Deep Learning Methods for Register Classification

Thesis: DP in Computer Science (MSc.)
University of Turku
Department of Future Technologies
November 2021
Prashant Mahato
Supervisors: Veronika Laippala,
Sampo Pyysalo

UNIVERSITY OF TURKU
Department of Future Technologies

PRASHANT MAHATO: Deep Learning Methods for Register Classification

Thesis: DP in Computer Science (MSc.)
Turku NLP Group
November 2021

For this project the data used is the one collected by, *Biber and Egbert (2018)* related to various language articles from the internet. I am using BERT model (Bidirectional Encoder Representations from Transformers), which is a deep neural network and FastText, which is a shallow neural network, as a baseline to perform text classification. Also, I am using Deep Learning models like XLNet to see if classification accuracy is improved.

Also, it has been described by *Biber and Egbert (2018)* what is *register*. We can think of register as genre. According to *Biber (1988)*, register is varieties defined in terms of general situational parameters. Hence, it can be inferred that there is a close relation between the language and the context of the situation in which it is being used. This work attempts register classification using deep learning methods that use *attention* mechanism. Working with the models, dealing with the imbalanced datasets in real life problems, tuning the hyperparameters for training the models was accomplished throughout the work. Also, proper evaluation metrics for various kind of data was determined.

The background study shows that how cumbersome the use classical Machine Learning approach used to be. Deep Learning, on the other hand, can accomplish the task with ease. The metric to be selected for the classification task for different types of datasets (balanced vs imbalanced), dealing with overfitting was also accomplished.

Keywords: Register, Genre, Register Classification, FastText, BERT, XLNet, Deep Learning, Attention Mechanism, Supervised Learning, Regularization

Acknowledgement

I would like to thank my supervisors, Sampo Pyysalo, for thorough guidance throughout my thesis and Veronika Laippala for providing me with the annotated data and for the instructions, guidance and encouragement throughout the work.

In Turku, November 2021

Prashant Mahato

TABLE OF CONTENTS

List of Figures

List of Tables

List of Acronyms

SOTA: State of The Art

CPU: Central Processing Unit

GPU: Graphical Processing Unit

TP: True Positive

TN: True Negative

FP: False Positive

FN: False Negative

SFL: Systemic Functional Linguistics

TF: Term Frequency

AGI: Automatic Genre Identification

IDF: Inverse Document Frequency

VSM: Vector Space Modeling

BoW: Bag of Words

SVM: Support Vector Machine

SVD: Singular Value Decomposition

POS: Part of Speech

LSI: Latent Semantic Indexing

SMO: Sequential minimal optimization

CV: Cross Validation

ML: Machine Learning

NLP: Natural Language Proessing

IR: Information Retrieval

CNN: Convolutional Neural Network

DNN: Deep Neural Network

RNN: Recurrent Neural Network

Bi-RNN: Bi-directional Recurrent Neural Network

W-RNN: Weighted Recurrent Neural Network

LSTM: Long Short Term Memory

BoW: Bag of Words

CBoW: Continuous Bag of Words

BERT: Bidirectional Encoder Representations from Transformers

OOV: Out of Vocabulary

MLM: Masked Language Modeling

NSP: Next Sentence Prediction

AUC-ROC: Area Under Curve-Receiver Operator Characteristic

1 Introduction

Computers have always been a means to simplify human life. Language is means of transferring knowledge among human beings. Natural Language processing is related to machine learning algorithms applied to linguistics. We use NLP for various day to day tasks like using *Alexa, Siri or Google Assistant*, searching on google whatever we want and so forth. NLP can provide human-like experiences by applying it to various problems like -

- information retrieval
- language translation
- email filtering
- text analytics
- speech to text (and vice-versa)
- chatbots and virtual assistants
- market intelligence

This thesis topic is related to text classification using Natural Language Processing. According to *Goldberg (2017, p.1)* :

> Human language is highly ambiguous ... It is also ever changing and evolving. People are great at producing language and understanding language, and

> are capable of expressing, perceiving, and interpreting very elaborate and nuanced meanings. At the same time, while we humans are great users of language, we are also very poor at formally understanding and describing the rules that govern language.

Excerpting from book 'Natural Language Processing with Python' *Bird, p. ix*

> We will take Natural Language Processing — or NLP for short –in a wide sense to cover any kind of computer manipulation of natural language. At one extreme, it could be as simple as counting word frequencies to compare different writing styles. At the other extreme, NLP involves "understanding" complete human utterances, at least to the extent of being able to give useful responses to them .

We are aware of the fact that the text following certain nuances has a particular pattern (e.g, conversation can differ from scientific exposition, considering the writing style). In context of NLP/linguistics we call it *register Manning et al. (2008).* It is similar to 'genre' *.Egbert et al. (2015)* describes register as

> text varieties that are initially defined based on their situational characteristics (e.g. participants, interactivity, communicative purposes, topic). Registers can be analyzed in terms of their pervasive lexico-grammatical linguistic charactersitics.

Hence, we can say that a register is distinguished based on particular situations and by the linguistic features (that the Deep Learning models can extract, in my case) that it shares *Argamon (2019).*

Objective

I am trying to apply few state-of-the-art Deep Learning algorithms viz. Google BERT (*Devlin et al. (2018)*), Facebook's fastText (*Joulin et al. (2016)*), also XLNet(*Yang et al. (2020)*) to see if they can generalize register patterns and perform register classification accurately. The main objective of this thesis is to do multi-class classification and compare the three algorithms' performance.

The Data

The data that I got is by *Egbert et al. (2015)*, which uses Web as Online Corpus (of registers). It has 26 labels, classified under four different classes (4 main classes (situational variables) have 22 sub classes): *viz.*

1. Narrative Register

   NE: News Report

   PB: Personal Blog

   SR: Sports Report

   HA: Historical Article

   TB: Travel Blog

   SS: Short Stories

2. Opinion, Advice and Persuasion Registers

   OB: Opinion Blog

   RV: Review

   DS: Description with Intent to Sell

   AV: Advice

   RS: Religious Blog/Sermon

3. Informational Descriptions, Explanations, and Procedures:

   IN: Informational Description

   IB: Informational Blog

   HI: How-to/Instructional

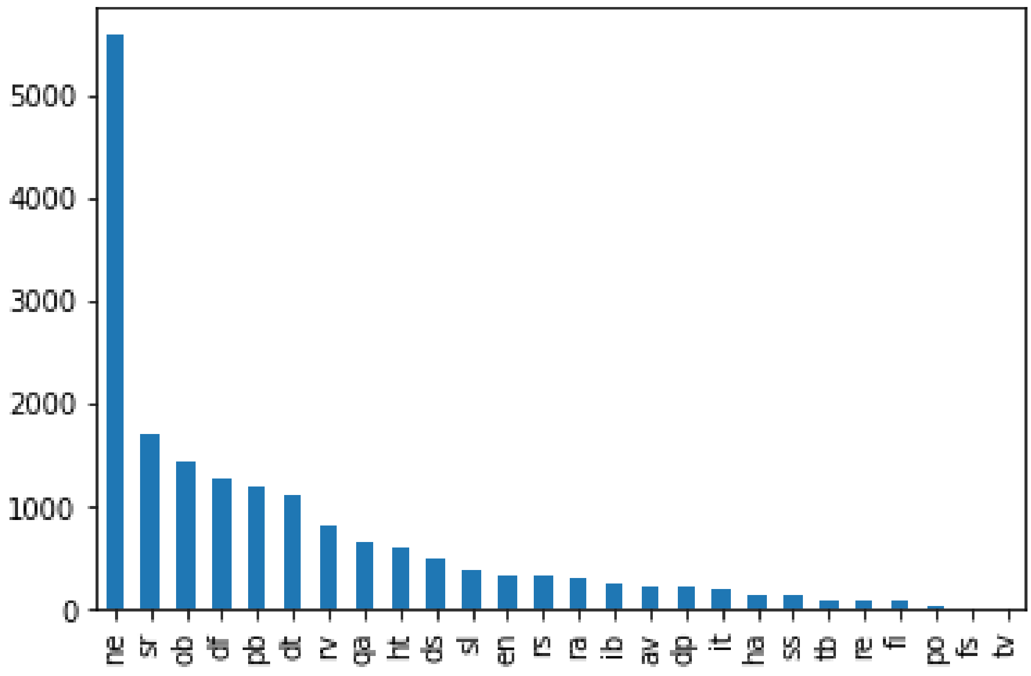


Figure 1.1: Class Distribution

RA: Research Article

DP: Description Blog

EN: Encyclopedia Article

FI: FAQ about Information

RE: Recipe

4. Oral Registers

   IN: Interactive Discussion

   IT: Interview

   SP: Spoken

It was scraped from various websites, following the book *(Biber and Egbert, 2018, p. 9-10,15,19,42)*. The annotation of registers was done manually. This work uses Web as a Corpus. Continuous Space of Registers Variation.

From the *Figure 1.1*, we can see that the data is highly imbalanced. So appropriate metric has to be used in this case. Or, balancing the classes with equal number of labels can be carried out.

Thesis Structure:

The first section is the introduction of the task. Introduction to *register* is included in this section. Also, the data is described in the introduction section itself. Basic terminology related to Machine Learning and Natural Language Processing are presented thereafter. After that, the related works done in this field are described. Then, the three methodologies that I have applied for this work, *FastText, BERT, and XLNet* are dealt with, followed by the results and conclusions. Further improvement that could be done with this task is discussed in the conclusion.

2 NLP and Machine Learning Related Terminology

2.1 Text Preprocessing

Text preprocessing is the most important initial step in NLP. We perform text preprocessing in order to transform text into a simpler form so that the machine learning algorithms can perform better. The basic steps in text preprocessing:

2.1.1 Tokenization

Tokenization is about splitting a string into smaller pieces. The algorithms that I am using tokenize the strings into words.

2.1.2 Normalization

There has to be uniformity in the sequence of words since computers are not good at dealing with variation (due to several form of a same word, and due to several contexts). Normalization reduces dimensionality of the input. So, the process of normalization can further involve :

Stemming/Lemmatization

Stemming converts the word to its stem form, whereas lemmatization considers the context of the word in which it is used. Either of the process could be used to make learning of words better.

Converting to Lowercase

Algorithms assign a different word vector for ’Capital’ different than for ’capital’. So it is important to have uniformity.

Noise Removal

Noise removal cleans text, so that unnecessary words like stop-words(articles, conjunctions etc.), punctuation, special characters, HTML tags can be removed. Noise removal also reduces number of inputs and the model doesn’t get interference by them for our task of text classification.

2.1.3 Cross Validation

Cross-validation is done by holding out a certain portion of data as unseen (as a test or a validation set) and use the rest for training. This can be useful in testing the model in an unseen data, as by doing cross validation, we are making it unbiased by reducing the variability (Often multiple rounds of cross-validation is carried out, which finally is averaged).

2.1.4 tf / tf-idf

It is often used in IR and keyword extraction. It can be used for feature weighing. Search engines use tf-idf for ranking relevance of articles. Words frequently used in the article are ranked higher than those used less. Mathematically, *idf* is represented as the logarithm of the number of documents to the number of documents with the particular term. Now *tf-idf* is the product of term frequency and idf.

$$tf = \frac{(Number\ of\ times\ term\ t\ appears\ in\ a\ document))}{(Total\ number\ of\ terms\ in\ the\ document)}$$

$$idf = \frac{log\_e(Total\ number\ of\ documents)}{(Number\ of\ documents\ with\ term\ t\ in\ it)}$$

*tf-idf=tf*idf*

2.1.5 Feature Engineering

Like in Machine Learning, in NLP, it is also essential to select only essential features from the document. Steps like removing stop-words, stemming, etc., can be considered as feature selection. idf-weighting for words with low or high frequency can also be used, as it can identify rare words, and it can convert strings to numbers.

2.1.6 Feature extraction

Before feeding into the ML classifier, each document has to undergo Vector Space Modeling. Bag-of-words is simply counting the occurrence of a particular word in a sentence (vector). Doc2Vec creates a numerical representation of the document regardless of what is the length. It is an unsupervised 3-layer shallow neural network. A set of documents is required for training a doc2vec model. A Word vector is generated for each word. Also, each document is represented by its vector. Hidden softmax layers are also trained, *Le and Mikolov (2014)*. BoW tf-idf can be one way of feature extraction, *Liu et al. (2018)*. Doc2Vec is also one of the ways.

