NOVA IMS Information Management School

MGI

Mestrado em Gestão de Informação
Master Program in Information Management

Multi language Email Classification Using Transfer learning

Mário Jorge Carvalho de Sousa

Dissertation presented as Master Degree in Information Management, with a specialization in Knowledge Management and Business Intelligence

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

MULTI LANGUAGE EMAIL CLASSIFICATION USING TRANSFER LEARNING

by

Mário Jorge Carvalho de Sousa

Dissertation presented as Master Degree in Information Management, with a specialization in Knowledge Management and Business Intelligence

Advisor: Roberto Henriques

Co Advisor*:* Ricardo Rei

ACKNOWLEDGEMENTS

I would like to express my gratitude to Cleverly for trusting me with this project and the chance of working with them as well as providing me with all the materials needed for its successful conclusion. A special thanks to Mariana Almeida for the sincere and tireless support, valuable time, knowledge sharing, and mentoring through this journey. Thanks to my co-advisor Ricardo Rei for introducing me to the wonderful world of NLP through his amazing teaching, transmitted knowledge, and key insights throughout the project. Thanks to my advisor professor Roberto Henriques for the trust and freedom given throughout the project as well as for providing me with key insights for its successful conclusion. This thesis has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 873904.

ABSTRACT

In recent years Artificial Intelligence has become a core part of many businesses, from manufacturers to service providers, AI can be found helping to improve business processes as well as providing customized experiences and support for the customers. Natural language processing gives computers the ability to understand human language, recent breakthroughs in multilingual models bring us closer to overcome language barriers and achieve various tasks regardless of the language. This brings to companies the opportunity to process data and provide services to customers regardless of their language. In this dissertation, we review the progress of NLP towards multilingual text classification, our results suggest that using a machine translation to augment our corpora is a suitable approach to fine-tune multi-language models like XLM-Roberta, obtaing better results than zero-shot approaches. Our results also suggest that in domain pre-training can help to increase the performance of the classification for both monolingual and multi-language classifiers

KEYWORDS

Multi-Language Text Classification; Supervised Machine Learning; Deep Learning; Text Classification; Natural Language Processing; Customer Support;

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

MLP	Multi-Layer Perceptron
RAM	Random Acess Memory
CPU	Central Processing Unit
VRAM	Video Random Acess Memory
GPU	Graphic Processing Unit
NN	Neural Network
RELU	Rectified Linear Unit function
MTL	Multi-Task Learning
LSTM	Long Short Term Memory
RNN	Recurrent Neural Network
RCNN	Recurrent Convolutional Neural Network
MLM	Masked Language Modeling
SVM	Support Vector Machines
NB	Naive Bayes
BERT	Bidirectional Encoder Representations from Transformers
MBERT	Multil-language BERT
RoBERTa	Robustly Optimized BERT Pretraining Approach

1 INTRODUCTION

In today's competitive market long term customer relationship is an integral part of a successful enterprise, it is also known that customer retention leads to profit increase, making it important to determine the factors that influence the longevity of the relationship between customer and service provider(Dovaliene et al., 2007). Customer support has an important role in maintaining a relationship between customers and companies, almost every company either manufacturer or service provider is performing some kind of customer support, the extent of this service can be specific and tailored for each customer or more general like repairing a piece of faulty hardware (Qasim & Asadullah, 2012). (Loomba, 1998) defines customer support as simply ensuring that a determined product is usable by the customer during the product's life span. (Coffin & New, 2001) complements saying that that customer support can be used to add value to a product and achieve customer satisfaction. However, customer support does not come without monetary and human resources, thus the use of AI can facilitate the streamline of processes and help to reduce costs.

The introduction of Artificial Intelligence to help with customer support ticket triage raises from the need to find a more efficient way to perform a repetitive and expensive problem. By eliminating the need of having agents dedicated to reading and classifying customer emails according to a set of topics we can let the agents focus on more meaningful tasks that AI cannot yet tackle. According to (Sarah Patterson, 2019) only 24% of the businesses in 2019 use AI to help in customer support tasks, however, the same study tells us that 56% of the services decision-makers are actively looking for ways to integrate AI.

As the amount of text data generated increases due to the extensive use of digital platforms for communication proposes, raises the need for classifying said data. On the topic of emails, the (The Radicati Group, 2015) shows that the daily email traffic is increasing every year reaching over 246 billion of emails sent/received per day in 2019. Has customers extensively use email services to reach companies, managing such emails is an important topic for the companies increasing the importance of automated email classification solutions.

As the globalization of markets speeds up and companies start to expand beyond the boundaries of their native languages the need for models capable of handling multiple languages also increases, traditional automated classification solutions are only able to perform in monolingual scenarios and due to complexity, implementing a classification model for each language can be difficult to manage, some languages also lack suitable data to train classification models. As the need for multilingualism raises a particular neural network architecture known as transformers is proving to be especially effective in handling multi-language tasks. These challenges that modern-day companies face serves as motivation for this dissertation

This thesis Goal is to train a classifier capable of classifying customer support emails in multiple languages according to a given set of topics. The project can be divided into two phases. In the first phase, the objective was to build an efficient monolingual English classifier that would serve as a baseline for the multilingual model. In the second phase, we experiment with various models and training approaches to fine-tune a multi-language classifier. Throughout the development of the project, several models have been tested as well as the effect of various hyper-parameters on the model's performance. This thesis aims to answer the following questions.

How well can transformers perform on customer support data classification?

Is it possible to build a performing multi-language classifier by training it on machine-translated data?

What is the performance trade-off between monolingual and multi-lingual models?

2 BACKGROUND

This section provides the reader with the fundamental concepts needed to fully understand the rest of the document. In section 2.1 the concept of machine learning is introduced. Section 2.2 establishes the fundamentals of neural networks as well as the different architectures. In section 2.3 we present different methods of representing textual information.

2.1 MACHINE LEARNING

Machine learning is a field of study that focuses on the research and development of algorithms that are capable of learning with experience (Mitchell, 1997). We can sub-classify machine learning in supervised and unsupervised depending on how the supervision of the training phase is done.

In supervised machine learning algorithm learns from pre-labeled examples known as ground truth, i.e observations of a training dataset that is annotated with a predefined set of possible category values. After training a model on the labeled training dataset, The goal of the machine learning algorithm is to, predict the labeling for the unseen un-labeled data.

In unsupervised machine learning, the data is not previously labeled in any anyway, the machine learning algorithm tries to learn by finding common patterns in the data and grouping it accordingly.

2.2 NEURAL NETWORKS

An artificial neural network or ANN is a type of machine learning model inspired by the neurobiology understanding of the human brain. In 1943 Warren McCulloch and Walter Pitts proposed the first computational model for a neuron. Inspired by the McCulloch-Pitts neuron Frank Rosenblatt proposed in 1957 what is the basic element of a neural network the perceptron.

2.2.1 Perceptron

Has previously mentioned the most basic unit of a neural network is a perceptron. Formally represented by the equation.

$$z = \sum_{i=l}^{N} x_i w_i + b$$

The perceptron takes an x vector of length N as input and calculates the product of each value of x with the corresponding weight of the matrix w, the resulting product is then summed to the bias b which typically has the value of 1, The bias term allows the decision boundary to be shifted. The output z is then passed through an activation function, this activation function can be changed according to the problem needs.

2.2.2 Multi-layer Perceptron

Multi-layer Perceptron or MLP is neural architecture composed of multiple perceptrons divided into at least three distinct layers; an input layer, one or multiple hidden layers, and the output layer, in other words, it can be seen as a logistic regression where the input is non-linearly transformed. The

number of perceptrons present in the hidden layers can be changed arbitrarily. Like in most neural networks the training of the MLP consists of two main phases that can be repeated N times where N is a number defined arbitrarily. In the first phase the forward pass, the input signal is propagated through the network. The inputs are multiplied with the respective weights and added to a bias, this process is repeated until it reaches the end of the neural network where the outputted probabilities will be compared with the ground truth and a loss is computed. In the second phase, the backward pass the loss is propagated backward through the network, in this phase, the model adjusts the learnable parameters to minimize the loss, this is done using gradient descent.

2.2.3 Optimizers

The optimization algorithm is a vital part of the neural network architecture helping the model to minimize the error function.

Adam optimizer or Adaptive Moment estimation proposed by (Kingma & Ba, 2015) is an optimizer specifically designed for training machine learning algorithms, has the name infers Adam computes different learning rates for each learnable parameter. The Adam optimizer is heavily inspired by momentum and RMSdrop optimizers

AdamW or Adam with weight decay is proposed by (Loshchilov & Hutter, 2019), this optimizer as the name indicates is a variation of the original Adam optimizer, AdamW detaches the weight decay from the optimization steps allowing the weight to be optimized independently from the learning rate.

2.2.4 Recurrent Neural Network

Unlike vanilla neural networks, Recurrent neural networks are capable of processing sequential data using internal memory.

$$h_{(t)} = f\left(U * x^{(t)} + W * h_{(t-1)}\right)$$

At time-step t the Hidden state h_t is calculated using an input $x^{(t)}$ and the previously hidden state $h_{(t-1)}$. U and W are weight matrixes that transform$x^{(t)}$ and $h_{(t-1)}$ linearly. These linear transformations are then passed through a non-linear function f in order to produce the next hidden state.

2.2.5 Long Short-term memory networks

In order to address the vanishing gradient problem (Hochreiter & Schmidhuber, 1997) proposed the LSTM architecture.LSTM's are composed of cells and three types of gates named, Forget gate, Input gate, and Output gate.

