multi-label text classification. In Boughanem, M., Berrut, C., Mothe, J., and Soule-Dupuy, C., editors, *Advances in Information Retrieval*, pages 102–113, Berlin, Heidelberg. Springer Berlin Heidelberg.

Gerniers, A. and Saerens, M. (2018). Maximum entropy method for multi-label classification. *Master degree thesis, EPL, Université Catholique de Louvain*.

Koehrsen, W. (2017). Random Forest Simple Explanation. `https://williamkoehrsen.medium.com/random-forest-simple-explanation-377895a60d2d`. [Online; accessed 2021-02-03].

Kyle, K. (2020). lexical-diversity. `https://pypi.org/project/lexical-diversity/`. [Online; accessed 28-February-2021].

Labrinidis, A. and Jagadish, H. V. (2012). Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12):2032–2033.

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397.

Liddy, E. D. (2001). Natural language processing. *Encyclopedia of Library and Information Science, 2nd Ed.*

Plisson, J., Lavrac, N., Mladenic, D., et al. (2004). A rule based approach to word lemmatization. In *Proceedings of IS*, volume 3, pages 83–86.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2020). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Schein, A. I. and Ungar, L. H. (2007). Active learning for logistic regression: an evaluation. *Machine Learning*, 68(3):235–265.

Settles, B. (2009). Active learning literature survey. .

Sorower, M. S. (2010). A literature survey on algorithms for multi-label learning.

Syakur, M. A., Khotimah, B. K., Rochman, E. M. S., and Satoto, B. D. (2018). Integration k-means clustering method and elbow method for identification of the best customer profile cluster. *IOP Conference Series: Materials Science and Engineering*, 336:012017.

Tran, M. and Truong, M. (2019). Clustering short text messages using unsupervised machine learning.

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

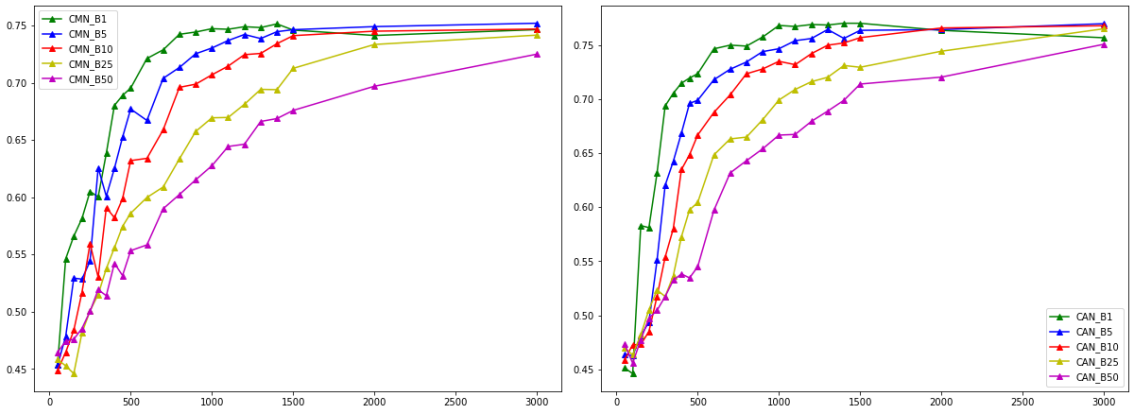
Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Łukasz Kaiser, Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, 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.

Yildirim, S. (2020). K-means Clustering Explained. `https://towardsdatascience.com/k-means-clustering-explained-4528df86a120`. [Online; accessed 17-February-2021].

Zhang, M.-L., Li, Y.-K., Yang, H., and Liu, X.-Y. (2020). Towards class-imbalance aware multi-label learning. *IEEE Transactions on Cybernetics*.

Zhang, M.-L. and Zhou, Z.-H. (2013). A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering*, 26(8):1819–1837.