2.1.7 Word Embeddings

Similar words appear in a similar context. In NLP, we have numerical representation for words as vectors called *Word Embedding*. Each word is mapped as a vector. These vector values in the embedding are learned using a neural network. Hence we will get a real-valued vector of fixed dimension, depending on the model we are using.

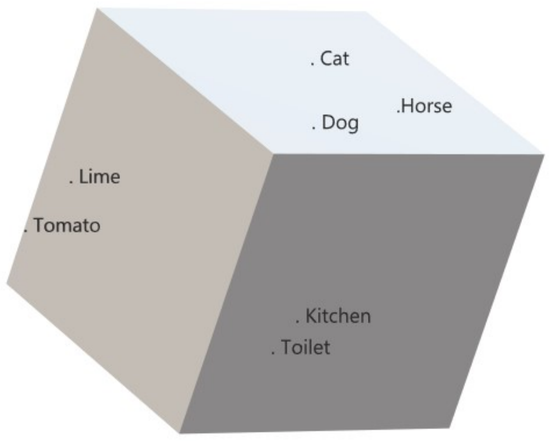


Figure 2.1: Word Embedding

Each word is mathematically represented as a vector. The word is represented as a one-hot vector (Categorical features represented as one-hot array) value multiplied by embedding weight matrix (neural network).

2.1.8 n-gram

A sequence of n characters or words in a sentence, is referred to as n-gram. n-gram follows the principle 'You shall know an object by the company it keeps', *Sadeghi et al. (2014)*. It is used in NLP to identify letters or words that adhere together. The n-grams information enriches word vectors with sub-word information. It is often used in predictive text input and speech-to-text. FastText incorporates a sub-word model. It can efficiently handle Out Of Vocabulary (OOV) words by using subword information. It also has shared parameters for words with the same roots.

2.1.9 Some Deep Learning Methods Used in the Field of NLP

Perceptron

Perceptron is the simplest neural network. In Perceptron multiple binary inputs produce a binary output. First the input X is multiplied by weight, bias added to this term gives us with weighted sum. The weighted sum is then passed through an activation function to give an output.

Weight and Bias Bias is a constant vector, added to the product of input and weight. The mathematical expression of perceptron is $Y = W*x+b$, where W is weight and b is bias. Weight decides what importance to give to the individual input, whereas bias offsets the results (shifts the result of activation function to positive or negative side). Hence there is better generalisation of neural networks (bias controls when to activate the activation functions).

Activation Functions Each neuron in the Neural network possesses an activation that tells it to what extent it should be stimulated to the connected neuron. It takes input as inputs, multiplied with the weights, added with bias to produce the binary output. This can further be passed to subsequent neurons.

Feed Forward Neural Network

Feed-Forward Neural Network can be thought of as a non-linear mapping such that u = G(x). Here input patterns x is mapped to output patterns u by a mapping function G, *Bebis and Georgiopoulos (1994, p. 27-31).* A multi-layered Feed-forward Neural Network would have an input layer, multiple hidden layers, and an output layer.

In a neural network, the nodes are interconnected so as to pass the information and compute the output. It is trained by using back-propagation, and minimizing the loss/cost in the error function. Various ways of increasing performance include:

Weight Initialization Weight initialization is used so that the neurons in neural networks learn properly from the features. For example, I have used random initialization of weights, using pytorch's inbuilt methods (For BERT).

Initially, the weight matrices contain random values, which are then gradually adjusted based on the feedback values. Training the neural network updates the weight values. Iteration over all the batches and epochs result in a finally updated weight matrix.

Learning Rate Learning rate is a hyperparameter which tells what amount of weight is updated while training and is used to control how quickly the model will adapt to the problem. It can take any value between 0.0 and 1.0.

Batches and Epochs Batch is one of the hyperparameters in the neural network. We use batches to load all of our data in memory to not make it too expensive for our CPU/GPU. Once all the samples in a batch of operations are complete, the internal model parameters can be updated then after. Error is calculated after a batch of operations. An epoch, on the other hand, is a complete pass through the entire dataset(forward and then backward). Increasing/decreasing it can affect neural network performance too. An epoch can have one or several batches. In deep learning algorithms, less epochs can cause underfitting, so increasing epoch size can make it optimal. Also, it should not be increase too much so that it can cause overfitting.

Regularization It is used to handle over-fitting by penalizing the weight, which maps input to output. The additional penalty term is used to control the coefficients of the error function. Hence the excessively fluctuating function is controlled. Typically *L1 (sum of absolute values of weight)* and *L2 (sum of squared values of weight)* vector norm penalty is used.

Optimization Loss Loss function is calculated at the end of each epoch. It tells how much quantity needs to be minimized during the training. Based on loss function, optimizer determines how the network will be updated.

Recurrent Neural Network (RNN)

RNNs work on sequential data: RNN is often used when pattern in data change with time. In terms of text data, the word gets transformed into vector first. Then in time series, the vector is processed one by one, passing through a succession of *cells*. Thus, the hidden states (memory) propagate sequentially as it passes through the cells. The problem, however, with RNN is that with longer sequences, it is difficult for them to remember the past states as there is exponentially small gradients and decay of information through time. Hence, there is inability to capture relevant dependencies in long sequences, *Cho et al. (2014)*

LSTM

LSTMs are also a type of RNN. In addition to having recurrent nodes, they also have an internal state (to forget or remember some information). Input value, the previous output and the internal state are all used in node calculation. We can get output value and also internal state is updated. LSTMs have a parameter known as gates to control the flow of information within the nodes. These gates parameters are weights and biases. Hence LSTM nodes are more complicated than RNN nodes and complex dependencies and sequences of data can be learnt, *Cheng et al. (2016).* So, they can better capture the dependencies between the long sequences, although they are slower than RNN.

CNN

Convolutional Neural Networks, trained on pretrained word vectors can be used for sentence classification. Word vectors, or a word is projected as a sparse representation (of an embedding) is passed into a hidden layer (which acts as feature extractors) . Hence we get the semantic feature of a word. CNN can have *filters* to learn multiple features, and also, it does not forget the past states easily as there are local dependencies and distance between positions is known, *Kim (2014).*

Transformers

The Transformer relies on self-attention for information interaction among the input sequence. This is the advancement made in RNN and CNN *Vaswani et al. (2017)*. Self-attention is an *attention mechanism (which helps focus on the important parts of an input data, and fade out the rest)* relating to the different positions of a single sequence so as to calculate representation of the same sequence. Thus, self-attention allows the inputs to interact with each other to compute which token should be paid attention to more.

Global Attention

Global Attention applies in terms of transformers (Refer *2.1.9*). Since the Vanilla encoder-decoder suffers from vanishing gradients, global attention mechanism was formed. When attention is placed on all source states, it is called Global Attention. Referring to *Luong et al. (2015)*, Global Attention can be described as :

1. At each position, there is a variable-length *alignment vector*
2. There is a current *target state* and *all source states*
3. Global *context vector* is computed as weighted sum of attention weights as well as the hidden states that maps to the input sequence.

Hence, all the words on the source side are attended to to get the target word, so it is expensive.

Pretrained Model

A neural language model, which often has millions of parameters is trained on a large unlabelled corpora. During this process, the word representations are learned in a pretrained model, which can be reused in supervised training for a task (e.g. genre classification) with optional fine-tuning, thus saving us a lot of time and effort, *Gururangan et al. (2020)*. Pretrained models are availed with a large amount of training data.

2.1.10 Classification Report

A Classification report is used to assess the quality of predictions from a classification model. The metrics are defined in terms of true positives & false positives and true negatives & false negatives. A true positive is when a class is actually classified correctly (gets correct label). A false positive is when some wrong class is classified in a true class (for example, a sick person is classified as a healthy person). Some terms used in the Classification Report are:

Precision is the labels correctly predicted among the labels predicted by the model.

Recall : is the number of labels that were successfully predicted, among all the real labels.

F1-Score : is the harmonic mean of both of the above. Micro-F1 score are used to assess the quality of multilabel problems. (1 to indicate the best of micro-precison and micro-recall)

Support : By Support, the inference from the dataset that can be made is the number of times the class actually appears in the specified dataset.

Mathematical Formula:

$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{Recall} = \frac{TP}{TP + FN}$$
$$\text{F1-Score} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

2.1.11 Confusion Matrix

The Confusion-Matrix takes the training data, test data, the classifier and shows the visualization of the performance of an algorithm. The report shows a comparison of the test values' predicted classes versus their actual classes. It is used to test which classes are most easily confused. [1].

[1]https://www.scikit-yb.org/en/latest/api/classifier/confusion_matrix.html

3 Background

Lots of work have been done in the field of text classification. A lot of effort is required in terms of classical machine learning. It is my approach to evaluate a number of SOTA algorithms and get results for my task of *register classification*. We see in this section, the works done in the field of register classification: classical machine learning approach and deep learning algorithms used for the task.

3.1 Register

There has been stackoverflow tag prediction online competetion (*Schuster et al. (2013)*). Various works have been done in the field of movie or book Genre classification by some companies. *Argamon (2019)* has done a work of *Register Analysis*:

Register Analysis - This is the task of classifying texts into a particular register using some computational methods. For the feature extraction part, machine learning techniques have been used. Then classification models were built. Also, multidimensional analysis is done. The purpose of multidimensional analysis is to find a set of continuous dimensions that exhibits register-relevant variation, *Argamon (2019). Egbert et al. (2015)* has developed a web tool that enables user-based text classification of texts (from the internet). It can be inferred from the work that registers can be analyzed in terms of their pervasive lexico-grammatical linguistic characteristics. They have worked on describing linguistic variation on the web by developing a hierarchical register framework and a decision tree to take a range of facets as well. Hence, information necessary to classify text

was developed (using a a user-based web register classification instrument) . They have taken the multifaceted structure of the register framework (multiple characteristics of the text). Thus, it is possible to get situational information in an incremental way. They have classified register categories based on the situations in which language is used, and further analyses have been done for investigating the characters of the defined registers.
They developed a framework that itemized the registers found on the web. Users are themselves assigning registers to the web texts, being able to modify the categories themselves when needed. Also, in one pilot phase, they modified the classification rubric to a decision tree, where situational parameters *viz.* mode, interactivity, communicative purpose, and factuality are considered for register classification. This resulted in the hierarchy in classification. Few sub-registers were highly frequent and easily identifiable. Hence users were able to select from a drop-down of various categories of registers.

For the user-based register classification, the first instrument developed was a simple web-tool of register categories. Considering reliability as a factor, a flowchart was developed later on, which followed situational binary variables as below:

mode (spoken/written)
interactivity (multi-participant/single author)
purpose (narrative/descriptive)
factuality (opinion/objective)

The study concluded that even non experts can, by using bottom-up approach, classify documents using a decision tree into the general registers, sub registers and register hybrids (one webpage having characteristics of multiple registers). They have concluded that the hybrids have more power to incorporate general patterns more than others.

Argamon (2019) has also made several studies related to academia, like finding differences in social communities based on registers usage, the evolution of registers in scientific publications, variation in different registers of various (World) Englishes. Also, works are done to analyze the similarity between registers, say based on linguistic features they share. *Clarke and Grieve (2017)* has used Multidimensional Analysis for Abusive Language detection. Also, *Egbert et al. (2015)* mentions that registers can be used to improve word disambiguation software, taggers, parsers, and IR tools. *Biber and Egbert (2016)* mentions the use of Web-as-corpus by researchers/linguists for language teaching, information scientists for information retrieval, and computational researchers for improving results of search engines and for spam filtering. There are some works done to improve Information Retrieval tasks and Search Engines. *Abdullah and Zamil (2018)* has used text classification during the indexing phase, as an improvement to classical IR Systems, to improve the results of IR systems. *Karlgren (1996)* has used register categories to improve information retrieval and search.