The Forget gate controls which values shall be kept or erased from the previous cell state. For this task it uses a sigmoid function, that looks at the previous state $h_{(t-1)}$and the input $x_{(t)}$ the output is a number between 0 and 1 for each number in the cell state $C_{(t-1)}$. The output will decide whether the information will be kept (1) or omitted (0).

$$f_t = \sigma\left(W_f . [h_{t-1}, x_t] + b_f\right)$$

The input gate decides what values from the input should be stored. This is done using two layers. The sigmoid layer that decides which values will be inputted and a tanh layer that creates a set of candidates $\tilde{C}_{(t-1)}$.

$$\tilde{C}_t = \tanh\left(W_C.[h_{t-1}, x_t] + b_C\right)$$

$$i_t = \sigma\left(W_i.[h_{t-1}, x_t] + b_i\right)$$

The old state $C_{(t-1)}$ is then multiplied by f_t in order to erase the values marked to be omitted by the forget gate. To store the new values the result of the previous operation is added to the product of the input values i_t with the candidate Values $\tilde{C}_t$

$$C_t = f_t \,.\, C_{t-1} + i_t \,.\, \tilde{C}_t$$

As the name indicates the output gate will decide the output, it does so by making use of a sigmoid layer that will decide what parts of the cell state will be outputted. A function $tanh$ is then applied to the cell state C_t and multiplied with the output of the sigmoid layer in order to output the marked values.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = \; o_t \,.\tanh(C_t)$$

2.2.6 **Sequence-to-Sequence models**

Sequence-to-Sequence is an architecture proposed by (Sutskever et al., 2014) in order to solve problems where the output sequence length is different from the input sequence length. This is particularly useful for machine translation, where the length of an input sequence in a given language does not have the same length as the corresponding translation in the target language.

The architecture uses two RNN models often called the encoder and the decoder. The encoder maps the input sequence to a fixed-length vector, the vector is then passed as an input to the decoder that maps it to a variable-length vector.

However, this architecture may have problems resulting from long-term dependencies, to solve this, the author suggests the use of LSTMs since as we have seen in the previous section these models are best suited for this particular problem. The decoder-encoder are jointly trained to maximize the log probability formally represented in the equation below.

$$1/|S| \sum_{(T,S)\in S} log\ P(T|S)$$

Once the training is complete, we can get the translation or the most likely sequence:

$$\hat{T} = \arg\max(\mathrm{T})\, p(T|S)$$

This is done by using a left-to-right beam search decoder. This method maintains a small number of B hypotheses, when all the B hypothesis are generated only the most likely hypothesis is not discarded.

2.2.7 Transformers

Transformers are an encoder-decoder architecture proposed by (Devlin et al., 2019) the model explores the concept of self-attention. The self-attention mechanism resembles the way humans read, shifting their attention from word to word depending on its importance for a given context.

For each imputed token, three matrixes are created Query, Key, and Value, in order to generate these matrixes an input value X is multiplied by a weight Wq generating the Query Q matrix, the Query matrix is then multiplied by Wk to generate the Key K matrix and finally, the Key matrix is multiplied by Wv in order to generate the Value V matrix. Wq, Wk, and Wv are randomly initialized and are trainable parameters. In order to produce a matrix Z the following function is applied.

$$softmax(\frac{QK^T}{\sqrt{d_k}})V$$

The Z matrix that results from each word is then summed and feedforwarded trough the network.

In the multi attention head case, the entire process is re-done generating N Z matrixes, N being a number of attention heads. Since this process generates multiple Z matrixes and the feedforward layers only receives a single matrix we concatenate every matrix Z into a single matrix.

2.3 WORD EMBEDDING

Word embedding is one of the most popular representations of text, capable of capturing syntactic and semantic relationships. Basically, an embedding is a dense vector of values that are trained to maximize the distance between words that usually appear in different context and minimize the distance between words that usually appear close-by. The length of the vectors that represent the words can be specified; high dimensional embeddings can learn more fine-grained relationships but are more data-hungry to train.

There are numerous proposed approaches to achieve this representation. The two most popular approaches are proposed by (Mikolov et al., 2013)

The continuous bag-of-words uses a window size in order to represent a target word, for example, given the phrase ”A cute cat looks through the windows” and use a window's size of two to predict the word “cat” the context words will be “cute” and “looks”.For instance, a target word wt is defined by the sum of the $w(t+n)$ words preceding wt and $w(t+n)$where n is a number arbitrarily set

The skip-gram model, this model finds word representations to predict the surrounding words. Formally, given a sequence of words $w_1, w_2, \ldots w_t$ the training objective of the skip-gram is to maximize the average log probability.

$$\frac{1}{T}\sum_{T=1}^{T} \sum_{-c\leq j\leq c, j\neq 0} log\ p(w_{t+j}|w_t)$$

c being the size of the training context, large c leads to higher accuracy but higher training times. However, since $p(w_{t+1}|w_t)$ is defined using softmax function, the computing cost of this formulation is proportional to the number of words in the vocabulary $O(n)$, rendering it impractical.

The author suggests two approaches to deal with this time complexity problem. The first approach is to use hierarchical softmax in place of the regular softmax. Hierarchical softmax uses a binary tree to represent the output words W as its leaves and for each node the probability of its child nodes. Each word can be reached by an appropriate path from the root of the tree, the probability of a word w is the product of the probabilities of each edge on the path to w. Since it is only needed to evaluate $log2(W)$ nodes the time complexity of this algorithm is $O(log\ n)$ speeding up the training time.

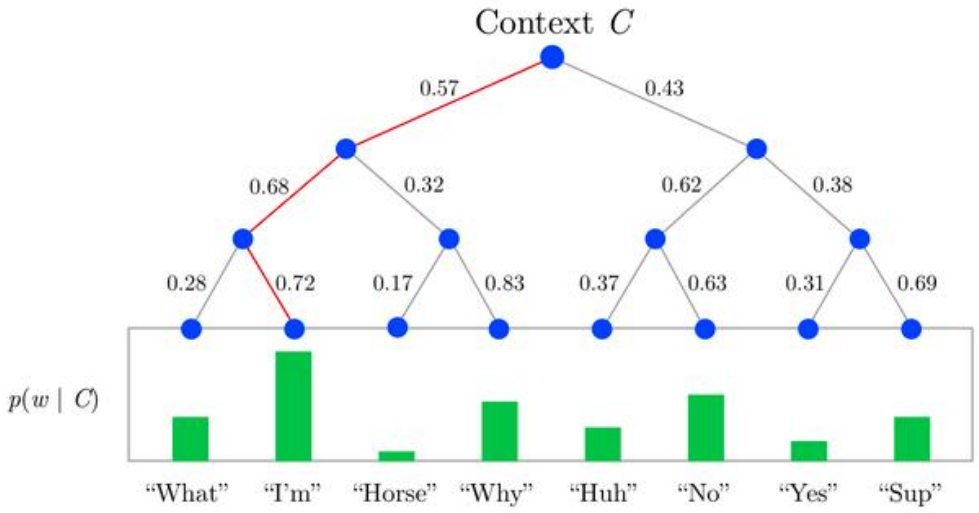

Figure 1 – Example of hierarchical softmax

The second proposed approach is called negative sampling. Negative Sampling uses a sigmoid function to learn for a given word-context pair (w, c) if the word c is in the context of the center word w. In context it is likely to observe pairs of w "animal" with context c {"dog","cat","zoo"}, this pairs will have output target of 1(Positive) while unlikely pairs, "animal" with {"plane","engine","car"} will have an output target of 0(negative). This approach turns a multi-classification problem into binary-classification reducing the time complexity.

2.4 MULTI-TASK LEARNING

Multi-task learning or MTL is a sub-field of Machine learning that aims to use the correlation between tasks to improve classification performance, it does so by forcing the model to learn generalized representations of features or attributes for various tasks. If there are N tasks that are related to each other MTL will use the knowledge contained in all the N tasks to improve its performance. Multi-task learning can be also be viewed as a way to introduce bias that prefers the hypothesis that explains all the N tasks, reducing the risk of overfitting. There are two major techniques to achieve MTL. The first

and most common is called hard parameter sharing. This technique shares hidden layers between all tasks while keeping task-specific output layers.

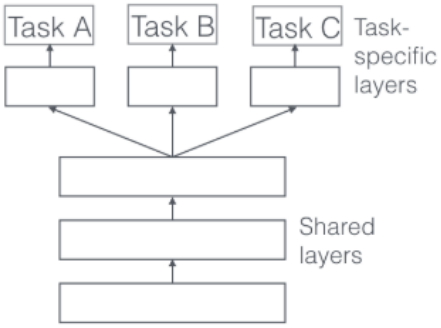

Figure 2 - Visual representation of hard parameter sharing. Taken from (Ruder, 2017)

The other major approach is called soft parameter sharing, each task has a separated model with its own weights and biases, these parameters are then regularized across the models in order to become similar and represent multiple tasks.

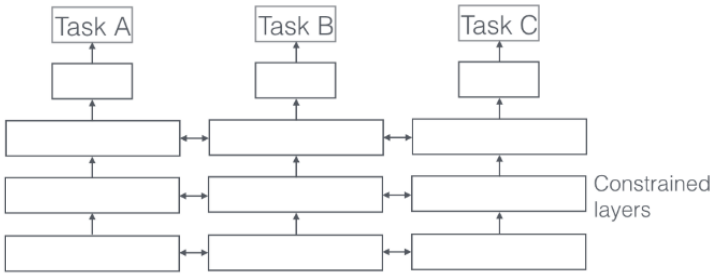

Figure 3 - Visual Representation of soft parameter sharing. Taken from (Ruder, 2017)

2.5 TRANSFER LEARNING

Transfer Learning is a method used in machine learning that consists in using a model that was pre-trained on a larger dataset, this allows the model to learn generic features, the model is then fine-tuned to perform on a smaller, domain-specific dataset on the desired task. This method allows for better predictive performance and faster training times.