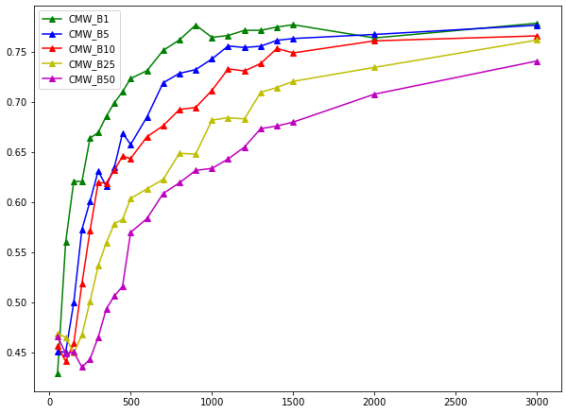
Appendix A
Graphs



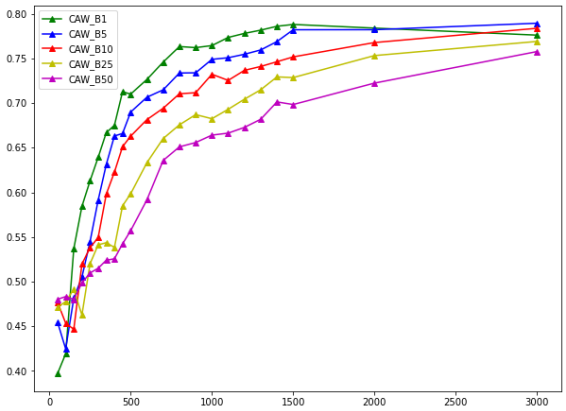
(a) MinConfidence Max NoWeighting **(b)** MinConfidence Average NoWeighting

Figure A.1: F1 Macro of the eight active learning strategies plotted over queried samples with different batch sizes. Note that the y-axis is not fixed.

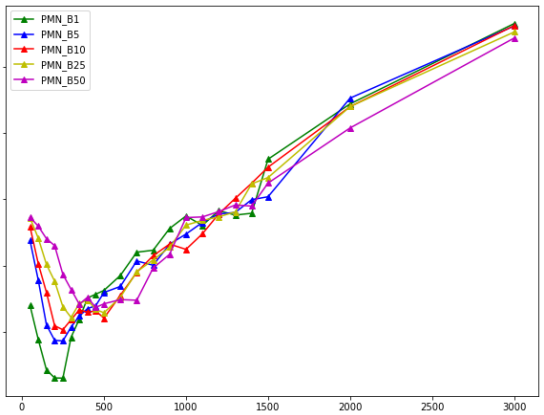
.pdf" .png" .jpg" .mps" .jpeg" .jbig2" .jb2" .PDF" .PNG" .JPG" .JPEG" .JBIG2" .JB2" .eps"