3.2 Machine Learning

Argamon (2019) has used conventional Machine Learning techniques for feature extraction and used techniques like Multidimensional Analysis Also, Syntactic parsing and shallow parsing are used as an alternative. Systemic Functional Linguistics (SFL) is used to observe differences between the various level of abstractions. Simply stated, SFL is a term used by applied linguists, which studies similarities and the contrast between a set of texts (having common/similar topic), based on metafunctions like *interpersonal (Speaker - addressee interaction)*, *textual (Organisational text)* and *experiental (representing experience) Briones (2016)*. Support Vector Machines (SVM), Neural Networks, Rule-Based Learning, and Bayesian Regression are some of the ML methods mentioned in the paper. The paper considers appropriate evaluation metrics like per class precision and recall, as overall accuracy may not be a good metric to consider. Further, it explains that accuracy can be high if the accuracy of majority is high. It has mentioned handling majority and minority classes, using undersampling and oversampling respectively. For text represen-

tation, it has used **Stylistic Text Features** as in *Herke-Couchman and Canzhong (2004)*, which has multidimensional approach of representing a register. One of the dimensions is *stratification*, which orders space into four different levels (*viz.*context, semantics, lexicogrammar and phonology/ graphology). Another dimension is *Ideational metafunction*, that divides space into *ideational, interpersonal and textual* functionally diverse modes. Ideational refers to the world around us: experimental (processes, circumstances, and participants) and logical (modification and exemplification). The third dimension is *instantiation*. At any point in instantiation, we can pick up a distinct task (eg. translating at one end, developing reference grammar at another).
Biber (2004) has used Multidimensional Analysis (Factor Analysis) and identified seven factors of variation for the registers. Factor Analysis has been used not only for lexical and syntactic features but also over cohesive text features (linguistic features that link sentences in a text together e.g. lexical syntax and grammar). Also, the article mentions the work of *Teich and Fankhauser (2010)*. The paper discusses the use of k-means clustering for investigating how the mixed type registers are related to the pure registers (e.g., how bio-informatics is related to biology and computer science). Also, SVM is used for this purpose.

Vidulin et al. has dealt with transforming a multilabel ML problem into several single-label classification problems, using a supervised ML approach. Since a web page is a complex document, the single page can be classified using multilabel classification. They have transformed the problem into multi-class and then into a set of binary sub-problems. Finally, standard ML algorithms have been used on the transformed data. The model that deals best with multi-genre web pages is then determined. Based on the result of a set of binary classifiers (one per genre) that can learn overlaps between several genres and classify a complex web document into a single label, then a zero-to-multi genre assignment is done. This is binary approach. In multi-class, the classifier is able to learn the overlap between the different genres (due to the induction of a single classifier). and generalizes the most dominant register. The performance, however, for such a classifier was low.

There were 20-genre collection corpus to cover aspects *viz. content, linguistic form, visual form, and context of a web page*. The features were separated into four groups: *surface, structural, presentation, and context*.

As mentioned already, the work has transformed a multi-class problem into a set of binary problems. Support Vector Machine (SVM) and Adaptive Boosting (AdaBoost) are used to learn classifiers (For easier model selection and better generalization performance AdaBoostSVM is used, *Li et al. (2008a)*. The way SVM works is by finding the best decision boundary (hyperplane) to separate two classes. A weakened SVM was used to provide weight to the classifier. Hence by tuning (kernel) parameters so that one classifier doesn't come as dominant by having too much weight, *Li et al. (2008b); Govindaraj (2016)*. The evaluation was done based on the Exact match ratio, which is similar to accuracy micro-averaged measures, which averages over all the web pages, and macro-averaged measures, which weigh all the genre categories equally regardless of their frequencies.

The paper suggests that precision is a good property of measuring.

Radovanovic et al. (2009, p.861-893) presents idea of constructing a meta-search engine powered by text classification techniques. The paper says, text categorization can have applications like text filtering, word-sense disambiguation and categorization of web-pages.

We can see the detailed implementation of the model in this work. The document is represented into its numerical counterpart using methods like *bag-of-words*. For measuring similarity between documents, cosine similarity is used. In deep-learning methods, word-embeddings serve this role. For dimensionality reduction stemming is used. Also, feature selection and feature extraction (LSI) has been used to reduce the number of features. LSI uses SVD to decompose the term-document matrix to transform the document vectors to lower-dimensional space Refer *(3.1)*. After this, modeling can be done using one among the models: *viz. Perceptron, SVM, Naive-Bayes, Nearest-neighbor classifier, Decision tree*

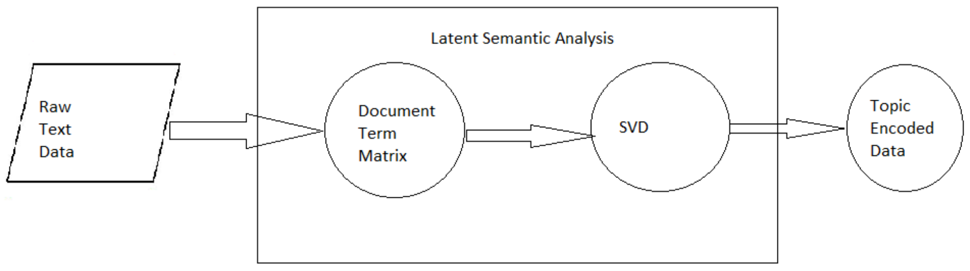


Figure 3.1: Latent Semantic Indexing

To evaluate the classifiers, cross-validation method is used. Evaluation of classifier is done using *accuracy, precision and recall.*

The datasets used were from six categories: arts, computer, sports, music, role-playing, and physics.

3.3 Deep Learning

There are deep learning algorithms text classification. Some of which are described in *Minaee et al. (2020)*. My effort is to evaluate some of the SOTA deep learning methods.

Prior to transformers, LSTM had been used for language translation, text classification, etc. LSTMs are a advanced form of RNNs capable of learning long-term dependencies. LSTMs were capable of solving the problem of vanishing gradients of the traditional RNNs. But LSTMs also suffered from the following drawbacks :

1. **Slow to Train** Sentences are processed sequentially, words by words. This is the reason it cannot be trained in parallel.

2. **Past Information Retained Through Past Hidden States** In Sequence-to-Sequence models, each state is dependent on the previously seen state. Hence encoding of the word strongly affects the next time step. But, after certain time-steps this dependency is lost . In LSTM, the dependency range can be boosted a bit due to the deeper processing of the hidden states, utilizing *gates* to allow only certain information to pass through the *cell states*. On the other hand, the self-attention mechanism used by Transformers can learn the dependency between the current word and the precious part of the same sentence, *Vaswani et al. (2017).*

3. **Not Truly Bidirectional** The true meaning of words is not entirely captured. Even the bidirectional LSTMs are learning right to the left or left to right, the context separately and then concatenating them. So the actual context is not fully captured.

Xu et al. (2020) has solved some of the problems in LSTM using per-layer-nonlinearity with depth-wise LSTM, instead of residual connections.

Lyu and Liu (2020) combined CNN and RNN employing the strengths of the two networks. CNN captures the local invariant features, so in this work, a 2D weight matrix is learned. Hence, the importance of each word from different aspects is represented. RNN (bidirectional) is utilized to learn the word contextual information. A neural tensor on top of bidirectional RNN obtains the fusion of bidirectional contextual information around the focus word. This new neural network architecture was evaluated on six different datasets for Text Classification.

Beltagy et al. (2020) has worked on decreasing the runtime of transformers from quadratic to linear. Also, they work on longer sequence texts (4096 tokens), 8 times longer than what BERT can work as BERT is pretrained to work on a maximum of 512 token-size. One of the works they have done is document classification. They have done replacement of self-attention, combining local windowed attention with a task motivated global attention. The proposed solution utilized multiple layers of attention to building contextual representations of the entire context. They have combined windowed local-context self-attention global attention to learn higher contexts in less time.

Sun et al. (2020) has done 3 works to fine-tune BERT for text classification:

1. **Further Pretrain BERT:** Further pretraining has been done in following ways:

   Within-task pre-training:

   In-domain pre-training

   Cross-domain pre-training

2. **Fine-tuning BERT:** To fine-tune BERT three things are necessary:

   Pre-processing of long texts Pre-processing of long texts (max allowed sequence length is 512). They have applied some truncation methods to deal with long texts. The logic they have given is that the key information is contained in the beginning and ending of an article.

   Layer Selection Selecting the most effective layer for text classification out of the embedding layer, 12-layer encoder and a pooling layer.

   Overfitting Problem For dealing with overfitting problem (occurs when model also memorizes the noise and model can't perform well on real data) appropriate learning rates were adopted.
   Multi-task fine-tuning refers to sharing of the knowledge that is obtained from several related supervised tasks. All the tasks share the BERT layers and the embedding layer, except the final classification layer. Hence, each task has a private classifier layer.

3. **Fine-tuning BERT for Target Class:** Investigation of fine-tuning methods were done. Some of which were:

   layer-wise learning rate

   catastrophic forgetting: In transfer learning, when pre-trained knowledge is erased when new learning is done, it is referred to as catastrophic forgetting. A lower learning rate was required for overcoming this.

low-shot learning problems: This occurs when there is less training data.

They have used BERT for question classification, topic classification etc.

4 Methodologies

The practical implementation has been carried out using three different algorithms *Fast-Text, BERT, and XLNet*. This chapter deals with the working of these algorithms.

4.1 fastText

fastText was created by Facebook's AI Research (FAIR) lab. fastText is an extension of the Word2Vec model. It represents each word as n-grams. It uses either skip-gram or CBOW (the former tries to predict the neighbours of a word whereas the latter predicts a word on the basis of the surrounding context) algorithm thereafter for the learning model. This model can be considered as a bag-of-words model with a sliding window. It works well with rare words. Word2Vec and Glove both fail at rare words.
The work of *Mikolov et al. (2013a)*

> We observed that it is possible to train high quality word vectors using very simple model architectures, compared to the popular neural network models (both feedforward and recurrent)

In fastText word embedding, we can set a character n-gram, which can represent rare unknown words. In such embedding, a word can be represented as subwords of n-grams. FastText uses a bag of n-grams as additional features to capture local word order information.

For example 'where' can be represented as <wh, whe, her, ere, re >

(Source — *Bojanowski et al. (2017)*)

The sum of the character n-grams is equal to the word representation (vector) of the word. This can, for example, be useful in a noisy context when some words are misspelled. Hence (Out of Vocabulary) OOV words can be better handled by FastText. (By utilizing the power of n-grams), *Bojanowski et al. (2017)*

I have used fastText as a baseline for text classification as it is fast in train/test. Good results can be obtained when proper parameters are used. fastText can scale to a very large corpus.

4.2 Exploring Word Embeddings:

Embeddings are often very good at capturing syntactic and semantic patterns or regularities in a language. For e.g., if country/capital relationship is to be learned, and with the induced vector representations, "Tokyo + Finland - Helsinki" results in a vector very close to "Japan", *Mikolov et al. (2013b)*. By default, Using C-BOW, we get a vector of length 100 for each word, whereas using skip-gram, we get a 300-length vector (by default).

For this experiment skip-gram and CBOW models have been used. The former can predict a target based on surrounding words. The latter predicts based on its context, which is represented as a bag of words of fixed size window around the target word. *Figure 4.1* can help make it clear. (The target word I have used is 'coffee')

Experimenting with Embeddings:

- Getting CBOW Word Embedding For example, in case of fastText, we can have an embedding for sentence ' I have Flu' as in *Figure 4.2*. We can see that a word vector is a sequence of numbers. Using Skipgram model for the word or token 'university', we see the following words that are similar to university *Figure 4.3*.