3 RELATED WORK

In this chapter, we start by presenting different approaches that were previously used by some authors to tackle email classification tasks. As research in NLP continues to grow new models have appeared that are capable of achieving better results in a varied set of tasks, however to the best of our knowledge some of them are yet to be properly explored in email classification tasks, that is the case of RCNN, TextRNN, BERT, BERT Multilingual, RoBERTa, and XLM-R that are formally explained in this chapter.

3.1 EMAIL CLASSIFICATION

Email classification consists of classifying an email under one or multiple pre-defined categories, machine learning provides an efficient and automated solution to achieve this goal. Literature shows numerous ways to achieve automatic email classification, (Rennie, 2000) proposes a system bases on Naive Bayes or NB algorithm that suggests the three most suited folders for an incoming Email with an accuracy of 85%.

(Kiritchenko & Matwin, 2001) the authors compare support vector machines or SVM and NB on an email classification task, showing that SVM performs better than Naive Bayes.(Gent, 2007) the authors propose an automatic email classification system that discriminates customer's complains from non-complains using the Adaboost algorithm, the authors show that adding linguistic style features to the set of predictors increases the predictive performance of the algorithm.

(Yang & Linchi Kwok, 2012) Put K-means++, NB and K-nearest neighbors or KNN to test on an email classification task, the email corpus consisted of 3015 manually classified entries labeled according to 200 classes, of the three algorithms tested K-means++ showed the best performance by classifying the 1611 emails of the test set with an accuracy of 96%.

(Bonatti et al., 2016) with the objective of classifying a private email corpus of 11410 emails in 120 classes the authors test SVMs and NB algorithms alongside pre-processing techniques like part-of-speech (POS) and lemmatization. The authors show that for their case lemmatization caused the algorithms to perform worse, however, a POS filter that keeps verbs, nouns, adjectives, and adverbs improved the classification performance. The authors show that SVMs achieved the best performance with a precision of 87%.

(Borg, 2017) uses LSTMs to classify an email corpus of 33 topics the author experiments with various word representation methods such as Skipgram, Skipgram N-gram, Continues Bag of words, and global vectors of word representation or GloVe, the LSTM model is put to test using different network sizes and compared against non-sequential models like Support Vector Machine or SVM and Naive Bayes. The author concludes that LSTM achieves higher performance when compared with the non-sequential models although it requires more data and training time.

3.2 RECURRENT CONVOLUTIONAL NEURAL NETWORKS

(Lai et al., 2015) proposes a model that uses a bi-directional recurrent neural network to capture the contexts in order to obtain word meaning.

$$c_l(w_i) = f(W^{(l)}c_l(w_{i-1}) + W^{(sl)}e(w_{i-1}))$$

$$c_r(w_i) = f(W^{(r)}c_r(w_{i-1}) + W^{(sr)}e(w_{i+1}))$$

The first equation $c_l(w_i)$ is defined as the left context of the word w_i where $e(w_{i-1})$ represents the word embedding of the word (w_{i-1}). $W^{(l)}$ is a matrix that transforms the hidden layer in the next hidden layer. $W^{(sl)}$ is the matrix that combines the semantic of the current word with the next word's left context. f is a non-linear activation function.

The second equation $c_r(w_i)$ is defined as the right context of the word w_i and is calculated in the same manner as the first equation.

$c_i(w_i)$ and $c_r(w_i)$ are dense vectors with $|c|$real value elements and both $e(w_{i-1})$ and $e(w_{i+1})$ are both dense vectors with $|e|$ real values.

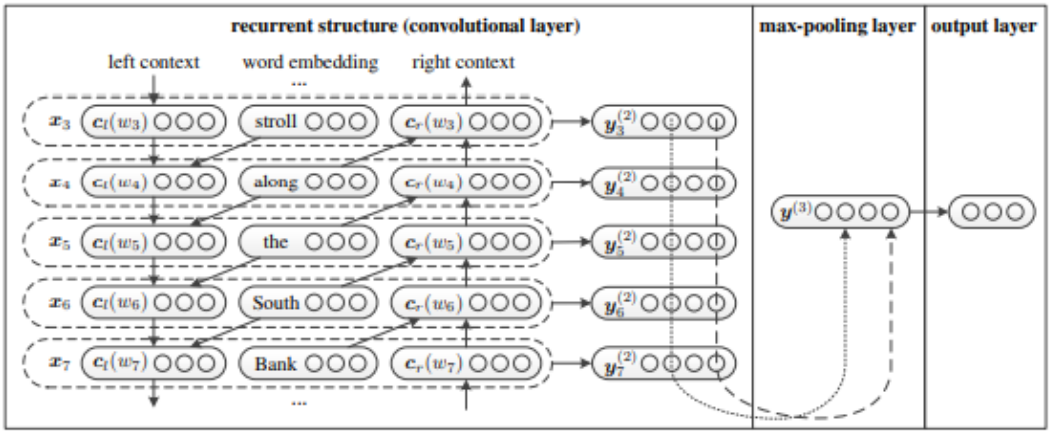

Figure 4 - Image representation of the RCNN layers. Taken from(Lai et al., 2015)

The third equation concatenates the left-side context with the embedding and the right-side context, this structure helps the model to disambiguate the meaning of the word w_i. c_r is obtained through a forward scan of the text while c_r is obtained through a backward scan.

$$xi = [c_i(w_i); e(w_i); cr(w_i)]$$

A linear transformation is applied to x_i together with a $tanh$ activation function and the results are sent to the next layer as shown in the equation above. $y_i^{(2)}$is a latent semantic vector used to determine the most useful factor to represent the text.

$$y_i^{(2)} = \tanh\left(W^{(2)}x_i + b^{(2)}\right)$$

In order to convert the text into a fixed length vector a max-pooling layer is utilized and represented on the equation above has $y^{(3)}$.

$$y^{(4)} = W^{(4)}y^{(3)} + b^{(4)}$$

In order to obtain the final probabilities, the softmax function is applied to $y^{(4)}$.

3.3 Recurrent Neural Network for Text Classification with Multi-Task Learning

In the section 2.4, we saw that multi-task learning can be used to further improve model performance, (P. Liu et al., 2016) proposes three different models of sharing information using recurrent neural networks.

In the first proposed model, uniform-layer architecture the multiple tasks share the same LSTM layer and embedding layer besides their own. For a given task m the input $x_t^{(m)}$ consists of two embeddings a task-specific $x_t^{(m)}$ and a shared word embedding $x_t^{(s)}$ concatenated

$$x_t^{(m)} = x_t^{(m)} \oplus x_t^{(s)}$$

The LSTM layer is shared for all tasks x_t, the final representation for a task m is the output of the LSTM at step t.

$$h_T^{(m)} = LSTM(x^{(m)})$$

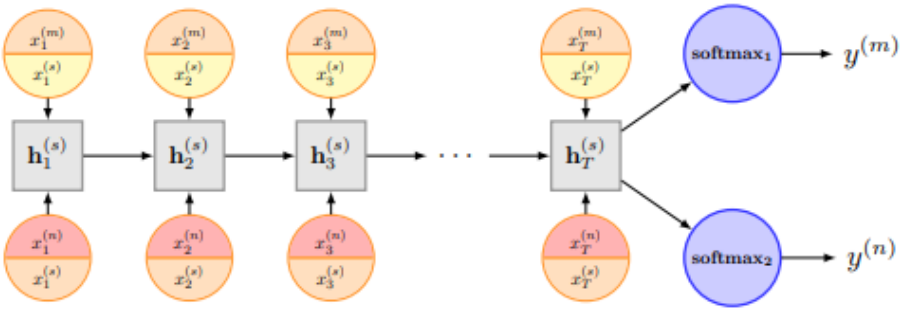

Figure 5 – Visual representation of the uniform-layer architecture. Taken from (P. Liu et al., 2016)

The second proposed model assigns an LSTM layer for each task that can share and retrieve information from the LSTM layers of the remaining tasks. Each pair of tasks (m, n) has its specific LSTM, at step t the output of two combined LSTM layers is $h_{(t)}^{(m)}$ and $h_{(t)}^{(n)}$.

To better control the information between tasks a global gating unit is implemented this gives the model the ability to decide how much information is accepted similar to the traditional LSTM model. Where $g^{(i\to m)} = \sigma(W_{(g)}^{(m)} x_t + h_{(g)}^{(i)} h_{(t-1)}^{(i)})$

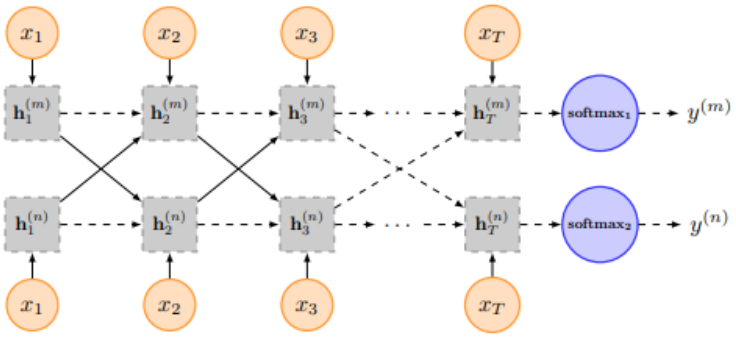

Figure 6 - Visual representation of the coupled layer architecture. Taken from (P. Liu et al., 2016)

The third and final model like the second model has an LSTM layer for each task on top of this the architecture introduces a bidirectional LSTM layer to share information through all tasks.

