**Candidate Number: 2110821**

# Comparing Machine Learning and Deep Learning models on Multi-Class Text Classification

Submitted for the Degree of Master of Science in

**Artificial Intelligence**

at Royal Holloway University of London.


Department of Computer Science

Royal Holloway University of London

Egham, Surrey TW20 0EX, UK

December 06, 2021.

# Table of Contents

## Table of Figures

## Abstract

This dissertation aims to compare and analyse how different machine learning classifiers with different feature extraction methods and deep learning architectures perform on text classification. Multi-Class classification is a widely used technique in machine learning for categorizing items such as music or news articles that may belong to multiple classes[1]. It will also analyse the effect of using different word embedding techniques such as Count Vector, TF-IDF, Word2Vec & GloVe are compared using different classifiers such as Naive Bayes, SVM, Bagging, Boosting, CNN & RNN's for multi-label classification. This report will address the error analysis and hyper-parameter tuning carried out on different classifiers and architectures. Finally, point outs models/architectures that performs best on the multi-class text classification task. The findings show that the word embedding technique and text pre-processing used has a significant influence on the classifications that arise, and TF-IDF and GloVe word embedding in combination with machine learning and deep learning algorithms provided better results. Results also indicate that deep learning architectures perform better than the Machine learning classifiers. All the comparison between the machine learning and deep learning models is carried out using precision, recall, and the F1 scores.

# Chapter 1

# Introduction

The quantity of digital material available on the Internet and in organisations across all industries is continually increasing and making the greatest use of the large amount of data available can be difficult. One solution to this challenge is to categorise data into separate groups, which provides a better overview. It would be extremely beneficial for organisations to avoid having to do this categorising method manually. Instead, machine learning techniques may be used to automate the process. Such techniques usually require distinguishing features of an object in order to be able to classify it.

There are multiple methods for extracting features from various types of data such as image, text, audio and video, and different approaches may be appropriate in different scenarios. For example, when categorising photographs of apples and oranges, a simple extraction approach may be to use the image's average pixel values. If the average pixel value is close to orange, the picture is labelled as an orange; if it is close to green, it is labelled as an apple. This technique, however, would most certainly categorise a tiger as an orange because it does not examine any other characteristic other than colour. This might possibly be prevented by using a more appropriate extraction approach. Instead of merely using the average pixel values, one may also consider the object's form, size, and texture.

Similarly features from text data can be extracted using different techniques, in this report I will implement different feature extraction techniques and use it along with different machine learning and deep learning models. I will evaluate and compare the different model's performances on the news article text classification task.

## 1.1 Problem Statement

Text can be a very rich source of information, but due to its unstructured nature, extracting information and insights can be difficult and a time-consuming process[2]. Sorting text data is becoming easier because to developments in natural language processing and machine learning, both of which lie under the broad umbrella of artificial intelligence. Text analysis (TA) is a machine learning technique that extracts useful information from unstructured text input automatically. Text analysis tools are used by businesses to quickly ingest web data and documents and turn them into actionable insights.

**Methods and Techniques for Text Analysis**

**Text Classification**

The technique of applying predefined tags or categories to unstructured text is known as text classification. It's considered one of the most useful natural language processing techniques because, it can organise, arrange, and categorise pretty much any type of text to offer relevant data and solve problems, it's considered one of the most useful natural language processing approaches.

**Entity Extraction**

Named-entity recognition (NER) is a part of information extraction task which is used for identifying entities present in the text into pre-defined categories like person names, organisations, locations, timestamps, monetary values and so on.

**Topic modelling**

Topic Modelling is an NLP technique that uses machine learning to extract the important topics that occur in a collection of documents[3]. Topic modelling is recognizing the words from the topics present in the document or the corpus of data.

**Text summarization**

Text Summarization is an NLP technique that uses machine learning algorithms to shortens longer texts and generates summaries[4]. It is a supervised method where the algorithm is trained using human labelled data using which it automatically generates summaries.

In my dissertation I will be solving the text classification problem using machine learning and deep learning techniques which I learned during my master's program. The aim is to build a text classification model that automatically classifies text into one or more pre-defined categories. Sentiment analysis, spam detection, tagging customer queries and categorization of news articles are few applications of text classification[5]. I will be using a data set that has news articles classified into 4 different categories, I will be building classifiers using machine learning and use deep learning architectures to automatically classify the text into its respective categories.