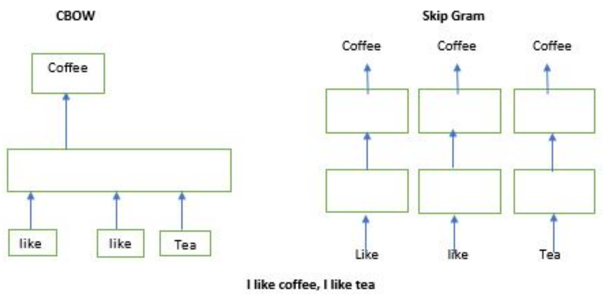
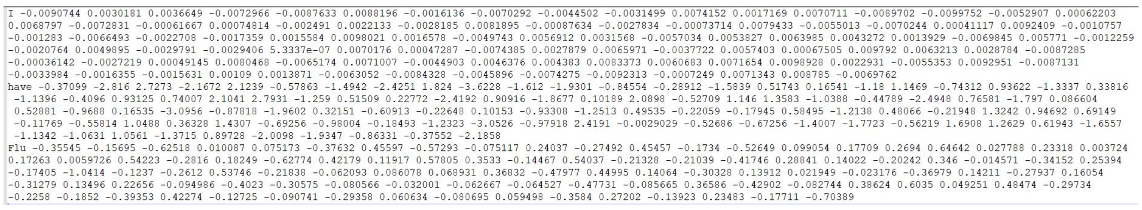


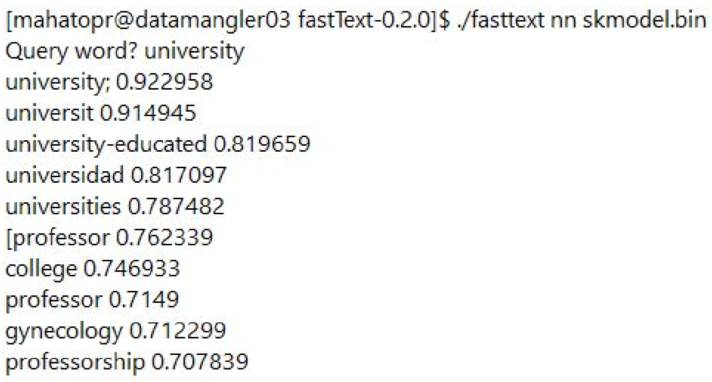
Figure 4.1: CBOW vs SkipGram



```
I -0.0090744 0.0030181 0.0036649 -0.0072966 -0.0087633 0.0088196 -0.0016136 -0.0070292 -0.0044502 -0.0031499 0.0074152 0.0017169 0.0070711 -0.0089702 -0.0099752 -0.0052907 0.00062203
0.0068797 -0.0072831 -0.00061667 0.00074814 -0.002491 0.0022133 -0.0028185 0.0081895 -0.00087634 -0.0027834 -0.00073714 0.0079433 -0.0055013 -0.0070244 0.00041117 0.0092409 -0.0010757
-0.001283 -0.0066493 -0.0022708 -0.0017359 0.0015584 0.0098021 0.0016578 -0.0049743 0.0056912 0.0031568 -0.0057034 0.0053827 0.0063985 0.0043272 0.0013929 -0.0069845 0.005771 -0.0012259
-0.0020764 0.0049895 -0.0029791 -0.0029406 5.3337e-07 0.0070176 0.00047287 -0.0074385 0.0027879 0.0065971 -0.0037722 0.0057403 0.00067505 0.009792 0.0063213 0.0028784 -0.0087285
-0.00036142 -0.0027219 0.00049145 0.0080468 -0.0065174 0.0071007 -0.0044903 0.0046376 0.004383 0.0083373 0.0060683 0.0071654 0.0098928 0.0022931 -0.0055353 0.0092951 -0.0087131
-0.0033984 -0.0016355 -0.0015631 0.00109 0.0013871 -0.0063052 -0.0084328 -0.0045896 -0.0074275 -0.0092313 -0.0007249 0.0071343 0.008785 -0.0069762
have -0.37099 -2.816 2.7273 -2.1672 2.1239 -0.57863 -1.4942 -2.4251 1.824 -3.6228 -1.612 -1.9301 -0.84554 -0.28912 -1.5839 0.51743 0.16541 -1.18 1.1469 -0.74312 0.93622 -1.3337 0.33816
 -1.1396 -0.4096 0.93125 0.74007 2.1041 2.7931 -1.259 0.51509 0.22772 -2.4192 0.90916 -1.8677 0.10189 2.0898 -0.52709 1.146 1.3583 -1.0388 -0.44789 -2.4948 0.76581 -1.797 0.086604
 0.52881 -0.9688 0.16535 -3.0956 -0.87818 -1.9602 0.32151 -0.60913 -0.22648 0.10153 -0.93308 -1.2513 0.49535 -0.22059 -0.17945 0.58495 -1.2138 0.48066 -0.21948 1.3242 0.94692 0.69149
 -0.11769 -0.55814 1.0488 0.36328 1.4307 -0.69256 -0.98004 -0.18493 -1.2323 -3.0526 -0.97918 2.4191 -0.0029029 -0.52686 -0.67256 -1.4007 -1.7723 -0.56219 1.6908 1.2629 0.61943 -1.6557
 -1.1342 -1.0631 1.0561 -1.3715 0.89728 -2.0098 -1.9347 -0.86331 -0.37552 -2.1858
Flu -0.35545 -0.15695 -0.62518 0.010087 0.075173 -0.37632 0.45597 -0.57293 -0.075117 0.24037 -0.27492 0.45457 -0.1734 -0.52649 0.099054 0.17709 0.2694 0.64642 0.027788 0.23318 0.003724
0.17263 0.0059726 0.54223 -0.2816 0.18249 -0.62774 0.42179 0.11917 0.57805 0.3533 -0.14467 0.54037 -0.21328 -0.21039 -0.41746 0.28841 0.14022 -0.20242 0.346 -0.014571 -0.34152 0.25394
-0.17405 -1.0414 -0.1237 -0.2612 0.53746 -0.21838 -0.062093 0.086078 0.068931 0.36832 -0.47977 0.44995 0.14064 -0.30328 0.13912 0.021949 -0.023176 -0.36979 0.14211 -0.27937 0.16054
-0.31279 0.13496 0.22656 -0.094986 -0.4023 -0.30575 -0.080566 -0.032001 -0.062667 -0.064527 -0.47731 -0.085665 0.36586 -0.42902 -0.082744 0.38624 0.6035 0.049251 0.48474 -0.29734
-0.2258 -0.1852 -0.39353 0.42274 -0.12725 -0.090741 -0.29358 0.060634 -0.080695 0.059498 -0.3584 0.27202 -0.13923 0.23483 -0.17711 -0.70389
```

Figure 4.2: Word Embedding CBOW

- Getting Nearest Neighbours with SkipGram model as shown in *Figure 4.4*



```
[mahatopr@datamangler03 fastText-0.2.0]$ ./fasttext nn skmodel.bin
Query word? university
university; 0.922958
universit 0.914945
university-educated 0.819659
universidad 0.817097
universities 0.787482
[professor 0.762339
college 0.746933
professor 0.7149
gynecology 0.712299
professorship 0.707839
```

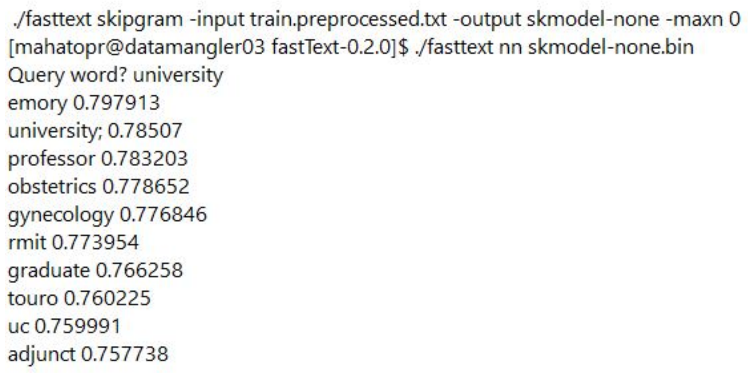
Figure 4.3: Nearest Neighbours Skipgram

- Measuring Similarities between Word Vectors by utilizing word analogies, also discussed in *section 4.2*. As words in an embedding are just vectors, we can use operations like addition and subtraction to get *Word Analogies*, *Allen and Hospedales (2019)*. For e.g., the analogy for man is to king as woman is to ..?

can be expressed as W_k-W_m+W_w, where W_k is the word embedding of the vector king, W_m is the embedding of man and W_w is the word embedding of the vector woman.

In fastText, we can enter *query triplets i.e, 3 word vectors* and observe the word analogy, doing additions and subtractions.

- Comparing Power of Embedding Using n-grams: Word n-grams can be used n-grams can be helpful in many scenarios, for e.g., for noisy documents, for segmenting compound words etc., *Kim et al. (2019)*
  Details about n-grams is dealt in *subsection 2.1.8*



```
./fasttext skipgram -input train.preprocessed.txt -output skmodel-none -maxn 0
[mahatopr@datamangler03 fastText-0.2.0]$ ./fasttext nn skmodel-none.bin
Query word? university
emory 0.797913
university; 0.78507
professor 0.783203
obstetrics 0.778652
gynecology 0.776846
rmit 0.773954
graduate 0.766258
touro 0.760225
uc 0.759991
adjunct 0.757738
```

Figure 4.4: Nearest Neighbour without Using n-grams

4.3 Working of FastText

fastText uses shallow neural networks, called a Continuous Bag of Words (CBOW) and skipgram, for the part of the word representation. It speeds up computation using hierarchical softmax over softmax, as we do not need to compute all the label probabilities, *Joulin et al. (2016).*

fastText model is capable of learning word representations and sentence classification. One good feature of fastText is that it can support multiprocessing during training. There are various types of support while training -CBOW or Skip-gram models using softmax and hierarchical softmax loss functions.

4.3.1 Working with fastText

There are two ways we can use fastText: one is in command line mode and the other is in python. Bash commands were used in command line mode. There is an option to activate hyperparameter optimization – By providing a validation file with -autotune-validation argument. So, working with fastText becomes quite easy.

4.3.2 Text preprocessing in fastText and the Hierarchical Softmax

In Hierarchical Softmax, a (balanced) binary tree is used instead of a flat layer as in Softmax. As a result of hierarchical softmax, the accuracy is decreased, but computation power is really high and the imbalance in data is handled. Each word is represented as a leaf of the tree. In softmax, we get a probability of a word being the target word given the context. As the softmax layer is very big, the computational cost is very high as we have to calculate all the pre-activation for every training example.
On the other hand, in Hierarchical Softmax, the probability of observing a word is decomposed into a sequence of probabilities corresponding to the choices of the word belonging to a particular group. As FastText deals with Word n-grams and hierarchical split, the imbalance was not a problem, *Joulin et al. (2017).* To illustrate the concept of hierarchical softmax with an example: Suppose we have a tokenized document. ['I','am','having','common','cold','and','not','corona'] We can further simplify the equation in the figure as in the equation below:

$$p('corona'|context) = sigm(b1 + V1.h(x)) * (1 - sigm(b3 + V3.h(x))*$$
$$(1 - sigm(b6 + V6.h(x))$$

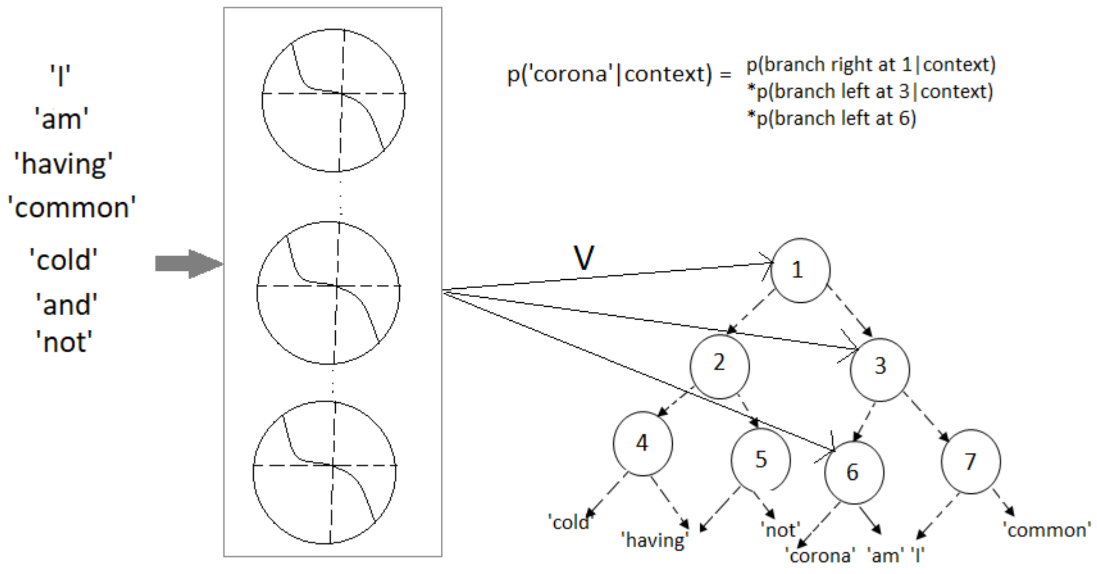


Figure 4.5: Hierarchical Softmax Function

In the equation above, V is the matrix of parameters where all the nodes in the tree is connected to the sigmoidal output units in the neural network, h is the hidden layer and b is the bias.