We denote the outputs of the forward and backward LSTMs at step t as $\overrightarrow{h}_{(t)}^{(s)}$ and $\overleftarrow{h}_{(t)}^{(s)}$ respectively. The output of the shared layer is $h_s^{(t)} = \overrightarrow{h}_{(t)}^{(s)} \; \overleftarrow{h}_{(t)}^{(s)}$.

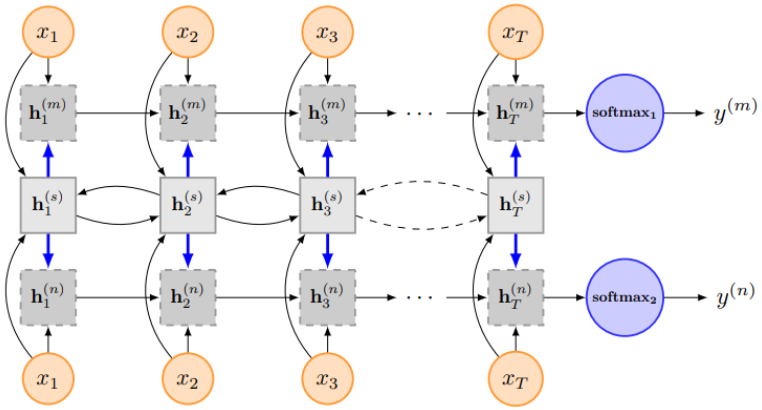

Figure 7 – Visual Representation of the Shared-layer Architecture. Taken from (Vaswani et al., 2017)

To improve the interaction between the task-specific layers and the shared layer this architecture also uses a gating mechanism to give the task-specific layer the ability to accept or refuse information from the shared layer.

3.4 BERT TRANSFORMER

Bidirectional Encoder Representation from Transformers or BERT proposed by (Devlin et al., 2019) is a transformer architecture designed to pre-train representations of unlabelled text.

The base model consists of 12 layers with hidden size of 768, 12 attention heads with a total of 340M parameters.

BERT model is pre-trained on BoooksCorpus that consists of 800M words and English Wikipedia Aricles (2,500M Words).

It is capable of handling various downstream tasks; to do so BERT makes use of an Input/Output representations that consists of a sequence of tokens, this sequences can be a sentence or a pair of sentences, the first token is the [CLS] that can be used for classification tasks since it is pre-trained to represent the entire sentencer. In order to discriminate pairs of sentences, a special token [SEP] is used. Each token is then represented using WordPiece Embeddings. BERT uses two more embeddings in order to represent the tokens, Segment embeddings that identify which sentence the token belongs to, and a Positional Embedding that identifies the position of the token in the inputted sequence.

For each token, the sum of the corresponding word embedding with the segment embedding and the positional embedding gives the representation of that token.

BERT is pre-trained on two different tasks, Mask language Modeling and Next sentence prediction.

3.4.1 Masked Language Modeling

MLM (Masked Language Modeling) this task aims to achieve contextualized embeddings. MLM consists of randomly masking a percentage(15%) of the inputted tokens and using the last hidden state of the model to predict the masked token, since the model makes use of the non-masked tokens in the sentence in order to make the prediction this enables BERT to learn bidirectional representations. Masked tokens only appear in the pre-training phase of BERT in order to mitigate this difference in the pre-training and fine-tuning BERT MLM is trained with the following strategy; 80% of the time the words are masked using the mask token [MASK], 10% of the time the token is replaced with a random word the other 10% of the time the word is not replaced at all. The prediction are made by feeding the masked tokens representation into a softmax function, the loss function used is cross-entropy.

(Wiedemann et al., 2019) proves that this approach can achieve the goal of contextualized embeddings. By simply clustering the BERT embeddings of a word in multiple contexts

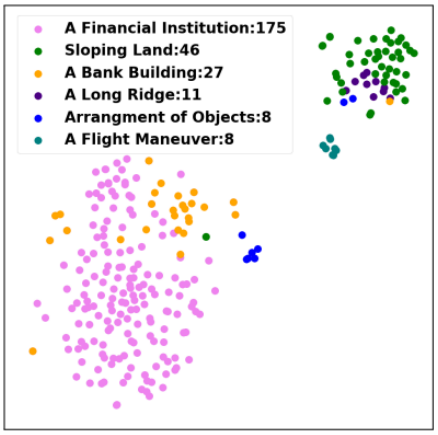

Figure 8 - Visualization of the clusters formed by the embeddings of the word Bank in different contexts. Taken from (Mickus et al., 2019)

We can clearly visualize different clusters that the embeddings form depending on the word context.

3.4.2 Next sentence prediction

The second task used to pre-train BERT is called next sentence prediction, this task allows the model to understand sentence relationships. Next sentence prediction is very similar to binary classification since it consists in training the model to predict if a give sentence A precedes a sentence B, in this specific application 50% of sentences are the following and they are labeled with (IsNext) the other 50% are random sentences and are labeled with (NotNext).

(Chan et al., n.d.) Finds that this task can negatively affect BERT embeddings. Next sentence prediction requires the tokens to be identified with a segment embedding as explained in section 3.4.1, this causes words with the same context being represented with different embeddings.

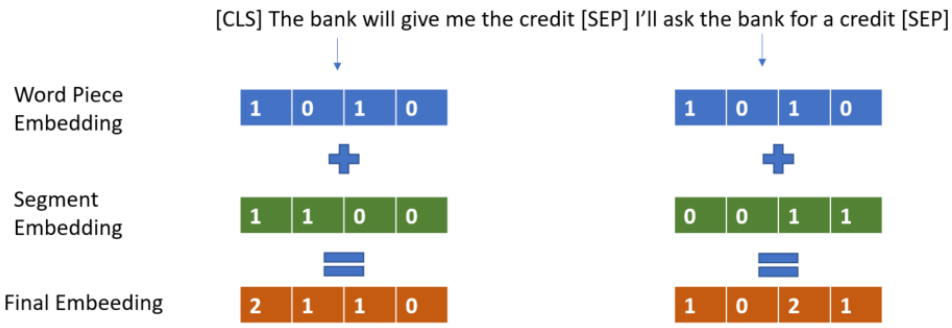

Figure 9 – Simplified example of how segment embeddings influence the final embeddings

In the example above the word, bank is used in the same context however because they are in a different pair due to the segment embedding the final embedding of the word diverges introducing noise in the contextualization of the word.

3.4.3 German BERT

German BERT is a model proposed by (Y. Liu et al., 2019) this model has the same architecture and training tasks as vanilla BERT. As the name indicates German BERT is a transformer model trained on Deutsch datasets like German Wikipedia OpenLegalData Dump and German news articles.

3.5 RoBERTa

RoBERTa or Robustly optimized BERT is a transformer model proposed by (Martin et al., 2019) RoBERTa is based of BERT architecture, the model archives state of art results on GLUE, RACE and SQuAD tasks. RoBERTa researchers found that BERT is undertrained and proposes an improved method to train BERT model. These changes to the training method include training more epochs with bigger batches and bigger Datasets, training on longer sequences of sentences, removing the next sentence prediction task that has shown in section 3.4.2, can degrade the quality of the contextualized embeddings. The masked language modeling task was also changed, instead of masking the words during the pre-process phase (static masking) the masks are generated dynamically every time a sequence is fed to the model, this way the model sees different masks over the various training epochs.

3.5.1 CamemBERT

CamemBERT is a model proposed by (Lample & Conneau, 2019) is a transformer model trained on French web crawled data, this model is based on RoBERTa architecture and training tasks

3.6 BERT Multilingual

BERT multilingual or MBert was released by (Devlin et al., 2019) this model has the same architecture as the original BERT, instead of being pre-trained only using English corpora, multilingual Bert is pre-trained on monolingual corpora of 104 languages from Wikipedia pages. Even though this model is capable of multilingual representation has (Pires et al., 2020) explains MBert does not incorporate any explicit multilingual training objective this hinders the performance of the model especially across languages that use different scripts, despite that the authors debate that the MBert capability to generalize across languages comes from the word piece tokenizer that causes words with similar meaning in different languages to be mapped to the same shared space.

XLM is a cross-lingual model proposed by (Conneau et al., 2019) the architecture of this transformer is the same as BERT, the differences are in the tokenizer, pre-training strategies and corpora used.

The XLM model is trained in a MLM task similar to BERT however instead of using pairs of sentences for pre-training has described in BERT section, XLM uses text streams of an arbitrary number of sentences, this eliminates the need of segment embeddings, language embeddings that identify the text stream language are used instead, furthermore to compensate for the imbalance between low-resource languages and high-resource languages most frequent tokens are subsampled.

There is an alternative training task for XLM called Translation Language Modelling this task is an extension of MLM, instead of using monolingual text steam, TLM uses pairs of sentences A and B,

sentence B being the translation of sentence A. This encourages cross-lingual embedding alignment since in order to predict the masked token the model attends to both languages.

This model makes use of a shared vocabulary across all languages created by using BPE (byte pair encoding). This method improves embedding alignment across all languages that share the same vocabulary or digits

3.7 **XLM-R**

XLM-R is a model proposed by (L. Liu et al., 2019), it is pre-trained on 100 different languages of CommonCrawl data since this corpus contains more data for low-resource languages, this allows the model to learn overall better representations for these languages.