The questions that this report will attempt to answer are:

- Which of the feature extraction methods (Count Vector, TF-IDF, Word2Vec and GloVe) performs the best?
- Does advanced deep learning architectures perform better than machine learning classifier models with the best feature extraction method?
- Compare and Analysis the performance of all the machine learning and deep learning models.

## 1.2 Scope

The main goal of this dissertation is to examine the performance of machine learning and deep learning methods for multi-class classification for textual data. Three feature extraction methods are evaluated in conjunction with the machine learning and deep learning methods. It would be possible to include more machine learning and deep learning models to compare their performances, but I would be using a few machine-learning models (such as Naive Bayes, SVM, Bagging, and Boosting) and a few deep learning models (CNN, RNN, and BERT). The resulting classification model can be integrated with banking and health care and other platforms to categorize documents or contents. But it is outside the scope of this dissertation to evaluate such a system. Instead, this work is to be viewed as examining the classification models which are the foundation for such platforms.

## 1.3 Objective

The objective of this report is to look at how various algorithms and feature extraction approaches perform on multi-class classification text classification tasks. When it comes to categorization, you naturally want the best outcomes possible. As a result, it may be worthwhile to dedicate additional time to selecting a suitable feature extraction approach and determining ideal hyper-parameters that will increase the classifier's performance. This report explores which machine learning algorithms performs better for this text classification task and which feature extraction method works better with which algorithm. Hopefully, the findings of this study will be valuable in the search for optimum feature extraction approaches and algorithms to better categorise multi-class text data.

# Chapter 2

# Background

## 2.1 Importance of data

Data is one of a company's or organization's most valuable assets. Data allows us to better understand the market and personalise products to meet the needs of our customers. Data can also be used to assess a company's success and productivity. Data is critical for any organisation to make informed decisions, finding solutions to problems and be strategic in your approaches.

Data, in all its forms, is critical to making business decisions. The ability of a company to collect, analyse, process, and act on data is critical to its success. However, in today's digital world, where the amount of data generated is expanding at a fast pace and arrives in a variety of formats, organisations are finding it more difficult than ever to analyse this data.

**What is structured data?**

Structured data is usually quantitative and is stored in pre-defined models or formats (think Excel or Google Sheets)[6]. It is normally mapped to standardised columns and rows within pre-defined limits. Data entry, search, extraction, and analysis are all made easier using structured data models.

**What is unstructured data?**

Unstructured data is any data that is not structured in a pre-defined manner and is instead stored in its natural state[6]. This data is more difficult to sort through, retrieve, and analyse due to its lack of structure. This is frequently qualitative, text-heavy data or data that has been complicatedly configured. Images, text messages, and audio recordings are examples of unstructured data.

By far the most common sort of data is unstructured data. Unstructured data accounts for about 80% – 90% of enterprise data, according to some estimates, and it is rising at a rate of 55 percent – 65 percent per year. Unstructured data isn't always well-organized or easy to find, yet it's essential for receiving a complete answer to your most important business problems.

The process of turning unstructured text data into a meaningful form that can be studied is known as text analytics. Text analytics involves getting unstructured data and then organizing it to identify patterns and trends. The process aids in a variety of ways. Sentimental analysis, grouping, lexical analysis, predictive analysis, and categorization are all part of the text-analysis process[6]. For instance, classifying your content into different categories aids users to easily search and navigate within a website or application. I will be using news articles data that fall into different categories to build and compare the text classification models.

## 2.2 Data used

I will be using AG's news topic classification dataset to implement my text classification models. I will briefly mention the origin and description of the dataset below.

**Origin**

AG is a repository of over 1 million news items. ComeToMyHead has collected news articles from over 2000 news sources in over a year of operation. ComeToMyHead is a search engine for academic news that has been in operation since July 2004. The dataset is utilised by academics for research purposes in data mining (clustering, classification, and so on), information retrieval, and so on (ranking, search, etc).

**Description**

The AG's news topic classification dataset is built by selecting the four most ubiquitous topics from the original corpus. There are 30,000 training samples and 1,900 testing samples in each class. The total number of training samples is 120,000 and the size of the test data is around 7500 samples. The dataset consists of 3 columns, corresponding to class index (1 to 4), title, and description. The 4-class index of the news categories is 1 - World news, 2 - sports, 3 - Business, and 4 - Science/Technology.