We can see from the figure - in the context we have - that the neural net is extracting the word representation, concatenating them, and computing the hidden layer of the neural net. The hidden layer will connect to all of the nodes in our tree. So internal nodes will connect to V. All the nodes in the tree are connected to sigmoid output units of the Neural Network. In short, V multiplied by hidden layer, and applying sigmoid gives us probability of branching right or left, *Rong (2016)*.

4.3.3 Summary

To summarize, following steps have been followed, loading data till autotuning:

1. **Prepare training and test/validation data**

   I had 17589 training samples (documents), 2212 test samples. The training data is used for model training, whereas test data is used to check model accuracy. Test data is isolated from the training data and can test the model's capability for unseen data. Validation data is used while training the model for tuning hyperparameters, avoid overfitting, i.e., check the performance difference in training and validation data.

2. **Text Preprocessing for making results better**

   For text preprocessing, all the noises and stop words in the data have been removed and they were translated to lowercase so that 'Data' and 'data' , for example, would have the same meaning for the model. Also, the data is shuffled so as to prevent overfitting.

3. **Make first supervised model specifying our input train data and output model**

   After preprocessing text, the loss obtained was little more than the one without preprocessing (around 1%).

4. **Specifying various parameters: epochs, learning-rates, word n-grams**

   This had been tried out manually. Also, automatic hyperparameter optimization as in (5) can help us estimate one of the best parameters selection. fastText allows automatic hyperparameter optimization, using keyword *-autotune-validation*. However, in this experiment, though, the best results were found specifying a few parameters manually.

5. **Applying Hierarchical Softmax Loss Function**

   Our dataset was still small. Hierarchical softmax is used when we have a large number of data and a large number of labels. With this loss function, the softmax is approximated much faster. This does so by building a binary tree corresponding to the labels. Each node uses binary decision activation, and thus the path of traversal is minimized by deciding whether we go left or right.

4.4 BERT - Bidirectional Encoder Representations from Transformers

The traditional methods used either the previous 'n' tokens, or the next 'n' tokens to predict the current token. There was no method that could take the previous and next tokens at a time. BERT, which works on the principle of 'transformers', solved this issue of 'bidirectionality'. Transofrmers employ encoder-decoder architecture, just like RNNs. Unlike RNNs, in transformers, input sequence can be passed in parallel. It has an encoder as in Transformer, which has self-attention and feed-forward neural network. Thus the model learns the context of a word as it is dynamically informed by its surroundings.

4.4.1 BERT Working Principle

The transformer is based on the self-attention mechanism. BERT is a large-scale transformer-based language model, *Vaswani et al. (2017).* BERT uses byte-pair encoding iteratively replaces the most frequently used byte pairs in a sequence, with a single unused byte, *Sennrich et al. (2015).* BERT developers trained a huge model with parallel computation imparting a significant improvement on RNN, *Devlin et al. (2018).* BERT is trained in several different ways in unsupervised data. It has token input, token embeddings, and positional embeddings. We can use BERT to extract high-quality language features from our text data as it has bidirectional training, hence better context awareness.

BERT that I am using (small uncased) has a vocabulary of 30 K tokens with 768 dimensional embeddings. The self-attention mechanism allows each word to be processed independently, and parallelization is possible. The vocabulary of BERT is fixed, so if there is any unseen word, BERT treats it smartly using the word-piece model breaking down the tokens into multiple sub-words.

Transformers are faster than RNNs as words can be processed simultaneously. Context of words is better learned as they can learn context from both directions simultaneously (deeply bidirectional).

4.4.2 Transformer Flow:

The Encoder consists of two layers, a self-attention layer and a feed-forward neural network. Attention considers a linear combination of hidden states to each input word and determines what weight to give to which word. For example, if we want to find attention for a particular word in a sentence, each embedding of the word having three vectors (key, query and value) obtained by matrix multiplication. The intuition behind self-attention is that inputs interact with each other and get a way of paying attention to a particular word,

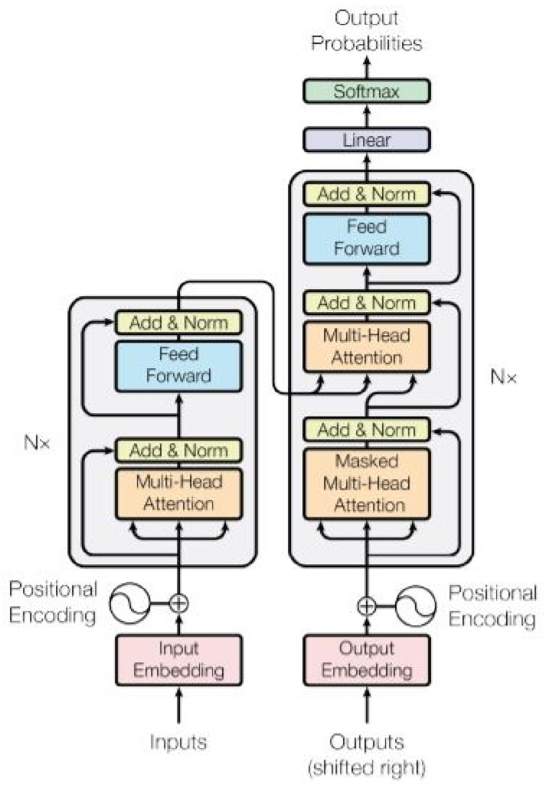


Figure 4.6: Transformer Architecture — Source: *Vaswani et al. (2017)*

Cheng et al. (2016).

The mathematical equation for self-attention can be given by:

$$Attention(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})$$

From the above equation, we can see that attention is a (scaled) dot product. Dot product of Query (Q) is computed with all the keys (K), each having a dimension d$_k$, as shown in the equation. The reciprocal of square root of this dimension helps scale the result and prevent from vanishing gradient problem (when the dot product of Q and K is large) before applying the activation function (softmax) to the output layer. We get high attention value when the Key and Query vectors are closely aligned. The Key with the highest dot product with the Query is then selected. All this phenomena is also pictorially represented in *Figure 4.8* (left). From the right part of the figure, we can see that multi-head attention as the parallel dot-product attentions, having *h* number of linear projections for Keys, Values and Queries. The result of multiple attentions, that are calculated in parallel are concatenated to produce final linear vector.

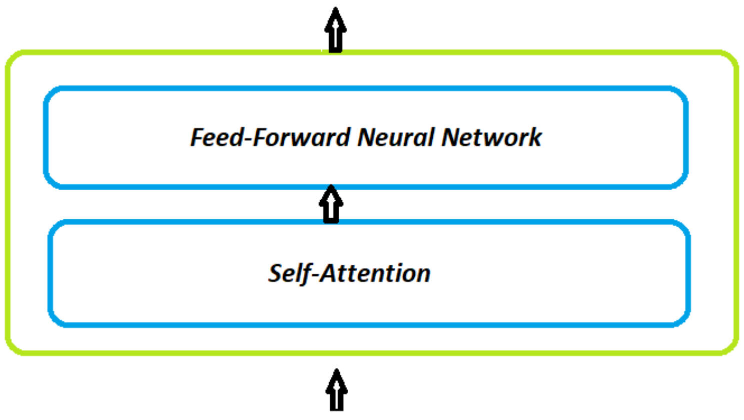
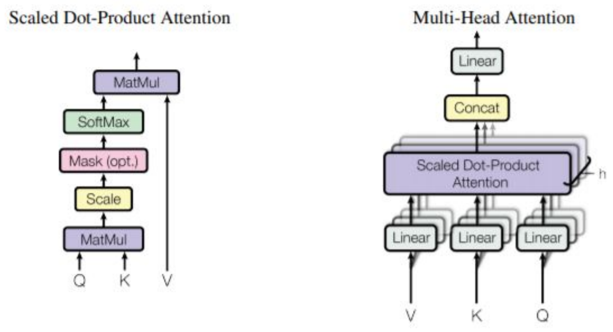


Figure 4.7: Encoder in BERT



Source — *Vaswani et al. (2017)*

Figure 4.8: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Each word (in the form of BERT-embedding) passes through self-attention, which gives a hidden state, and then through the feed-forward neural network, where learning takes place. During this phase, various weights of self-attention mechanism are also learned. In BERT architecture, the encoder is a stack of several encoders (12), each having similar architecture but different weights.
BERT pretraining has following steps:

1. Masked Language Modeling:
   15% of the words are masked, and predictions are made on them, based on context provided by non-masked words.

2. Next Sentence Prediction:
   The model receives a pair of sentences as input. The classifier learns to predict if the second sentence is the subsequent sentence in the document.

Execution time could be decreased due to parallelization. Large more resource-intense problems could be computed due to parallelization of the heads . Every encoder gives an enriched embedding as an output.

4.4.3 Self-attention

Solves the 'bad memory' of RNN and can remember longer sequences. There is a connection between the output and the hidden units of the encoder. Focus is provided by adding soft attention weights. It focuses on an essential part of the sequence (Soft alignment).

Attention is a two matrix multiplications, so its quadratic in the length of our sequence. For short sequences, it has a very good profile, and can perform 4 times faster than RNNs. As we linearly transform into a query, linearly transforming every position in the neighborhood (or input) to a key (attention heads can be paying attention to certain language structure and semantics). These linear transforms can be thought of as features. The attention can be thought of as a feature detector, which carries in itself linear transformation (projecting them in a space). For efficiency, instead of having the dimensions operate in a large space, the dimensionality of all the heads are reduced (in comparison to RNNs and CNNs), and operated in parallel. The number of softmax operation is increased as each head has its own softmax operation. Having more heads, we can also simulate CNNs, but with a lots of parameters, *Vaswani et al. (2017).*

4.4.4 Multi-headed self-attention

1. It extends the model's ability, hence can focus on several words. The current word itself dominates the output Z (output of self-attention), but it focuses on several words.

2. Each word gets represented as several different embeddings, as output by each attention head in the transformer (Z). Each of these sets is used to project the input embeddings into a different representation subspace. These Zs could be concatenated finally.

BERT does not consider the decoder side of the original encoder-decoder Transformer model. We have encoders only. One benefit of the Transformer is the use of parallelization due to use of feed-forward neural network. BERT (small model) has 12 heads, each having length 64 gives us 768 dimensional self-attention output.

After the input words are processed by self-attention, it passes through Feed-Forward Neural Network or Dense Layer, before it can be fed to another encoder.

The order of the words is taken into account by the use of Positional encoding. These are the vectors added to each input embedding. The model learns a specific pattern, hence the position of each word and the distance between different words in a sequence can be determined.

BERT will have a subword for every character in the alphabet. We can see (S##Var) marks on subwords except for the first subword (pad), which is a token. There are embedded vectors like CLS (reserved token for the start of the sequence), SEP (separate segment/sentence), and MASK.

The BERT I have been using is small-uncased and uses 12 BERT layers, and on the top, there is a single classification layer and load all the pre-trained weights from BERT layers. We can configure if we can use the class output from all hidden states. There are 109 million parameters in 12 layers of the BERT model, and it takes 417 MB. All the weights are pre-trained. The weights for the classification layer in the output are randomly initialized.

The model learns the context of each word before and after it. BERT is also trained in high data (around 3.3 billion word corpus), with the majority from Wikipedia and about 0.8 billion from BooksCorpus.

There are 26 outputs (the register categories in my case), 768 weights each (20K weights on the output layer, and 110M behind that in all these BERT layers). The fine-tuning phase has following steps:

The Training loop:

The first token ([CLS]) of the last layer of BERT encoder is used as feature vector for the document, and it further passes to the softmax to get the output probability distribution over the classes. In this training process, the weight of already pretrained BERT model is optimized (fine-tuned) based on our dataset - CORE corpus. For this experiment, the best result was found with 4 epoch and 16 batches. The detailed process for training phase is discussed in *subsection 4.4.5*

The Evaluation loop:

Here we measure the evaluation accuracy. When there was overfitting, regularization using dropout technique was applied. After looking at the Training curve vs Validation curve, we can decide if we need some steps to control the overfitting or determine if the training on the data is done properly.