XLM-R is trained using a dynamic MLM task very similar to the one explained on the XLM section without making use of language embeddings, this allows XLM-R to have a better performance when it comes to code-switching.

4 METHODOLOGY

In this chapter, we present the methodology used to tackle the multi language classification problem. We start by collecting data relevant to the task, the dataset used throughout the experiments was provided by Clevery in sections 4.1 we present the corpora used as well as the method used to augment it.

The dataset is pre-processed by replacing names, emails, and URLs with specific tokens. Different methods of tokenization are used, for TextRNN and RCNN models we use the tokenizer provided by the NLTK library for the remaining models we use the specific tokenizers described in the previous chapter.

Various methods of encoding are tested, for the textRNN and RCNN models we start with randomly initialized embedding, this is before the model starts training the embedding for each word is randomly initialized and trained along with the classifier. According to (Kocmi & Bojar, 2017) pre-trained embedding can boost the model's performance, Inspired by the author we also train Word2Vec Embedding described in section 2.3 using the gensim library on the English train Dataset.

Various machine learning models are tested for this task, TextRNN and RCNN are implemented using the Neural Classifier framework presented in section 4.2, RoBERTa, CamemBERT, BERT, GermanBERT, BERT Multilingual, and XLM-R are implemented using the transformers library described in section 4.3, all the implemented models are formally explained in chapter 3.

Hyperparameters play a crucial role in the training and fine-tuning of a given model having a high impact on the final performance. These are sets of variables that determine the neural network structure and how it's trained, they are usually fixed and unchangeable during the training phase, unlike parameters, hyper-parameters cannot be learned from data. The best value for each hyperparameter entirely depends on the problem we are trying to solve and the model we are working with. Hyperparameter tuning is performed against a metric, this helps us to understand which set of hyperparameters works best for the task that we are trying to achieve. There are various ways to do hyperparameter tuning; Grid search where a model is built for every combination of pre-specified hyperparameters, Randomized search where the hyperparameters are randomized N times, N being a number arbitrarily set, and a model is built for each random combination, for this dissertation, we will perform hyperparameter tuning manually, this way we can better understand the effect of each hyperparameter on the final model and adjust it in to achieve the best performance possible. The evaluation metrics used to access the model performance are presented in section 4.5

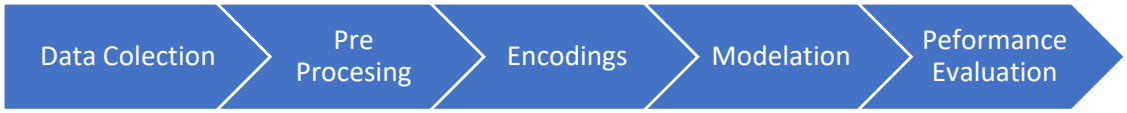

Figure 10 – Methodology order

4.1 CORPORA USED

The corpora used consists of 5926 in-house customer support emails with the average of 250 tokens written in English, 2990 emails written in Deutch, both annotated, and 66 912 non-annotated English written emails. Both the annotated English dataset and the Deutsch dataset were previously cleaned, removing names, replacing emails, URLs with specific tokens. The annotated datasets were previously partitioned in train and validation sets. In order to create additional datasets in multiple languages, we used marianMT transformer to translate the original English dataset into French and Spanish. The original English dataset was also augmented by translating the original Deutsch dataset to English, all the validation sets for the non-English languages are translations of the English validations dataset. After the machine translation step, we ended up with 8318 English Emails, 5338 Spanish Emails, 5338 French Emails, and 3508 Deutsch emails.

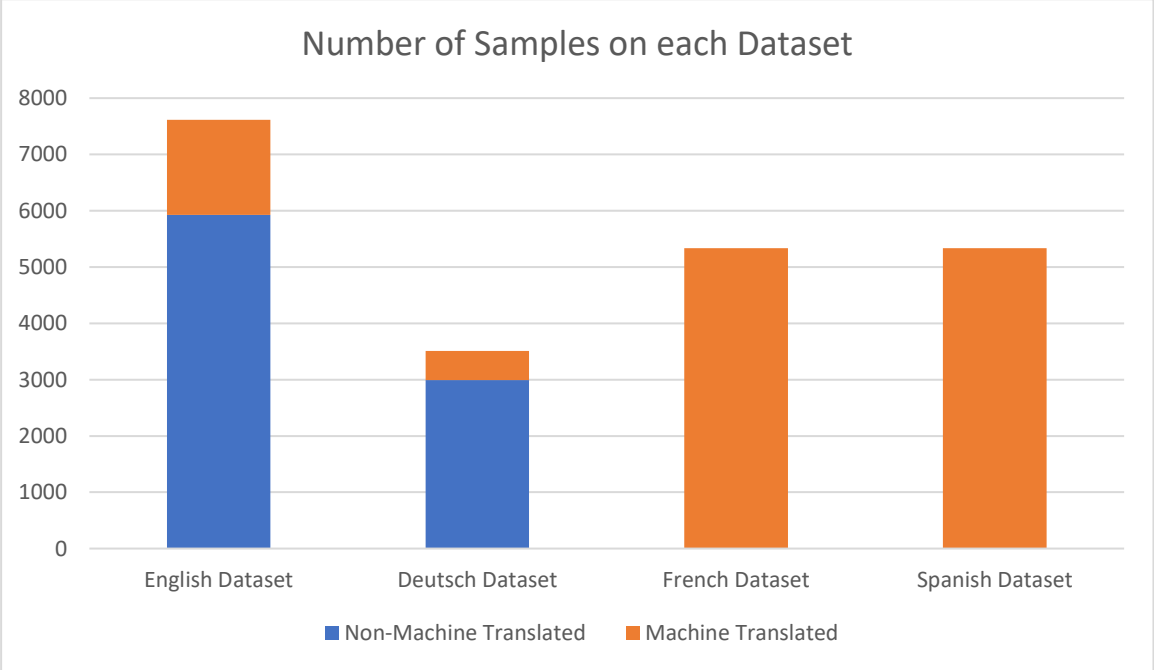

Figure 11 – Number of samples on each dataset

4.2 NEURAL CLASSIFIERS

We start by testing the models with the best performance according to NeuralClassifier toolkit (L. Liu et al., 2019), Neural Classifier is an open-source text classification toolkit designed to solve hierarchical and multi-label problems using a modular framework.

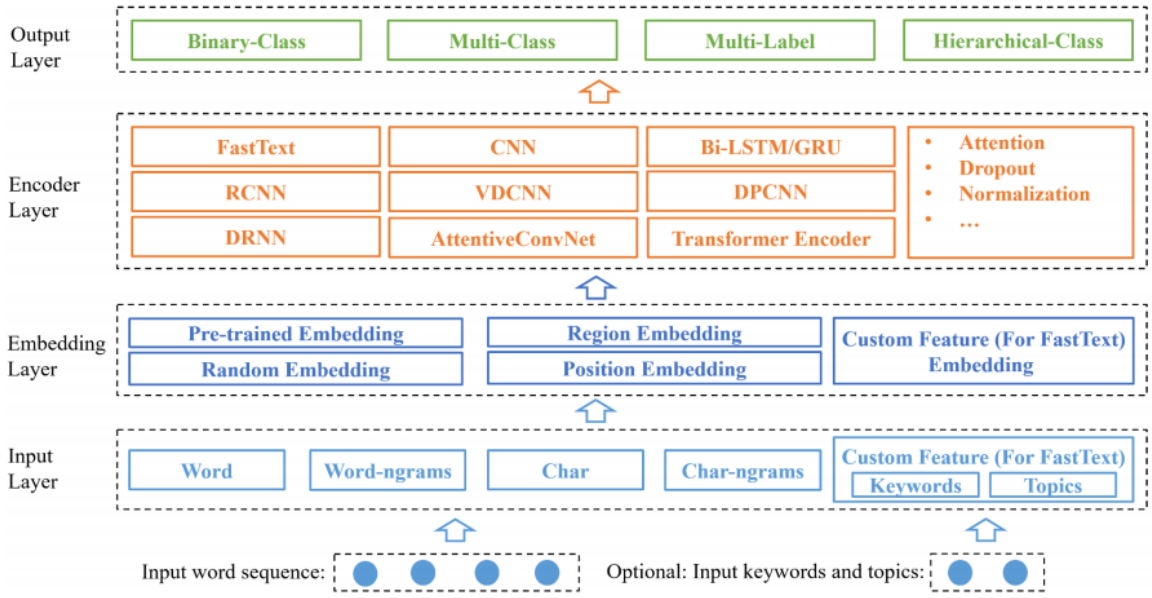

Figure 12 – NeuralNetwork toolkit framework. Taken from (Devlin et al., 2019)

The toolkit uses a four layer architecture that can be configured via JSON file. The JSON configuration file acts as a user interface and is separated into four parts;

Common Settings where the classification task can be specified whether if its single-label classification, multi-label classification, hierarchical or flat.

Input settings provide control over input data such has max sequence length, dictionary size, and pre-trained embeddings.

Training settings include features like batch size, type of loss function, optimizer, learning rate number of epochs, and CPU or GPU training.

Network structure settings is used to specify the pretended text encoder, for each text encoder the corresponding hyperparameters can be tuned here.