## 2.3 Data Pre-Processing

Natural language processing (NLP) activities need text pre-processing. It makes language easier to understand for algorithms that use machine learning[7]. Text pre-processing is a technique for cleaning and preparing text data for use in a model's calculations. Noisy text data includes things like emoticons, punctuation, and text in several case systems. Cleaning and pre-processing data is just as critical as developing models when it comes to Natural Language Processing. Below I have mentioned the list of pre-processing steps which I have done before using the text for model building.

### 2.3.1 Lower Case & & Special character removal

Because the machine treats lower case and upper case differently, it is easy for a computer to read the words if the text is in the same case. Words like Ball and ball, for example, are processed differently by machines[7]. To avoid such issues, we must make the text in the same case, with lower case being the most preferable instance.

### 2.3.2 Stop Words

Stop words are the most frequently occurring words in a text that offer no useful information. Stop words such as they, there, this, where, and others are examples of stop words. In any

human language, there are plenty of stop words. We remove the low-level information from our text by deleting these terms, allowing us to focus more on the crucial information.

### 2.3.3 Stemming & Lemmatization

To prepare text, words, and documents for further processing, Natural Language Processing methods such as Stemming[8], and Lemmatization are used.

Lemmatization reduces word forms to linguistically acceptable lemmas, while stemming reduces word forms to stems. Below I have a attached a screen shot which shows the text before and after pre-processing.

| | Class | Description | cleaned_text |
|---|---|---|---|
| 0 | Business | AP - Both sides in an employment discrimination suit against Abercrombie amp; Fitch Co. reported agreeing to a multimillion dollar deal that would settle accusations the clothing retailer promoted whites at the expense of minorities. | ap side employment discrimination suit abercrombie amp fitch co reported agreeing multimillion dollar deal would settle accusation clothing retailer promoted white expense minority |
| 1 | Business | Continental Airlines Inc., the fifth-largest US carrier, said it will skip contributions to employee pension plans this year, taking advantage of a law enacted in April to conserve cash. | continental airline inc fifthlargest u carrier said skip contribution employee pension plan year taking advantage law enacted april conserve cash |
| 2 | Business | Russia and Belarus achieved compromise and settled the gas pricing problem, Russian President Vladimir Putin said at a joint press conference with his Belarussian counterpart Alexander Lukashenko. | russia belarus achieved compromise settled gas pricing problem russian president vladimir putin said joint press conference belarussian counterpart alexander lukashenko |
| 3 | Business | New Delhi: The United Progressive Alliance (UPA) government on Wednesday hiked foreign direct investment (FDI) limit in the civil aviation sector from 40 to 49 per cent but barred foreign airlines from holding stakes. | new delhi united progressive alliance upa government wednesday hiked foreign direct investment fdi limit civil aviation sector 40 49 per cent barred foreign airline holding stake |
| 4 | Business | Reuters - Shares of Taser International Inc.\(TASR.O) rose before the bell on Wednesday after the company\said it had received five orders totaling more than #36;741,000\for its stun guns. | reuters share taser international inctasro rose bell wednesday companysaid received five order totaling 36741000for stun gun |

*Figure 1 Text data before and after Pre-Processing*

## 2.4 Feature Extraction

The process of extracting a list of words from text data and translating them into a feature set that can be used by a classifier is known as text feature extraction.

The techniques used to turn Text into features can be referred to as "Text Vectorization" there are many feature extraction techniques but they all have the same goal: to convert text into vectors that can be fed to machine learning models in a traditional way.

I have implemented the following feature extraction methods and measured the performance of it by building a machine learning classifier using each each.

### 2.4.1 Bag of Words (BOW)

When comparing all of the different feature extraction procedures, the bag of words strategy is the most common and starightforward procedure as it builds a word presence feature set from all of an instance’s words[9]. It's called a "bag" of words since the method doesn't care how many times a word appears or in what sequence the words appear; all that matters is that the word appears in a list of terms. This method is incredibly simple and straightforward.

**Bag-of-Words working**

By counting how many times each word appears in the text, the bag-of-words model transforms it into fixed-length vectors.