4.4.5 Steps followed for BERT Training

1. *Shuffling Dataset*

   Shuffling dataset increases the variability within batches of data. It reduces the variance, and overfitting can be prevented.

2. *Running BERT*

   Running Classifier To run BERT first, we need to download BERT pretrained model and Embeddings, then load the BERT vocabulary.

   Optimization

   For the optimization, the parameters used are as per recommended by the authors: *viz. Batch size: 16 or 32, Optimizer - Adam, Number of epochs 2,3 or 4*

   Tokenization

   I have downloaded the uncased lowercase version of BERT *bert_uncased_L-12_H-768_A-12/1*. Before tokenization, the text is normalized by converting the whitespaces characters to spaces and using the lower cases. Then afterward, CLS and SEP tokens are added to the beginning and end of a sentence. Breaking into word-pieces like: 'slump', '##s' , 'marius', 'k', '##lo', '##pper', '##s' (Random example). Finally, words in the text are mapped to indexes using BERT's vocabulary.

4.4.6 Implementing BERT

The types of classification that I have followed are:

Addressing Imbalance in Dataest

As the data is highly imbalanced, I have tried to balance all the classs.When there is such a high imbalance in data, the result is mostly made in favor of the majority classes. Thus the minority classes are considered less important by the classifier. For this purpose, I have tried to balance it all with a count of 802 (I chose this threshold, which is the number of one of the labels). Also, classes with higher frequency were undersampled to the same count .
Random Undersampling[1] is used for majority classes. It randomly undersamples (or selects) examples in the majority classes.
Random Oversampling[2] is used in the minority class. It randomly duplicates or synthesizes examples of the minority classes.

Imblanced Dataset

In order to compare the accuracy result without handling the imbalance, the experiment is carried out with the original label distribution.

Hence, the experiment with BERT has been carried out. BERT, unlike RNN and LSTM, can parallelize the input data. All words' embeddings are determined simultaneously. Also, Transformer is faster than RNN and ,unlike LSTM, deeply bidirectional.

[1]https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html

[2]https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html

Transformers also have the property of solving exploding/vanishing gradient like LSTM, with less complicated architecture. Self-attention helps in getting the proper context of the word (By using multiple heads). Positional encodings are required as transformer archtecture doesn't have a sequential architecture as in RNN. The normalization layers reduce the number of steps to optimize the network. Hence the transformers is fast.

4.5 XLNet

4.5.1 Overview

The transformer was originally built for translation tasks. TransformerXL was developed for Language modeling, which yielded a lot of improvements in many tasks, *Dai et al. (2019).* XLNet is an autoregressive pre-training for language understanding, which is based on Transformer-XL. The two significant modifications made from Transformer to Transformer-XL: Memory between segments and Change from Absolute to Relative Positional Encoding (to maintain positional information of each token), which enable state reuse without causing temporal confusion (confusion in a given context). The hidden state of first segment is cached in each layer's memory so that attention is updated accordingly. So, reuse of memory for each segment is allowed. Hence much information from the history can be utilized, and context level fragmentation can be avoided.

BERT is limited to a small number of sentences. Also, there is missing dependency due to the *[MASK]* tokens, which leads to fine-tune discrepancy. BERT used Autoencoding language modeling. It employs Masked Language Modeling or it rebuild original data from the corrupted input. BERT has a limitation that words cannot be dependent on distant words, and predicted tokens are independent of each other. *Yang et al. (2020).*
For example in the following case:
I am doing NLP thesis. With 40 % masking, it can be represented as:
I [MASK] doing [MASK] thesis. It tries to predict both the [MASK] tokens using the sen-

tence 'I [MASK] doing [MASK] thesis.' So only 40% of tokens are masked. In the paper of BERT, it is stated that 10% of the masked tokens will be replaced by themselves. The architecture of XLNet is quite similar to that of BERT. There is a different pretraining procedure than BERT, however. Until Transformers, autoregressive language modeling had been used. Autoregression means regression of a variable against itself. Hence the subsequent or the next token is dependent on the previous tokens. XLNet combines the best of both, autoencoder and autoregression model. In pre-train phase, it focuses on permutation language modeling, where it trains the model on all possible permutation of words in a sentence. During this process, each position learns the information from the surrounding context from all the positions. Hence, it captures the bi-directional context of the given text sequence.

Specifically, given a text sequence x = (x1,..., xT), auto-regressive language modeling factorizes the likelihood into a forward product

$$p(x) = \prod_{t=1}^{T} p(x_t|x_{<t}) \tag{4.1}$$

or a backward one

$$p(x) = \prod_{t=T}^{1} p(x_t|x_{>t}) \tag{4.2}$$

According to the equation, each word is predicted from the word before it or after it. XLNet considers all possible orderings. For, e.g., if the sample comes: 'New York is a city.', BERT has limitations as BERT neglects dependency between the masked positions, whereas XLNet solves it.

$$J_{XLNET} = \log \mathrm{P}(\text{New} \mid \text{ is a city }) + \log \mathrm{P}(\text{ York } \mid \text{New, is a city }) \tag{4.3}$$

$$J_{BERT} = \log \mathrm{P}(\text{New} \mid \text{ is a city }) + \log \mathrm{P}(\text{ York } \mid \text{ is a city }) \tag{4.4}$$

Source — *Yang et al. (2020)*

We can see *Equation (4.3)* and *Equation (4.4)* applied in case of *Equation (4.5)*

4.5.2 Working of XLNet

BERT captures bidirectional context using MASK token and parallels independent predictions. On the other hand, XLNet uses *permutation language modeling* as done by *Uria et al. (2016).* It uses random ordering of tokens and not sequential as in traditional models.

During the pretraining phase, all the tokens are used in random order. For a sequence containing n tokens, there are n! possible factorization possible. It doesn't use fixed left-right or right-left, but instead log-likelihood over all possible permutations of the sequence is carried out.

In addition, XLNet uses two critical ideas from TransformerXL: Relative Positional Embeddings (to track position of each segment) and the Segment Recurrence mechanism (hidden state of first segment is allowed to update attention in each layer, which allows reuse of memory for each segment).

> Transformer-XL manages to model dependency that is about 80% longer than recurrent networks and 450% longer than Transformer. Moreover, Transformer-XL is up to 1,800+ times faster than vanilla Transformer during evaluation.

(Source — *Dai* et al. (2019)*)

XLNet uses the idea of *Permutation Language Modeling* as in

XLNet works smart by introducing *Two Stream Self-Attention* (Probability is conditioned not only by context token indices but also the index of the token whose probability is taken). This is also of two types:

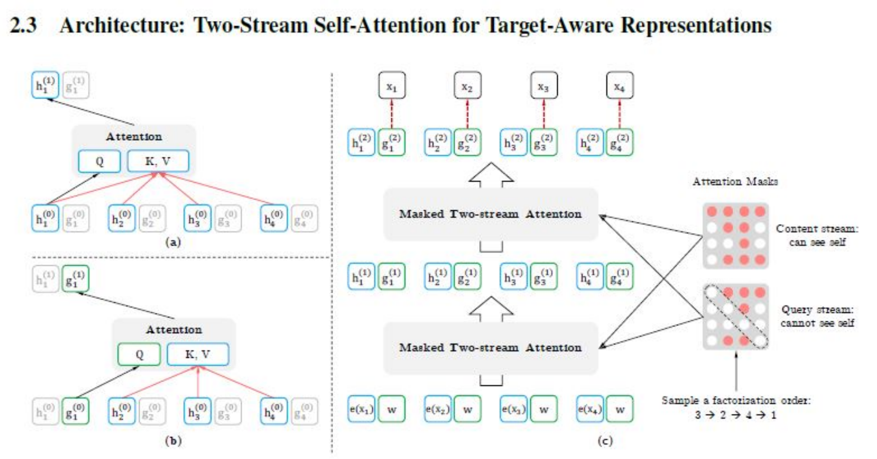

2.3 Architecture: Two-Stream Self-Attention for Target-Aware Representations
Attention
Q
K, V
(a)
(b)
(c)
Masked Two-stream Attention
Attention Masks
Content stream: can see self
Query stream: cannot see self
Sample a factorization order: 3 → 2 → 4 → 1

Figure 4.9: XLNet working

(Source — *yang2020xlnet*)

1. Content Stream: Just as in self-attention, the content and the positional information of the token to be predicted is examined

2. Query Stream: It replaces the input [MASK] token with query representation. Hence, it can predict target token using context and the positional information (and not the actual content) of the target .

Figure 4.9 explains the working of XLNet. The left corner shows how content and query representation are calculated. The content mask (h) and query mask (g) are both matrices. The input sentence can be permuted to different factorization orders using the different attention masks.

To conclude, XLNET has following great features:

1. It is Autoregressive model, which has reparameterization of Transformer XL to avoid ambiguity arose due to arbitrary factorization order (XLNET considers permutations of the factorization order)

2. It is order aware with positional encodings

3. There is segment-recurrence and relative encoding scheme of Transformer XL in

pretraining stage. Thus, it addresses task involving longer sequences.

4.5.3 The Experiment

The Parameters that I used are as recommended by the authors of the work *Yang et al. (2020) viz. Learning rate 2e-5 and 3e-5, optimizer Adam epsilon, weight decay 0.01*. Higher batch sizes and epochs resulted in overfitting, so the maximum batch size used was 16, and maximum epoch size was 4 in both the models (for imbalanced type). The result and analysis for both balanced and imbalanced dataset are in Result and Analysis section.

5 Results and Analysis

To recap, for fastText, the model was provided by Facebook and is a pretrained model. The dimension of embedding is 300 and vocabulary of 1M words.

BERT, on the other hand which uses word-piece model is a model developed by Google. It is trained on Wikipedia corpora as well. Also, BookCorpus corpus are used. It is pretty massive with 12 stacked encoders and 24 attention heads. The vocabulary limit size of the tokenizer is 30K, base number of transformer blocks is 12 (for large its 24), hidden layer size is 768 and attention heads is 12.

BERT is different from XLNet as it does not permute the tokens; instead, it depends on the network's computational dependency. XLNet base-cased embedding is of size 768, maximum. Both the BERT and XLNet support *Masked Word Prediction* and *Sequence Classification* but BERT has the limitation. It can be used only for a small number of sentences. So XLNet was developed to address this (*XL stands for extra long*). Also, BERT suffers from fine-tuning discrepancy due to masked tokens. BERT is an auto-encoder-based model, whereas XLNet is an auto-regressive model and is aware of the positional encoding.

5.1 Comparing the Three methods

Comparison: Metrics - Accuracy, Micro Average F-score (not necessarily both) . Accuracy is a good choice for Balanced data (when we are interested in True Positives and True Negatives). F-1 score is more relevant when the data is imbalanced.

F-Score gives the harmonic mean of precison and recall, and F1-score gives the best f-score.For this task, which is multi-class, confusion matrix is a better way of assessing performance. Balanced dataset was made by random upsampling (for minority classes) and random oversampling (for the majority classes). The results of these three can be directly compared to each other, and we can summarize all these.

Then these three methods *Fasttext, BERT and XLNet* make a reasonable package. Fasttext being a shallow neural network model, BERT a Autoencoder model and XLNet being a Autoregressive model.

5.1.1 Result on FastText:

I have tried three values of epochs *10,25&50* and learning rate as well as n-grams.

We can see from the *Table 5.1*, with an epoch of 10, the F1-Score was higher . On increasing it to 25, it decreased a bit. With an epoch of 50, however, the accuracy observed was the highest - 0.694.

With a learning rate of 0.1, the F-Score was low. With the learning rate of 0.5 and 1.0, though, the accuracy was almost the same.

With a learning rate of 1.0 and epoch 25, the accuracy was 0.688.

Now I have fixed learning rate 1.0, epoch 25, and vary wordNGram as 2,3 and 5. The F-Score obtained was almost the same (around 0.68 F-score and loss was 0.25, 0.27, and 0.42, respectively). All the results can be seen in the *Table 5.1*

Using Hierarchical Softmax, with parameters (not in the table):

$$-lr\ 1.0 - epoch\ 25 - wordNgrams\ e2 - bucket\ 200000 - dim\ 50 - loss\ hs$$

Thus, using hierarchical softmax, result obtained using above parameters is :

Loss = 0.416960, accuracy = 0.646

The best obtained score, using fastText, was 69$\%$. Whereas, by using the autotune validation parameter, the result obtained was 64$\%$.