4.3 TRANSFORMERS LIBRARY

The Transformers library is a large open-source community supported by Hugging Face company (Wolf et al., 2019). This library largely developed in python provides an API to use various state of the art pre-trained models. The main focus is to provide easy and compressible access to pre-trained transformers models for researchers and companies as well as pushing good implementation practices. The library uses three main classes. The configuration class defines the architecture optimization of the model, the tokenizer class provides the tokenizer used in each model as well as configurations that can be modified by the user, the model class provides the model logic.

4.4 HYPERPARAMETERS

4.4.1 Batch size and Epochs.

The batch size defines the number of training data samples that will pass through the network and is highly limited to the amount of RAM available in the machine when a CPU is used for training, if a GPU is used for training the batch size will be limited by the amount of VRAM available in the GPU. Epochs are the number of times the entire dataset will pass through the model, the number of steps an epoch needs to perform to be considered complete is formally explained by the following formula

$$Steps = \frac{train\ size}{batch\ size}$$

4.4.2 Learning Rate

The Learning rate is a hyperparameter that affects how much the model's weights are adjusted during the training phase, the learning rate value can dramatically change the model performance and the optimal value can differ a lot depending on the model architecture. By default, low learning rates are better at finding local minima but can lead to massive increases in training time or in some cases completely stall the model of any progress. However, high learning rates can make the model unstable and prevent it from converging.

Learning rates can be adjusted while the model is training by setting up a scheduler that makes the learning rate decay accordingly to a linear, exponential, or cosine function

4.4.3 Dropout

Dropout is a regularization technique used to avoid overfitting, therefore improving the generalization capacity of the NN. It consists of ignoring a certain set of neurons during the training phase excluding them from being considered during the forward and backward pass phases.

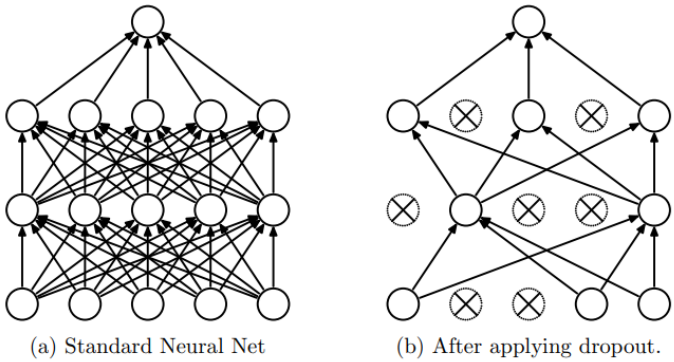

Figure 13- Visual representation of a NN without using dropout (a) and using dropout (b). Taken from (Galeone, 2017)

4.4.4 Activation function

The activation functions are equations that are bound to each neuron of the NN, they determine if a specific neuron is relevant to the prediction and normalize the output of each neuron to a range between 1 and 0 or -1 and 1 depending on the function used. The activation function can greatly impact not only the quality of the predictions but also the training time of the model.

Sigmoid Function – Commonly used in classification, transforms values into probabilities making the output values range between 1 and 0, formally explained in the equation below.

$$y = \frac{1}{1 + e^{-x}}$$

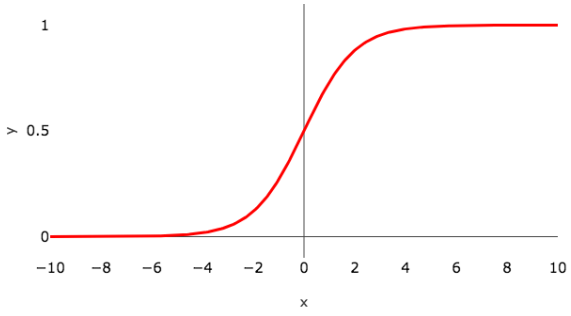

Figure 14 - Sigmoid function plot

Tanh Function – Also referred to as the hyperbolic tangent makes the normalizes the values to range between 1 and -1, formally explained in the equation below

$$y = \frac{2}{1 + e^{-2x}} - 1$$

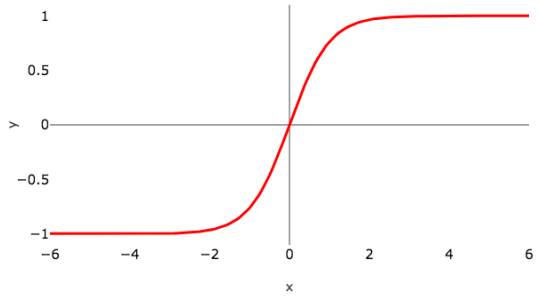

Figure 15 - Tanh function plot

Rectified Linear Unit function – Or simply abbreviated to RELU is the most commonly used function in deep learning due to the ability to solve the vanishing gradient problem (gradient decreasing exponentially halting the train of the model). In this function, negative input values are set to 0, positive input values are mapped to it's owned value, formally explained in the equation below

$$y = \max(0, x)$$

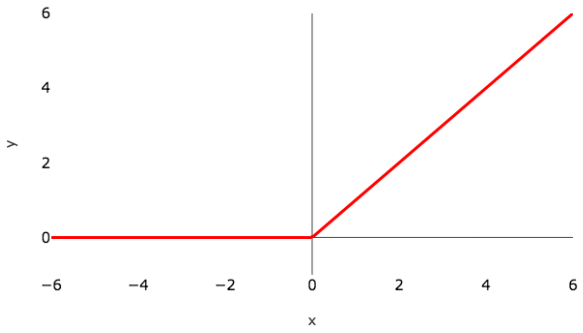

Figure 16 - RELU function plot

4.5 Evaluation Metrics

Evaluation metrics are used to determine how well a given model is performing and allows us to have feedback on the changes we do to hyperparameters. The choice of the metric is a crucial part of building a classifier and one that is entirely dependent on the goal we want to achieve.

Accuracy metric is the ratio between the number of correctly predicted labels related to the total of input samples. This metric works well only if there is an equal number of samples in each class. Let us take a binary classification example. If a given dataset is composed of 90 samples of a class X and 10 samples of a call Z, we can easily achieve 90% accuracy by simply labeling all the samples with class X

$$Accuracy = \frac{\text{tp} + \text{tn}}{\text{tp} + \text{fn} + \text{fp} + \text{tn}}$$

In the case of multi-classification, we can only calculate the Average Accuracy which gives us the average per-class effectiveness of a classifier, formally explained by the following equation

$$Average\ Accuracy = \frac{\sum_{i=1}^{l} \quad \frac{\text{tp}_i + \text{tn}_i}{\text{tp}_i + \text{fn}_i + \text{fp}_i + \text{tn}_i}}{l}$$

Precision metric tells us, of all the predicted labels for a given class, the ratio of those entries that were correctly predicted, in the binary classification case precision can be explained by the following equation

$$Precision = \frac{\text{tp}}{\text{tp} + \text{fp}}$$

Recall metric tells us, for all entries that should have been classified with a given class, the ratio that was correctly classified. For the binary classification, case recall is defined as follows

$$Recall = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

In the case of multi-classification problem, just like precision, recall calculated differently and for the purpose, we will use weighted recall, formally explained in the equation bellow

F1-Score Is the harmonic mean of precision and recall, F1-score is often used to seek a balance between precision and recall, in binary classification F1-Score is defined as follows

$$F1 - Score = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

In a multi-classification problem, precision recall and f1-score are calculated differently, for the purpose of this thesis we will use the weighted variation of all the metrics, that are formally explained in the equation below.

$$\frac{1}{\sum_{l \in L} |\hat{y}_l|} \sum_{l \in L} |\hat{y}_l| \varphi\,(\hat{y}\,, \hat{y}_l)$$

Where l is a set of labels, $\hat{y}$ a ground truth label, y a predicted label, $\hat{y}_l$ is all the ground truth labels that have the label l, $|\hat{y}_l|$ the number of ground truth labels that have the label l, $\varphi(\hat{y}\,, \hat{y}_l)$ computes the precision, recall, and f1-score for the ground truth and predicted labels that have the label l.

To calculate precision let $\varphi(A,B) = \frac{|A \cap B|}{|A|}$, to calculate recall let $\varphi(A,B) = \frac{|A \cap B|}{|B|}$, and to calculate f1-score let $\varphi(A,B) := (1+\beta^2)\frac{Precision(A,B) * Recall(A,B)|}{\beta^2 Precision(A,B) + Recall(A,B)}$

5 EXPERIMENTS

In this chapter, we present the experiments conducted throughout this dissertation as well as the results obtained. In section 5.9 we discuss the obtained results and some of the limitations met.

5.1 TRAINING TEXTRNN

We train a TextRNN explained in section 3.3 using the original English Dataset described in section 4.1 Inspired by (Goodfellow et al., 2016) we start with a high learning rate of 0.1 and then try with exponentially lower values, similar to a log scale from 0.1 to 1e-5 after we found the value that yields good results we further test with smaller increases until we reach an optimal value, in this case, we found the optimal value for the learning rate at 1e-2. To further improve the model, we initialize the word embeddings with the pre-trained word2vec embeddings. Since we are no longer initializing the embeddings randomly, freezing the embedding layer on the first epochs of the training phase can often prove beneficial, not doing so can cause the quality of the embeddings to decay.

Model	Precision	Macro F-score
TextRNN	68%	59%
TextRNN + w2v Embbedings	71%	66%
TextRNN + w2v Embbedings + Freeze	72%	64%

Table 1 – Weighted Precision and Macro F-score for TextRNN, TextRNN + w2v embeddings and TextRNN + w2v Embbedings embedding freeze

by comparing the models, we can clearly see a boost in the model performance when the pre-trained embeddings are introduced.