We can understand this with an example. Consider that we have the following sentences:

*Vaccination is necessary.*

*vaccination is necessary and important.*

*vaccination is complicated.*

We will refer to each of the above sentences as documents. If we take out the unique words in all these sentences, the vocabulary will consist of these 6 words: {'vaccination', 'is', 'necessary', 'and', 'important, 'complicated'}.

To implement bag-of-words, we need to count the number of times each word appears in each of the documents.

| Document | vaccination | is | necessary | and | important | complicated |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 1 | 1 | 1 | 1 | 0 |
| 3 | 1 | 1 | 0 | 0 | 0 | 1 |

*Figure 2 Bag-of-words vectorization on my data*

Document 1: [1,1,1,0,0,0]

Document 2: [1,1,1,1,1,0]

Document 3: [1,1,0,0,0,1]

**Limitations of Bag-of-Words**

In the case of large texts, if we use bag-of-words to generate vectors, the vectors will be massive and will include an excessive number of null values, resulting in sparse vectors.

The bag-of-words does not convey any information regarding the meaning of the content it contains[10]. Take the following two statements into consideration: "*Vaccination is simple but tiresome*." and "*Vaccination is tedious but easy*.". Despite the fact that they have different meanings, a bag-of-words model would yield the same vectors for each of them.

### 2.4.2 TF-IDF

A problem with using a bag of words is that the words with the highest frequency become dominating in the data[9]. These words may not offer the model with much information. As a result of this issue, domain-specific terms with lower scores may be deleted or disregarded.

To remedy this issue, the frequency of the terms is rescaled by considering how often they appear in all the publications. As a result, the ratings for frequently used words are reduced across all documents.

TFIDF works by incrementing the number of times a word occurs in the document in proportion to the number of documents in which it appears[9], [10]. As a result, terms like 'this,' 'are,' and so on, which appear often in all documents are not given a high ranking. A term that appears too frequently in a few of the documents on the other hand, will be given a higher score since it might be interpreted as an indication of the document's context.

**Term Frequency**

Term frequency can be calculated by obtaining the number of times a word (i) appears in a document (j) divided by the total number of words in the document[10].

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

**Inverse Document Frequency**

The inverse document frequency is defined as the log of the total number of documents divided by the number of documents containing the phrase in the logarithmic form. It is necessary to utilise the logarithm in order to diminish the relevance of a very high IDF score[10].

$$idf(w) = \log\left(\frac{N}{df_t}\right)$$

TF-IDF is calculated by multiplying the above two values: term frequency and the inverse document frequency[10].

$$w_{i,j} = tf_{i,j} * \log\left(\frac{N}{df_t}\right)$$

We can understand the working of TFIDF using the following sentences, we will consider these sentences as documents.

Document 1: Vaccination is necessary.

Document 2: Vaccination is necessary and important.

| **Word** | **TF** | | **IDF** | **TFIDF** | |
|---|---|---|---|---|---|
| | **Doc 1** | **Doc 2** | | **Doc 1** | **Doc 2** |
| Vaccination | 1/3 | 1/5 | log (2/2) = 0 | 0 | 0 |
| is | 1/3 | 1/5 | log (2/2) = 0 | 0 | 0 |
| necessary | 1/3 | 1/5 | log (2/2) = 0 | 0 | 0 |
| and | 0/3 | 1/5 | log (2/1) = 0.3 | 0 | 0.06 |
| important | 0/3 | 1/5 | log (2/1) = 0.3 | 0 | 0.06 |

*Figure 3 TF-IDF vectorization on my text data*

The table above illustrates how the TFIDF of certain terms is zero and that of others is non-zero based on their frequency in the document and across all documents.

TFIDF's shortcoming is that this vectorization does not aid in bringing in the contextual meaning of the words because it is based solely on frequency.

### 2.4.3 Word2Vec

In Word2Vec instead of encoding words as a one-hot encoding (like count vectorizer/tf-idf vectorizer) in high-dimensional space, we encode words as dense low-dimensional vectors in such a way that related words get comparable word vectors and are mapped to nearby points.

Word embeddings are created using Word2Vec. The models built by word2vec are two-layer neural networks that are shallow. They can reproduce semantic contexts of words once they've been trained. As input, the model uses a massive corpus of text. After that, it generates a vector space with hundreds of dimensions. In the vector space, each discrete word in the corpus is assigned a corresponding vector.