Results with various parameters using fastText Model				
Epochs	Learning Rate	Word n-gram	Loss	F-Score
10			1.737522	0.557
25			2.058319	0.457
50			0.869470	0.694
	0.1		2.049748	0.462
	0.5		1.351652	0.659
	1.0		1.152584	0.69
25	1		1.143183	0.688
25	1	2	0.368338	0.694
25	1	2	0.500568	0.666
50	1	2	0.185064	0.691

Table 5.1: Results of Using fastText

5.1.2 Result on BERT:

The experiments were carried using two deep learning libraries *Pytorch* and *Tensorflow (Keras).*

Imbalanced Data:

The best accuracy score obtained, keeping in consideration that model was not overfit, was 77% . It is practical to compare F1-Score for the imbalanced class only. The result obtained was better with higher epoch and batch size. We can see the result of classification in *Table 5.3*. Classes having higher support (having higher instances) have shown better F-Scores. From the *Figure 5.1*, if we observe the diagonal, which is the labels correctly predicted. In most cases, the majority of the instances have been predicted correctly. The results obtained in table *Table 5.2* was from Bert Small Uncased pretrained model.

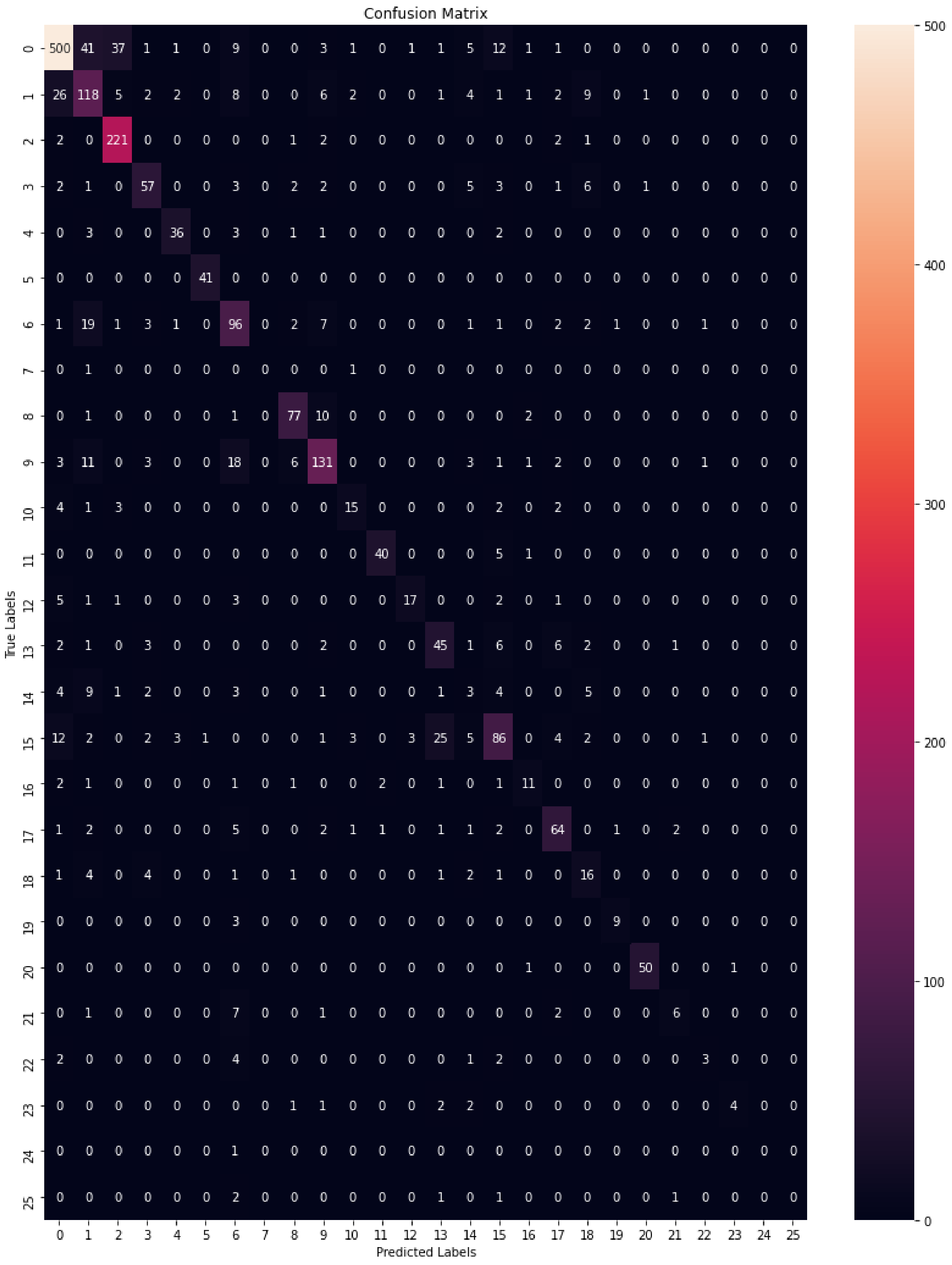
The plot of Training Set Accuracy vs Validation Set Accuracy is as in figure *5.2*. There seems to be some overfitting. So after adding a dropout of 0.5, the result obtained is as in *Figure 5.3*.

Balanced Dataset:

For balanced dataset, accuracy alone is sufficient to consider as evalutation metric. The highest obtained accuracy was 74%.

Batch size	Epochs	Warmup Proportion	Learning Rate	Epsilon	Weight Decay	Accuracy
8	3	0.1	2e-5	1e-6	1e-8	76%
16	4	time-dependent	2e-5	1e-8	0.1	77%
16	4	time-dependent	3e-5	1e-6	0.1	75%

Table 5.2: Different Parameters Used For BERT (Imbalanced Data) model Training



True \ Predicted	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	500	41	37	1	1	0	9	0	0	3	1	0	1	1	5	12	1	1	0	0	0	0	0	0	0	0
1	26	118	5	2	2	0	8	0	0	6	2	0	0	1	4	1	1	2	9	0	1	0	0	0	0	0
2	2	0	221	0	0	0	0	0	1	2	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0
3	2	1	0	57	0	0	3	0	2	2	0	0	0	0	5	3	0	1	6	0	1	0	0	0	0	0
4	0	3	0	0	36	0	3	0	1	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	41	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	19	1	3	1	0	96	0	2	7	0	0	0	0	1	1	0	2	2	1	0	0	1	0	0	0
7	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	1	0	0	0	0	1	0	77	10	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
9	3	11	0	3	0	0	18	0	6	131	0	0	0	0	3	1	1	2	0	0	0	0	1	0	0	0
10	4	1	3	0	0	0	0	0	0	0	15	0	0	0	0	2	0	2	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	40	0	0	0	5	1	0	0	0	0	0	0	0	0	0
12	5	1	1	0	0	0	3	0	0	0	0	0	17	0	0	2	0	1	0	0	0	0	0	0	0	0
13	2	1	0	3	0	0	0	0	0	2	0	0	0	45	1	6	0	6	2	0	0	1	0	0	0	0
14	4	9	1	2	0	0	3	0	0	1	0	0	0	1	3	4	0	0	5	0	0	0	0	0	0	0
15	12	2	0	2	3	1	0	0	0	1	3	0	3	25	5	86	0	4	2	0	0	0	1	0	0	0
16	2	1	0	0	0	0	1	0	1	0	0	2	0	1	0	1	11	0	0	0	0	0	0	0	0	0
17	1	2	0	0	0	0	5	0	0	2	1	1	0	1	1	2	0	64	0	1	0	2	0	0	0	0
18	1	4	0	4	0	0	1	0	1	0	0	0	0	1	2	1	0	0	16	0	0	0	0	0	0	0
19	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	50	0	0	1	0	0
21	0	1	0	0	0	0	7	0	0	1	0	0	0	0	0	0	0	2	0	0	0	6	0	0	0	0
22	2	0	0	0	0	0	4	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	3	0	0	0
23	0	0	0	0	0	0	0	0	1	1	0	0	0	2	2	0	0	0	0	0	0	0	0	4	0	0
24	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	2	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0

Figure 5.1: Confusion Matrix, BERT (Imbalanced Class) with 4 epochs and 16 Batch Size

	Precision	Recall	F-1Score	Support
0	0.41	0.42	0.43	31
1	0.88	0.73	0.80	180
2	0.85	0.70	0.78	30
3	0.60	0.71	0.64	69
4	0.64	0.67	0.66	150
5	0.89	0.87	0.88	46
6	0.60	0.40	0.41	10
7	0.00	0.00	0.00	2
8	0.82	0.65	0.73	20
9	0.79	0.77	0.79	83
10	0.14	0.18	0.14	33
11	0.89	0.78	0.83	27
12	0.89	0.83	0.86	614
13	0.63	0.65	0.63	188
14	0.62	0.68	0.63	138
15	1.00	0.18	0.28	5
16	0.80	0.86	0.83	91
17	0.95	0.98	0.97	41
18	0.73	0.92	0.75	12
19	0.91	0.89	0.90	46
20	0.81	0.86	0.80	83
21	0.93	0.98	0.96	52
22	0.83	0.97	0.89	229
23	0.89	0.77	0.83	17
24	0.30	0.35	0.29	12
25	0.00	0.00	0.00	1
Accuracy			0.77	2210

Table 5.3: BERT Classification Report (Imbalanced Class) with 4 epochs and 16 Batch-Size

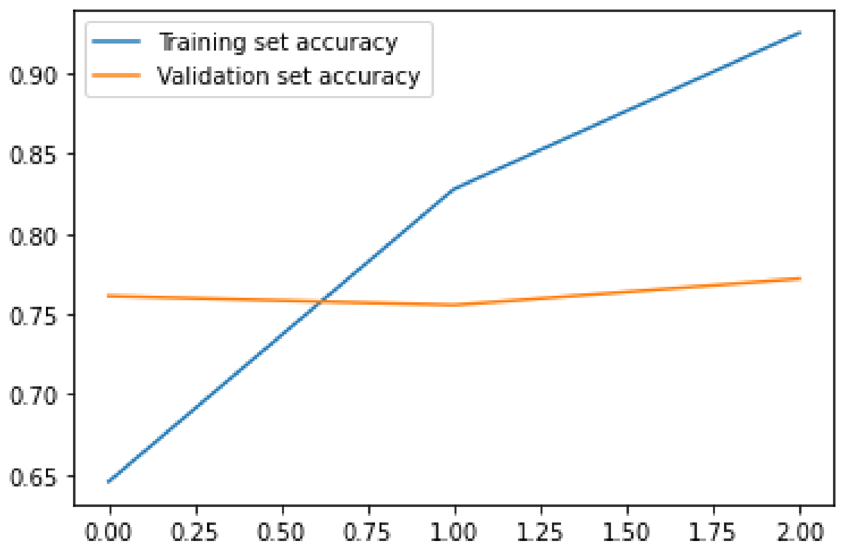


Figure 5.2: Accuracy Plot, BERT with 4 epochs and 16 Batch

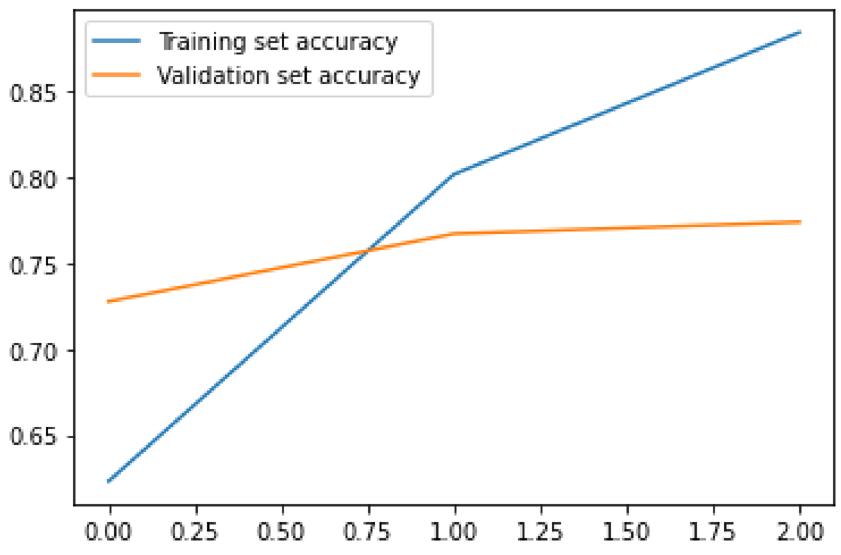


Figure 5.3: Accuracy Plot for *Figure 5.2* with a Dropout of 0.5

Batch size	Epochs	Epsilon	Learning Rate	Weight Decay	Accuracy
16	3	1e-12	2e-5	0.01	77
16	4	1e-12	2e-5	0.01	76
8	4	1e-12	2e-5	0.01	77

Table 5.4: Parameters Used in XLNet Classification(Imbalanced Dataset)

5.1.3 Result on XLNet:

For Imbalanced Dataset

With the default parameters (as in the XLNet paper), the highest accuracy obtained was 77%, using 4 epochs and 8 or 16 batches (for batch 16, dropout was used for prevent overfitting). With epoch more than 4 and batches greater than 8, there was overfitting. In BERT, highest accuracy was obtained with 4 epoch and 16 batches.