5.2 RCNN

The implementation of the RCNN explained in section 3.2 is made using the NeuralClassifier Toolkit described in section 4.2. RCNN training is made on the same dataset as the previous model, using the same strategy to find the optimal learning rate. The toolkit provides interface to edit 7 settings for the RCNN, RNN type, hidden dimensions size, number of layers and bidirectionality were explained in the previous chapter, kernel sizes, max pooling, num kernals.

Model	Precision	Macro F-score
TextRCNN	69%	65%
TextRCNN + w2v Embbedings	72%	65%
TextRCNN + w2v Embbedings + Freeze	73%	69%

Table 2 - Wheighted Precision and Macro F-score for TextRCNN, TextRCNN + w2v embbedings and TextRCNN + w2v Embbedings embbeding freeze

5.3 Fine-Tuning Bert

We start by fine-tuning the BERT-base model previously explained in section 3.4. on the original dataset described in section 4.1. With the recommended learning rate between 2e-5 and 5e-5 in mind (Devlin et al., 2019) the objective of the first test is to find the best learning rate for this classification problem. Four tests were ran using learning rate of 2e-5, 3e-5, 4e-5 and 5e-5.

Learning Rate	Precision	Macro F-score
5e-5	75%	64%
4e-5	74%	60%
3e-5	72%	60%
2e-5	67%	50%

Table 3 – Bert weighted precision and macro F-Score variation across various learning rates

With this experiment, we conclude that higher learning rates work better to fine-tune the BERT base model to our specific domain. This conclusion contradicts the results from (Sun et al., 2019), where

BERT-base model would incur in a catastrophic forgetting problem, this is, a higher learning rate would make the model erase knowledge acquired in the pre-training phase, however, the authors used a much larger dataset that can intensify this unintended behavior.

As we augmented the original English dataset with translated text from the Dutch dataset, it was important to test if Bert would benefit from more data in the fine-tuning phase. In order to validate this premise, we fine-tune Bert using the same hyper-parameters from above with learning rate of 5e-5 variating the amount of data in the train set, since we are adding data to the train set in order to not incur in under fit problem the training epochs are set to 20. We ran 4 tests each test adds 450 random entries to the English train Dataset.

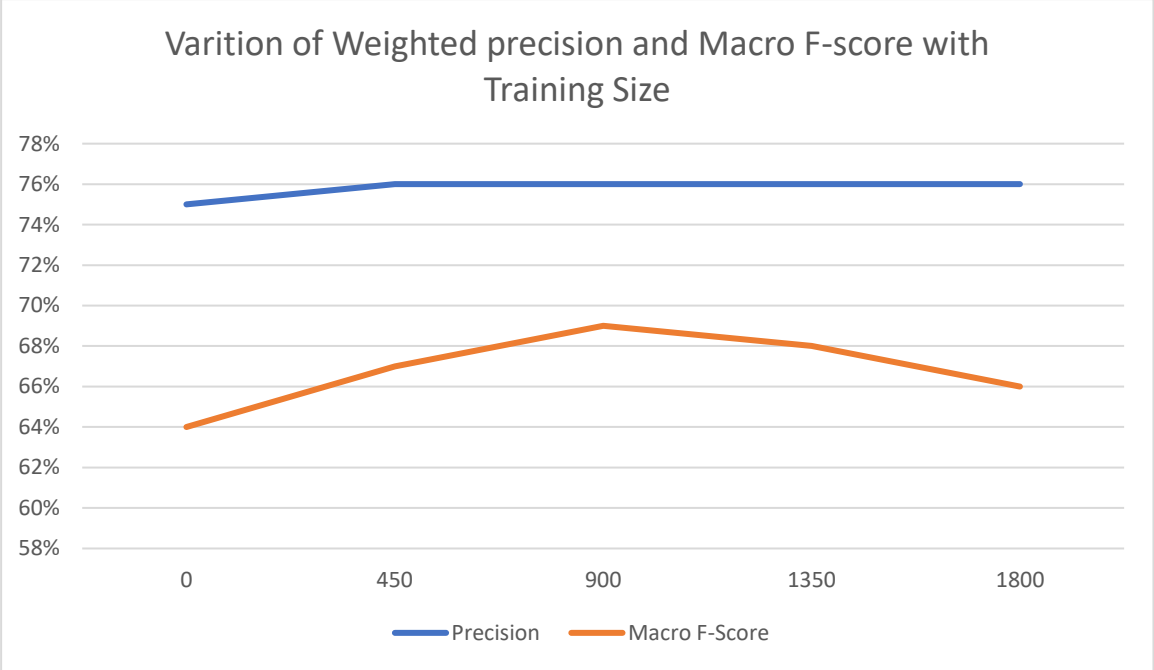

Figure 17 - Variation of Weighted precision and Macro F-score with Training Size

With this test we can conclude that although adding more data seems to slightly improve the classifier performance when compared to the first test, further adding data to the fine-tuning can make the performance decay. Further hyper-parameter tuning was performed but the results always led to the same conclusion, do to this problem we decided to move away from BERT.

5.4 FINE-TUNING ROBERTA

With the goal of improving the performance of the monolingual classifier we decided to test RoBERTa model, as explained in section 3.5, this model has better benchmark performance than BERT due to the larger amount of data used in the pre-training and the modification made to the pre-training tasks. We fine-tune RoBERTa base model on the augmented English train dataset. The objective is to assess if Roberta is capable of taking advantage of a larger dataset.

Model	Precision	Macro F-score
RoBERTa	86%	67%
BERT	76%	68%

Table 4- Bert and Roberta weighted precision and Macro F-Score

With this test we conclude that RoBERTa has much better performance than the previously tested model, making a better baseline for the final multilingual model.

To assure that we get the best results possible for the baseline further pre-training is applied to the model. As explained in section 3.5, Roberta is pre-trained in a general domain corpus, the data distribution of this corpus may differ from our domain-specific corpus. Inspired by (Sun et al., 2019) we further pre-train RoBERTa using masked language modeling tasks, the best epoch is chosen based on loss.

To achieve the In-domain pre-training previously explained, we start by running a masked language modeling task on a non-finetuned RoBERTa model using the train split of the augmented English dataset. After the task is complete we fine-tune RoBERTa with the same hyper-parameters used in the previous test.

Model	Precision	Macro F-score
RoBERTa	86%	67%
In-domain pre-trained RoBERTa	88%	71%
In-domain pre-trained RoBERTa (non-annotated English Dataset)	88%	72%

Table 5 - RoBERTa weighted precision and Macro F-Score with and without In-domain pre-training

As described in the table 5 the pre-training strategy results in a performance increase of 2% in precision and 4% in Macro F-Score. To further improve the performance, we try to the same strategy using the non-annotated English Dataset described in section 4.1 although we can improve the classifier by 1% in Macro F-Score the fine-tuning takes more time to converge.

5.5 Fine Tuning Multi-Lingual Bert

Multi-Lingual Bert is our first attempt to train a multilingual classifier, taking a zero-shot approach for the first test, we fine-tune multi-lingual Bert on the English augmented Dataset and evaluate on the English, Deutsch, French, and Spanish evaluation set to see how well the model generalizes across languages. On the second test, we concatenate all the training datasets and train them together

evaluating with the same method. On both tests, we chose the model that has better performance across the various languages.

Model	EN	DE	FR	ES
MBert Zero-Shot	72%	56%	46%	42%
Mbert	78%	74%	74%	75%

Table 6 – Comparison of MBert weighted precision on English, Deutch, French and Spanish test sets, using zero-shot approach and training on all languages

5.6 XLM, CAMEMBERT AND GERMANBERT

We fine-tune XLM by trining in on the concatenation of the English training set and Deutsch training set. We also experiment with CamemBERT and GermanBERT by training in on the French training set and Deutsch training set. We decided to not further explore these models since they produce worse results when compared with the previous models.

Model	EN	DE	FR
CamemBERT	-	-	70%
German BERT	-	69%	-
XLM	74%	65%	-

Table 7 – CamemBERT, German BERT, and XLM weighted precision

5.7 FINE TUNING XLM- ROBERTA

We fine-tune XLM- RoBERTa using three different strategies. In the first strategy, we only use monolingual training sets for each language, resulting in a different fine-tuned model for each language. The second strategy uses the concatenation of all monolingual language sets in a single multilingual training set. In the third strategy, we use the concatenated set to pre-train and then fine-tune the model. The in-domain pre-training is done as described in section 5.4.

Model	EN	DE	FR	ES
XLM-R English	78%	-	-	-
XLM-R Deutsch	-	75%	-	-
XLM-R French	-	-	75%	-
XLM-R Spanish	-	-	-	75%
XLM-R All Languages	82%	78%	80%	80%
XLM-R Zero-Shot	78%	67%	68%	71%
In-domain pre-trained XLM-R	84%	82%	82%	82%

Table 8 – Comparison of XLM-R weighted precision trained on mono language and multi-language training sets, zero-shot and in-domain pre-training

5.8 TIME PERFORMANCE

Training NLP models can come with a hefty time cost that can translate into monetary costs, not having the right hardware to train more complex models like transformers can halt the process of hyperparameter tuning or make the training of the model impossible do to memory problems, therefore, it's important to take in consideration the hardware available when selecting a model to train. The time a given model takes to classify unseen data can also be important, especially if real-time results are needed. Has described in the table 8 the increase in classification performance that transformers provide come at cost of training times and classification times.