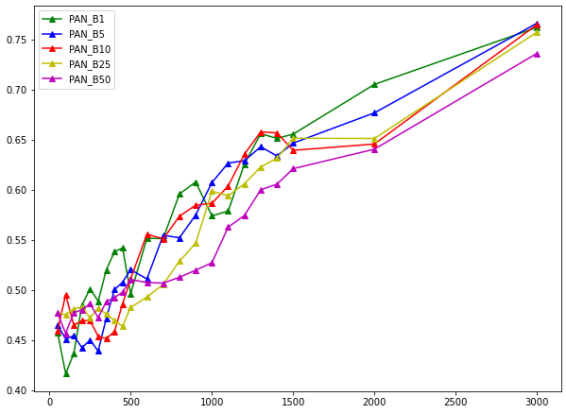
(c) MinConfidence Max Weighting



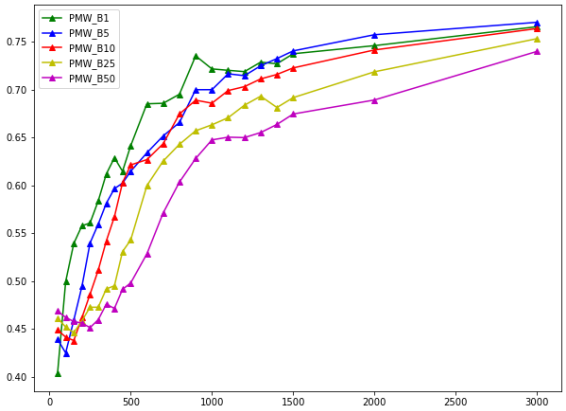
(d) MinConfidence Average Weighting



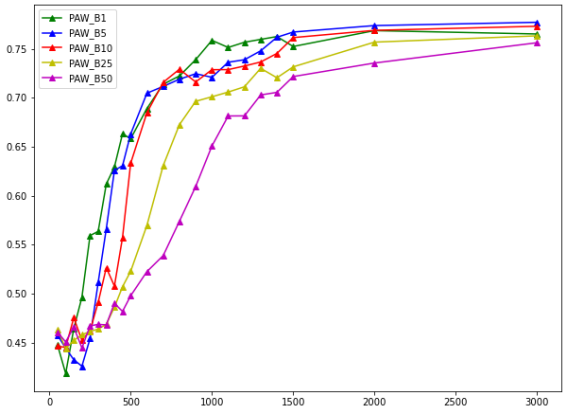
(e) Positive MaxScore Max NoWeighting



(f) Positive MaxScore Average NoWeighting

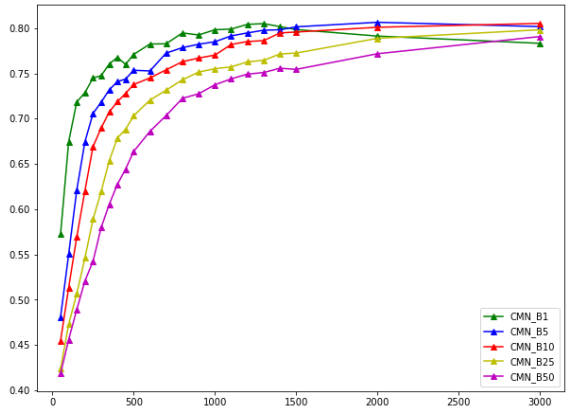


(g) Positive MaxScore Max Weighting

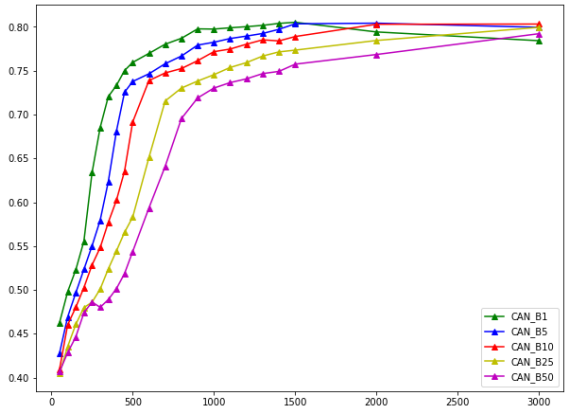


(h) Positive MaxScore Average Weighting

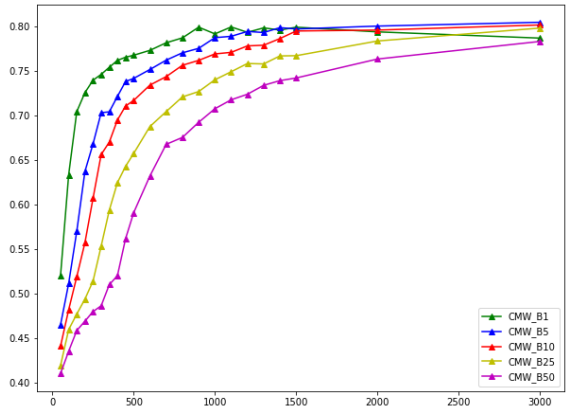
Figure A.1: F1 Macro of the eight active learning strategies plotted over queried samples with different batch sizes. Note that the y-axis is not fixed. (Continuation)



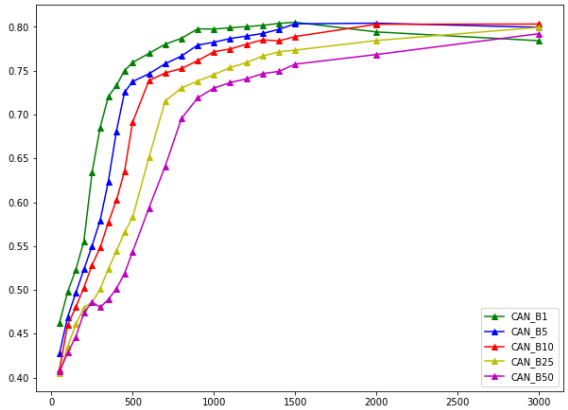
(a) EMR score of CMN - MinConfidence MaxNoWeighting