We can also see the per class F1-score were quite good for classes having good number of samples. Whereas, classes having lesser samples have very less F-Score, some even have 0.

For Balanced Dataset:

The best accuracy obtained for the balanced dataset was 74% With the same parameters as used above, the accuracy obtained was 73%.

	Precision	Recall	F-1Score	Support
0	0.33	0.48	0.39	29
1	0.83	0.75	0.81	154
2	0.60	0.50	0.55	30
3	0.52	0.75	0.61	58
4	0.65	0.58	0.63	134
5	0.84	0.90	0.89	40
6	1.00	0.40	0.57	9
7	0.00	0.00	0.00	2
8	0.79	0.56	0.67	17
9	0.75	0.71	0.74	74
10	0.21	0.15	0.18	30
11	0.74	0.52	0.61	25
12	0.89	0.81	0.85	705
13	0.56	0.66	0.60	195
14	0.63	0.59	0.61	150
15	1.00	0.20	0.33	5
16	0.84	0.87	0.88	72
17	0.87	0.98	0.92	39
18	0.55	0.92	0.69	12
19	0.82	0.77	0.79	40
20	0.75	0.81	0.78	112
21	0.96	0.90	0.93	48
22	0.80	0.91	0.89	188
23	0.69	0.53	0.60	15
24	0.25	0.25	0.25	12
25	0.00	0.00	0.00	1
Accuracy			0.77	2210

Table 5.5: XLNet model Classification report for Imbalanced Dataset with 16 Batches and 4 Epoches

5.2 Comparing the three models:

The highest obtained accuracy for fastText was 69.1%, using 50 epochs, learning rate of 1 and word nGram of 2. For BERT, the highest obtained accuracy was 77% using 4 epochs and 8 or 16 batches, with a learning rate of 2e-5. For XLNet, epoch 3 and 4 were ideal. Ideal batch size was 4 or 8. The highest obtained accuracy in XLNet was also 77%.

Talking about the balanced class, both the models had similar performance.

	Balanced	Imbalanced
fastText	-	69.1%
BERT	74%	77%
XLNet	73%	77%

Table 5.6: Comparing Highest Accuracy (Balanced Data) and F-Score (Imbalanced Data) in the Three Algorithms

5.3 Conclusion

After knowing that XLNet has so many advantages over BERT, I thought XLNet would give a better result. However, the performance of BERT and XLNet was similar for this dataset. The XLNet model was overfit if the epochs was exceeded more than 4. Despite the vast computation capability of these massive models, the performance metric of fastText was not so much lower than that of BERT or XLNet.

While working with deep learning models, I concluded that dealing with different types of data with proper parameters to measure proper evaluation metric should be done. The proper way of working with Deep Learning models is also to check if the model will work for new data, hence *overfitting* should always be in the background of mind while training the model.

Following steps were considered while working with the deep learning models :

Shuffling - the Dataset. This was needed so that data from all the classes get distributed to the training and validation set.

Checking the Training Accuracy vs. Validation Accuracy - to check if the model has overfitted. To reduce overfitting, I applied *dropout*. Dropout skips some portion of the hidden neurons during the training phase and controls overfitting. Adding dropout seemed to show some improvement, which can be seen from the figure **??**, as in the case of BERT.

Tune parameters - Further tuning of parameters to see if better accuracy can be obtained. For which I tried various batches, epochs, and learning parameters.

> **Hyperparameters vs Parameters** - Hyperparameters are specific to models, whereas parameters are something to be learnt by the model, which can be varied to improve the performance of the model.

Maximum Sequence Length - BERT and XLNet have parameters *max_seq_len* and *max_seq* respectively to take into consideration the maximum number of tokens. When a sequence length of 256 was used, the accuracy was higher than the sequence length of 512. However, since most of the documents are very lengthy, a sequence length of 512, which is the highest length allowed, would be a better choice for training the model since it can take surrounding contexts properly.

Balancing the Classes - It can be seen from the figure about data distribution in the Introduction section, the classes are highly imbalanced. The classes were balanced using

undersampling and oversampling - In my case, the imbalanced class dataset showed a bit higher accuracy than the balanced class. Nevertheless, the model trained in the balanced dataset predicts all the classes properly, whereas model trained in imbalanced class dataset, mostly majority class was predicted well. Hence, even though the accuracy for the balanced class was low, all the classes were generalized properly.

6 Conclusion and Findings

In this thesis work, I started with annotated data (A web corpus) given by the Turku NLP group. The task of register classification was addressed by three algorithms: FastText, BERT, and XLNet. Register can be referred to as a corpus following a particular situational nuance.

6.1 Conclusion:

To conclude, the task was a supervised machine learning problem with text as the data, alongwith the corresponding labels. This was a practical real-life problem. While working with this, a lot of things have to be taken into account like how to work with imbalanced data, which accuracy metrics to use, which activation functions to use, which algorithms to use for the best performance.

As the main goal of the task was to evaluate the SOTA Transformer models for this classification, so based on the experiment it could be figured out that BERT outperformed the other algorithms. Even though fasText has significantly less parameters and uses less computational resources, it gave quite acceptable result. XLNet, which uses *Permutation* method, also gave similar result to BERT in this experiment, though BERT gave slightly higher result. Hence, BERT was the overall winner for this dataset.

In order to get better results, the steps carried were:

1. Train over fewer/higher epochs

2. Checking overfitting by process like *dropout*

3. Optimization of other hyperparameters like: *weight decay, learning rate, warmup steps*

Also, one more way of optimizing would be adding more layers in the pre-trained and Change the number of nodes in the layers.

6.2 Future Extensions:

Further, an extension of the work can be carried out by combining the embeddings of these two deep learning methods, viz. BERT and XLNet, as an example. Also, there has been more recent work like *Longformer Beltagy et al. (2020)* to work for long sequence. I have worked with only maximum sequence length allowed by the BERT and XLNet embeddings. Also, *Sun et al. (2020)* has dealt with ways to deal with longer sequence and fine-tune BERT embedding.

Furthermore, Multilabel classification would be a significant problem to be solved utilizing these deep learning models. Often common language, grammatical and lexical features sometimes can confuse various registers classification. Hence various registers can have some characteristics and situations in common. Thus, multilabel classification would be a good thing to try in such a scenario.

Bibliography

M. Abdullah and M. G. I. A. Zamil. The effectiveness of classification on information retrieval system (case study), 2018.

C. Allen and T. Hospedales. Analogies explained: Towards understanding word embeddings, 2019.

S. E. Argamon. Computational register analysis and synthesis, 2019.

G. Bebis and M. Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13(4): 27–31, 1994.

I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer, 2020.

D. Biber. *Variation across Speech and Writing*. Cambridge University Press, 1988. doi: 10.1017/CBO9780511621024.

D. Biber and J. Egbert. Using grammatical features for automatic register identification in an unrestricted corpus of documents from the open web. 2016.

D. Biber and J. Egbert. *Register Variation Online*. 08 2018. ISBN 9781107122161. doi: 10.1017/9781316388228.

D. E. Biber. Conversation text types : A multi-dimensional analysis. 2004.

S. Bird. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*.

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information, 2017.

R. Briones. Textual analysis through systemic functional linguistics. *Journal of English Language Teaching and Linguistics*, 1, 08 2016. doi: 10.21462/jeltl.v1i2.27.

J. Cheng, L. Dong, and M. Lapata. Long short-term memory-networks for machine reading, 2016.

I. Clarke and J. Grieve. Dimensions of abusive language on twitter. In *ALW@ACL*, 2017.

Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context, 2019.

Z. Dai*, Z. Yang*, Y. Yang, W. W. Cohen, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-XL: Language modeling with longer-term dependency, 2019. URL https://openreview.net/forum?id=HJePno0cYm.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv e-prints*, art. arXiv:1810.04805, Oct. 2018.

J. Egbert, D. Biber, and M. Davies. Developing a bottom-up, user-based method of web register classification. *Journal of the Association for Information Science and Technology*, 66, 2015.

Y. Goldberg. *Neural Network Methods in Natural Language Processing (Synthesis Lectures on Human Language Technologies)*, page 1. Graeme Hirst, 2017.

D. Govindaraj. Can boosting with svm as week learners help? 04 2016.

S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith. Don't stop pretraining: Adapt language models to domains and tasks, 2020.

M. Herke-Couchman and W. Canzhong. Stylistic features as meaning representation: Text as phase portrait. In *AAAI Technical Report*, 2004.

A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification, 2016.

A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. *ArXiv*, abs/1607.01759, 2017.

J. Karlgren. Stylistic variation in an information retrieval experiment. *CoRR*, cmp-lg/9608003, 1996. URL `http://arxiv.org/abs/cmp-lg/9608003`.

G. Kim, K. Fukui, and H. Shimodaira. Segmentation-free compositional n-gram embedding, 2019.

Y. Kim. Convolutional neural networks for sentence classification, 2014.

Q. V. Le and T. Mikolov. Distributed representations of sentences and documents, 2014.

X. Li, L. Wang, and E. Sung. Adaboost with svm-based component classifiers. *Engineering Applications of Artificial Intelligence*, 21:785–795, 08 2008a. doi: 10.1016/j.engappai.2007.07.001.

X. Li, L. Wang, and E. Sung. Adaboost with svm-based component classifiers. *Engineering Applications of Artificial Intelligence*, 21:785–795, 08 2008b. doi: 10.1016/j.engappai.2007.07.001.

Q. Liu, J. Wang, D. Zhang, Y. Yang, and W. Naiyao. Text features extraction based on tf-idf associating semantic. pages 2338–2343, 12 2018. doi: 10.1109/CompComm.2018.8780663.

M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation, 2015.

S. Lyu and J. Liu. Combine convolution with recurrent networks for text classification, 2020.

C. D. Manning, P. Raghavan, and H. Schütze. Introduction to information retrieval. Cambridge, England, 2008. Cambridge University Press.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space, 2013a.

T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, June 2013b. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/N13-1090`.

S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao. Deep learning based text classification: A comprehensive review, 2020.

M. Radovanovic, M. Ivanovic, and Z. Budimac. Text categorization and sorting of web search results. *Computing and Informatics*, 28:861–893, 01 2009.

X. Rong. word2vec parameter learning explained, 2016.

Z. Sadeghi, J. Mcclelland, and P. Hoffman. You shall know an object by the company it keeps: An investigation of semantic representations derived from object co-occurrence in visual scenes. *Neuropsychologia*, 8, 09 2014. doi: 10.1016/j.neuropsychologia.2014.08.031.

S. Schuster, W. Zhu, and Y. Cheng. Predicting tags for stackoverflow questions. 2013.

R. Sennrich, B. Haddow, and A. Birch. Neural Machine Translation of Rare Words with Subword Units. *arXiv e-prints*, art. arXiv:1508.07909, Aug. 2015.

C. Sun, X. Qiu, Y. Xu, and X. Huang. How to fine-tune bert for text classification?, 2020.

E. Teich and P. Fankhauser. Exploring a corpus of scientific texts using data mining. 2010.

B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle. Neural autoregressive distribution estimation, 2016.

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

V. Vidulin, M. Luštrek, and M. Gams. Multi-label approaches to web genre identification.

H. Xu, Q. Liu, D. Xiong, and J. van Genabith. Transformer with depth-wise lstm, 2020.

Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2020.