The Continuous Bag of Words (CBOW) model and the Skip-Gram model are two different model architectures that Word2Vec use to build word embeddings.

**How does CBOW work?**

If you provide Word2Vec with a corpus without any label information, it will construct dense word embeddings, but Word2Vec uses an unsupervised classifier to get these embeddings from the corpus. Word2Vec is an unsupervised model.

We use a deep learning classification model in the CBOW architecture to try to predict our target word (Y) from context words (X).

Consider the example that "Deep learning models outperform other models." Context words and target words may be used in conjunction with each other. It is possible to generate pairings such as ([deep, models], learning) and so on with a context window size of two. These target words would be predicted by the deep learning model based on the context

Words.

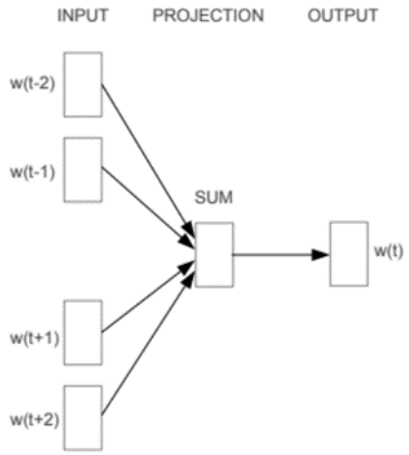
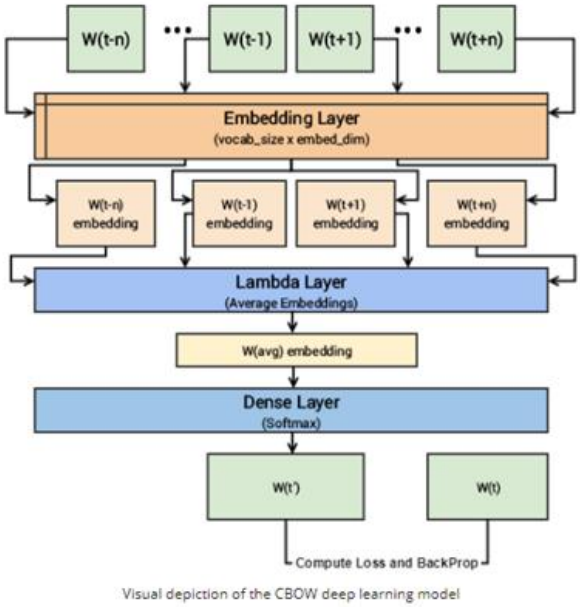



*Figure 4 explains the working of continuous bag-of-words architecture* [11]

**The model's working is described in the following steps:**

- As indicated in the figure below, the context words are initially fed into an embedding layer (which is initialised with some random weights).
- The word embeddings are then transferred to a lambda layer, where they are averaged out.
- Our target word is predicted by a dense SoftMax layer, which receives these embeddings and processes them. To keep the embedding layer updated, we do backpropagation with each epoch while comparing this to our target word and computing the loss.
- Our embedding layer can be used to extract the embeddings of the desired words after the training is complete.



Visual depiction of the CBOW deep learning model

*Figure 5 depicts the architecture of CBOW deep learning model* [12]

**How does the Skip-gram work?**

Predicting context words based on a target (centre) word is possible using the skip-gram technique. For the same example mentioned above, "Deep learning models perform better," given the term 'learning' and the context window size of 2, an algorithm is attempting to forecast ("deep", "models)" and so on.

We feed the skip-gram model (X, Y) pairs, where X is our input and Y is our label, in order to predict several words from a single input word. This may be done by creating both positive and negative input samples.

When using positive input samples, the following data will be used for training: [(target, context),1] where target is the centre word and context is the surrounding context words and label 1 indicates the importance of the pair. [(target, random),0] is the training data for Negative Input Samples, same as it is for Positive Input Samples with value “0” for label. Randomly selected words are substituted for the actual context-relevant ones in this example, and the label "0" indicates that the combination is irrelevant.

Words with similar meanings may be associated with similar embeddings based on these examples of contextually relevant keywords.

**The model's working is described in the following steps:**

- Both the target and context word pairs are fed into separate embedding layers, yielding dense word embeddings for each of these two terms[13].
- The dot product of these two embeddings is then computed using a 'merge layer,' and the dot product value is obtained.
- Using the result of the dot product, a dense sigmoid layer is produced, which outputs either 0 or 1.
- The predicted values are compared to the actual label and the actual loss is calculated. Based on the loss the embedding layer is updated during back propagation with each epoch.