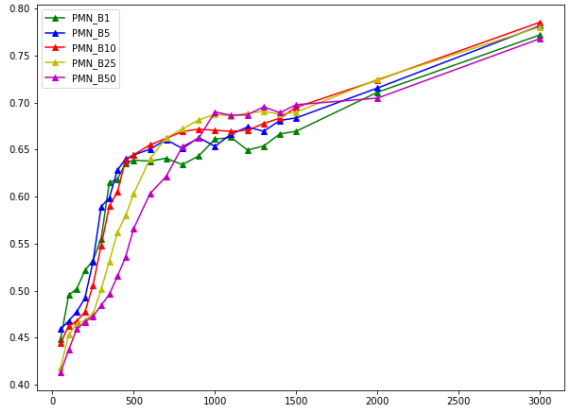
(b) EMR score of CAN - MinConfidence AverageNoWeighting



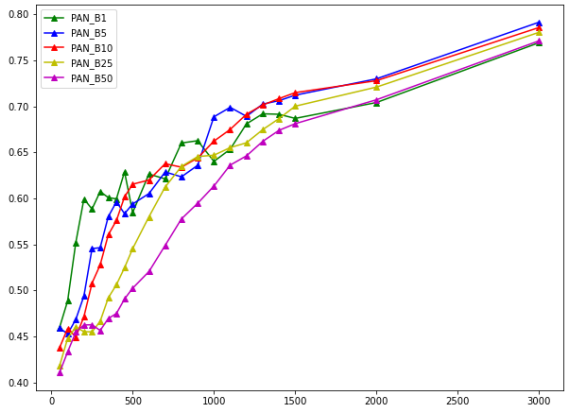
(c) EMR score of CMW - MinConfidence MaxWeighting



(d) EMR score of CAW - MinConfidence AverageWeighting

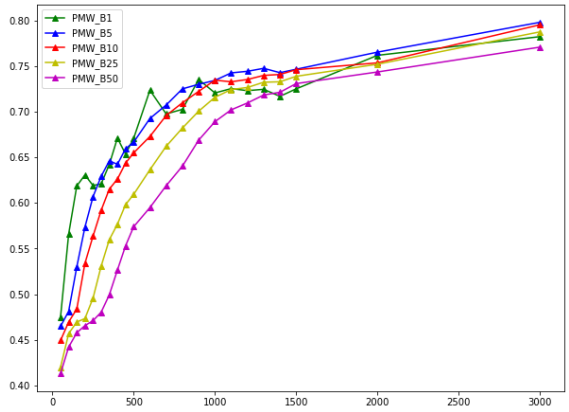


(e) EMR score of PMN - Positive MaxScore MaxNoWeighting

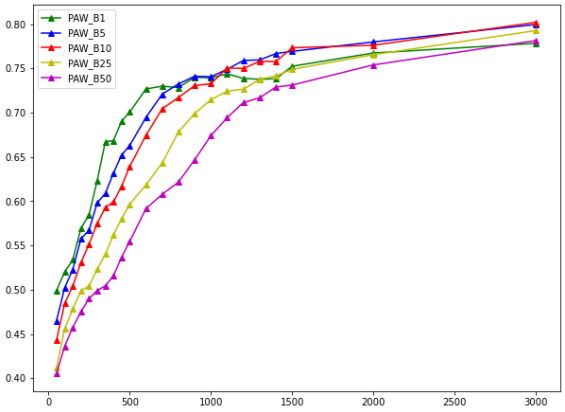


(f) EMR score of PAN - Positive MaxScore AverageNoWeighting

Figure A.2: EMR of the eight active learning strategies plotted over queried samples with different batch sizes.



(g) EMR score of PMW - Positive MaxScore Max Weighting



(h) Positive MaxScore(P) Average(A) Weighting(W)

Figure A.2: EMR of the eight active learning strategies plotted over queried samples with different batch sizes. (Continuation)

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2021-03-11

EXAMENSARBETE Evaluation of Active Learning Strategies for Multi-Label Text Classification
STUDENTER Henric Zethraeus, Philip Horstmann
HANDLEDARE Pierre Nugues (LTH)
EXAMINATOR Jacek Malec (LTH)

Kan active learning hjälpa oss att optimera maskininlärningsalgoritmer i en till synes oöverskådlig skog av ansamlad data?