Model	Train time 1000 steps	Time for Classificating 1 label
TextRCNN	2min 12sec	7e-5 secs
RNN	1min 48sec	6e-5 secs
BERT	14 min	0,2 sec
RoBERTa	14min	0,2 sec
CameBERT	14mins	0,2 sec
German Bert	14mins	0,2 sec
XLM	19 mins	0,2 sec
XLM-R	15 Mins	0,3 Secs

Table 9 – Comparasion of training times and classification of one email for each tested model ran on Tesla K80

All the tests carried trought this dissertation were realized using a machine running ubuntu with a Tesla K80 GPU, 64GiB RAM, and 250 terabytes of disk memory, costume code was executed on jupyter lab.

5.9 DISCUSSION AND LIMITATIONS

Our results suggest that XLM- RoBERTa combined with in domain pre-training is the best approach for our domain-specific case, fine-tuning the model with a training set where all the target languages are represented, has proven to be the best approach to achieve a multi-language classifier that has uniform results across all the evaluated languages. However, in the English case, XLM- RoBERTa was never able to outperform monolingual transformers like RoBERTa and BERT, achieving only better results than non-transformer models like RCNN and TextRNN. The zero-shot approach has proven to be less precise however it eliminates the need to perform machine translation which is a resource-intensive step and can be unreliable for larger datasets.

When it comes to English only classifiers, Roberta combined with in-domain pre-training achieves the best results, our experiments suggest that using machine translation to augment the training dataset allows for better performance.TextRNN and RNN cannot keep up with the transformer's classification performance. however, these models are far simpler to train and less resource-intensive making them a good option if the hardware is a limitation.

While our approach is capable of outputting performing classifiers both for English only and multi-language, hardware limitations prevent us to test on larger transformers such has Roberta-Large, Bert-Large and XLM-Roberta-Large, the literature suggests that these models can outperform the models that we use.

Machine-translation is used to generate multi-language to compensate for the lack of human-generated entries, however, this translation can be inaccurate and can affect the fine-tune and evaluation of the models.

6 CONCLUSIONS

Has shown in section 5.9 the goal of training a multi-language classifier capable of classifying customer support emails has been successfully achieved, throughout the dissertation we were also able to answer the following initial questions.

- **How well can transformers perform on customer support data classification?**

From our experiments throughout this thesis, we can conclude that transformer architecture performs exceptionally well for monolingual and multi-lingual classifications although difficult to train from scratch do to the amount of data and computational resources needed, transfer learning allows us to achieve viable results by using pre-trained versions of the models.

- **What is the performance trade-off between monolingual and multi-lingual models?**

The use of various languages in the training set using a multi-lingual model helps to leverage the results across all languages achieving better results than training a monolingual classifier for each language, however, the multi-linguage models are more computational and time expensive to train. TextRNN and RCNN although less accurate than transformers are capable of outputting competent results for monolingual classification requiring less computational resources. For both models, pre-training embeddings seem to be a crucial step to achieve viable results.

- **Is it possible to build a performing multi-language classifier by training it on machine-translated data?**

Machine-translation has proved viable to produce datasets of various languages but the quality of the translations can drastically affect the final performance of the classifier.

7 FUTURE WORK

Future work is aligned with the limitation described in the previous section, testing the “large” variant of the models used can improve the results of the final classifier. All the multi-language models should be fined-tuned and re-evaluated using human-generated data to have a better understanding if machine-translation is affecting the model performance.

There are other variants of the models used that can be interesting to explore, DestilBERT and DestilRoBERTa are less resource-intensive variants that can retain almost 90% of the original performance of BERT and RoBERTa, this can lead to faster fine-tuning times and faster classifications.

8 BIBLIOGRAPHY

Bonatti, R., de Paula, A. G., Lamarca, V. S., & Cozman, F. G. (2016). Effect of part-of-speech and lemmatization filtering in email classification for automatic reply. *AAAI Workshop - Technical Report*, *WS-16-01-*, 496–501.

Borg, A. (2017). *DEGREE PROJECT FOR MASTER OF SCIENCE IN ENGINEERING Email Classification with Machine Learning and Word Embeddings for Improved Customer Support*.

Chan, B., Möller, T., Pietsch, M., Soni, T., & Yeung, C. M. (n.d.). *Open Sourcing German BERT*. https://deepset.ai/german-bert

Coffin, K., & New, C. (2001). Customer support and new product development an exploratory study. *International Journal of Operations and Production Management*, *21*(3), 275–301. https://doi.org/10.1108/01443570110364605

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., & Stoyanov, V. (2019). *Unsupervised Cross-lingual Representation Learning at Scale*. http://arxiv.org/abs/1911.02116

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, *1*(Mlm), 4171–4186.

Dovaliene, A., Gadeikiene, A., & Piligrimiene, Z. (2007). Customer Satisfaction and its Importance for Long-Term Relationships with Service Provider: the Case of Odontology Services. *Inzinerine Ekonomika-Engineering Economics*, *5*(5), 59–67.

Galeone, P. (2017). *Analysis of Dropout*. https://pgaleone.eu/deep-learning/regularization/2017/01/10/anaysis-of-dropout/

Gent, B.-. (2007). *Improving Customer Complaint Management by Automatic Email*. 1–35.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1–15.

Kiritchenko, S., & Matwin, S. (2001). Email Classification with Co-Training. *Proceedings of the 2001 Conference of the Centre for Advanced Studies on Collaborative Research*, 301–312. http://portal.acm.org/citation.cfm?id=782096.782104

Kocmi, T., & Bojar, O. (2017). *An Exploration of Word Embedding Initialization in Deep-Learning Tasks*. http://arxiv.org/abs/1711.09160

Lai, S., Xu, L., Liu, K., & Zhao, J. (2015). Recurrent convolutional neural networks for text classification. *Proceedings of the National Conference on Artificial Intelligence*, *3*, 2267–2273.

Lample, G., & Conneau, A. (2019). *Cross-lingual Language Model Pretraining*. http://arxiv.org/abs/1901.07291

Liu, L., Mu, F., Li, P., Mu, X., Tang, J., Ai, X., Fu, R., Wang, L., & Zhou, X. (2019). Neural classifier: An Open-source neural hierarchical multi-label text classification toolkit. *ACL 2019 - 57th Annual Meeting of the Association for Computational Linguistics, Proceedings of System Demonstrations*, 87–92. https://doi.org/10.18653/v1/p19-3015

Liu, P., Qiu, X., & Xuanjing, H. (2016). Recurrent neural network for text classification with multi-task learning. *IJCAI International Joint Conference on Artificial Intelligence*, *2016-Janua*, 2873–2879.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. *1*. http://arxiv.org/abs/1907.11692

Loomba, A. P. S. (1998). Product distribution and service support strategy linkages: An empirical validation. *International Journal of Physical Distribution & Logistics Management*, *28*(2), 143–161. https://doi.org/10.1108/09600039810221694

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. *7th International Conference on Learning Representations, ICLR 2019*.

Martin, L., Muller, B., Suárez, P. J. O., Dupont, Y., Romary, L., de la Clergerie, É. V., Seddah, D., & Sagot, B. (2019). *CamemBERT: a Tasty French Language Model*. http://arxiv.org/abs/1911.03894

Mickus, T., Paperno, D., Constant, M., & van Deemter, K. (2019). *What do you mean, BERT? Assessing BERT as a Distributional Semantics Model*. https://doi.org/10.7275/t778-ja71

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, 1–12.

Mitchell, T. M. (1997). *Machine Learning*.

Pires, T., Schlinger, E., & Garrette, D. (2020). How multilingual is multilingual BERT? *ACL 2019 - 57th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, 4996–5001. https://doi.org/10.18653/v1/p19-1493

Qasim, M., & Asadullah, M. (2012). *The Role of Customer Support Service in Relationship Strengthening A Case of Swedish Broadband Internet Service Providers*.

Rennie, J. D. M. (2000). ifile : An Application of Machine Learning to E-Mail Filtering. *Proceedings of the KDD (Knowledge Discovery in Databases) Workshop on Text Mining*.

Ruder, S. (2017). *An Overview of Multi-Task Learning in Deep Neural Networks*. https://ruder.io/multi-task/

Sarah Patterson. (2019). *How Customer Service Trends Are Changing in 2019: Highlights From the New State of Service Report*. https://www.salesforce.com/blog/customer-service-trends/?utm_source=Customer+Centric&utm_campaign=5df677a6fe-EMAIL_CAMPAIGN_2019_03_27_01_56&utm_medium=email&utm_term=0_f97fb3dbbf-5df677a6fe-138419807

Sun, C., Qiu, X., Xu, Y., & Huang, X. (2019). How to Fine-Tune BERT for Text Classification? *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *11856 LNAI*(2), 194–206. https://doi.org/10.1007/978-3-030-32381-3_16

Sutskever, I., Vinyals, O., & Le, Q. v. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, *4*(January), 3104–3112.

The Radicati Group, Inc. (2015). Email Statistics Report, 2015-2019. *Email Statistics Report*, *44*(0), 4. http://www.radicati.com/wp/wp-content/uploads/2015/02/Email-Statistics-Report-2015-2019-Executive-Summary.pdf

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *2017-Decem*(Nips), 5999–6009.

Wiedemann, G., Remus, S., Chawla, A., & Biemann, C. (2019). *Does BERT Make Any Sense? Interpretable Word Sense Disambiguation with Contextualized Embeddings*. http://arxiv.org/abs/1909.10430

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., & Brew, J. (2019). *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. http://arxiv.org/abs/1910.03771

Yang, W., & Linchi Kwok. (2012). Improving the automatic email responding system for computer manufacturers via machine learning. *2012 International Conference on Information Management, Innovation Management and Industrial Engineering*, *3*, 487–491.