POPULÄRVETENSKAPLIG SAMMANFATTNING **Henric Zethraeus, Philip Horstmann**

IT företag samlar på sig allt mer data och de allra framgångsrikaste företagen är de som kan tolka datan och översätta den till värdefulla marknadsinsikter. Problemet med den nästan gränslösa informationsinsamlingen är att mängden data blir svårhanterlig. I detta arbete har vi utvärderat *active learning*, en metod för att på ett kostnadseffektivt sätt låta maskininlärningsalgoritmer bestämma vilken data som är mest användningsbar för att optimera klassificering.

Ponera en aspirerande botaniker som vill lära sig trädarter genom att vandra omkring med en botanikprofessor som pekar ut träd i en skog och därmed lär eleven känna igen egenskaper hos dem. Detta kan liknas vid det traditionella sättet på vilket *supervised machine learning* (vägledd maskininlärning) genomförs. Alltså, en kunnig människa (botanikprofessorn) annoterar data (träd) för en dator (elev) som därefter lär sig att kategorisera objekten.

Det finns dock ett problem med detta tillvägagångssätt. Botanikprofessorns tid är begränsad och att berätta om alla världens träd är kostsamt, om inte rent omöjligt. I maskininlärning löses detta genom att skapa ett mindre *training set* d.v.s. ett annoterat subset av objekten som önskas läras. Oftast består denna urvalsgrupp av slumpmässigt utvalda – i denna analogi träd i skogen – som sedan annoteras av professorn. Även denna metod är suboptimal för att det kan vara så att för få (eller inga) träd väljs av vissa sorter och för många av andra, vilket leder till att eleven inte får en optimal inlärningsprocess. Det är ju onödigt om professorn pekar ut en ek flertalet gånger trots att eleven redan vet hur en ek ser ut. Och motsatt om professorn i sin urvalsgrupp inte har med en enda björk att lära eleven, så att eleven aldrig tillskansar sig den kunskapen.

Ett potentiellt sätt att i sin tur förbättra denna process vore därför om eleven istället för professorn fick välja vilka träd som hon känner sig osäker på. Alltså, eleven går först igenom skogen, iaktar vilka träd hon känner sig mest osäker på och väljer därefter den urvalsgrupp som professorn får lära eleven. Denna metod inom maskininlärning heter *active learning* och vår studie har undersökt huruvida denna metod är fördelaktig i jämförelse med slumpmässigt urval av träningsdata.

Problemet med annotering av data i supervised machine learning genomsyrar dagens techföretag i ett klimat där icke-annoterad data är lättillgänglig, men annoterad data är svårtillgäng-

EXAMENSARBETE Evaluation of Active Learning Strategies for Multi-Label Text Classification

STUDENTER Henric Zethraeus, Philip Horstmann

HANDLEDARE Pierre Nugues (LTH)

EXAMINATOR Jacek Malec (LTH)

lig. Den ickeannoterade datan finns i överflöd och kan bidra med mycket positiv information om bland annat användar- och marknadsbeteenden, om bara effektiv annotering och urvalsprocesser kan uppnås.

I vårt arbete studeras SMS distribuerade av Sinch AB. Dessa SMS skickas kontinuerligt till kunder runt om i världen och uppgår i miljarder till antal och kan därför omöjligt kategoriseras manuellt. Vi studerade metoder för *multi-label text classification*, där målet är att kategorisera varje SMS i en eller flera kategorier samtidigt. Efter val av klassificerare undersökte vi hur precis vår klassificering blev beroende på om training-setet valdes slumpmässigt eller genom active learning.

För active learning krävs att klassificeraren först tränas på ett slumpmässigt utvalt mindre dataset. Därefter väljer klassificeraren ut vilka SMS den vidare vill tränas på av en större pool av oannoterade meddelanden. Det finns en mängd sätt på vilka klassificeraren kan välja ut vilka SMS som potentiellt kan ge den mest information. Vi fokuserade på *uncertainty sampling*, d.v.s. att man antar att de mest informativa SMSen är de som klassificeraren är mest osäker på.

Våra resultat visar tydligt att active learning med uncertainty sampling är bättre än slumpmässiga urvalsgrupper, både i fråga om precision och noggrannhet. Specifikt visade sig logistic regression med CAN (Min-Confidence, Average, Non-weighting) som active learning strategi vara den bästa metoden, vilken beskrivs med större noggrannhet i vår rapport.

Följaktligen kan det därmed fastslås att, ja, det är bättre om eleven får välja vilka träd hon vill lära sig än om professorn pekar ut träd slumpmässigt.