Like CBOW, the embeddings of the skip-gram model can also be obtained by extracting out the embeddings of the needed words from our embedding layer, once the training is completed.

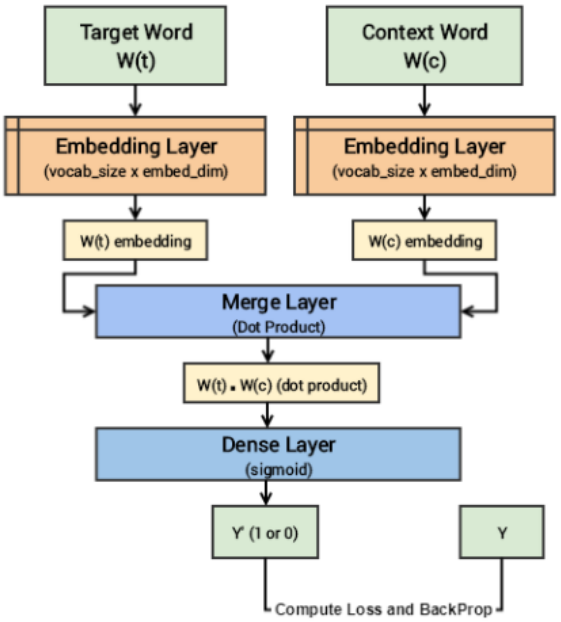



*Figure 6 The architecture of skip-gram deep learning model* [14]

Gensim package can be used to train and build Word2Vec models using different dataset corresponding to different tasks. To create the word embeddings using CBOW architecture or Skip Gram architecture, the mentioned code can be used:

```
Model_CBOW = gensim.models.Word2Vec(data, min_count = 1,size = 100,
window = 5, sg=0)

Model_Skip_gram = gensim.models.Word2Vec(data, min_count = 1, size =
100, window = 5, sg = 1)
```

*Figure 7 CBOW and Skip Gram model building using the gensim implementation - code snippet*

But setting the parameter sg = 0 we can train a CBOW model and by setting it to 1 we can build a skip-gram model.

### 2.4.4 GloVe

GloVe (Global Vectors for Word Representation) is another approach for generating word embeddings. GloVe has the benefit that, unlike Word2vec, it uses global statistics (word co-occurrence) to create word vectors rather than merely local statistics (local context information of words). GloVe combines the benefits of the two most important model families in the literature explained below.

**Matrix Factorization Techniques:** Methods for reducing a matrix into constituent parts to make more complex matrix operations easier to compute.

**Shallow Window-Based Methods:** Another technique is to learn word representations that help predictions inside local context windows.

#### Matrix factorization vs shallow-window methods

LSA and other matrix factorization algorithms use low-rank approximations to divide big matrices and extract statistical information from a corpus. There are rows for terms or ideas, and columns for documents in the corpus. Regarding HAL and derivative approaches, square matrices are used in which the rows and columns both correspond to words; each entry is based on how often a specific word occurs in relation to another (a co-occurrence matrix). Common words contribute disproportionately to the similarity score, needing some type of normalisation. This must be addressed.

Short-term approaches use local context windows to learn word representations:

These techniques, unlike matrix factorization methods, do not operate directly on co-occurrence data in the corpus. The huge repetition in the data is missed by these models, which instead scan the whole corpus for context windows.

#### Pre-trained Word Embeddings

Pretrained Word Embeddings are embeddings learned in one task and used to another comparable activity. These embeddings are trained on big datasets, stored, and then applied to different problems[5].

We can easily make use of pre-trained word embeddings trained on huge corpus of data for example Word2Vec pre-trained embedding developed by Google on the Google News dataset

(about 100 billion words), GloVe pre-trained model built using 6 billion Wikipedia data by Stanford NLP and fastText word embedding developed by Facebook.

Steps to convert text to features using pre-trained word embeddings[5]

1. Loading the pretrained word embeddings
2. Creating a tokenizer object
3. Transforming text documents to sequence of tokens and pad them
4. Create a mapping of token and their respective embedding.

# Chapter 3

# Algorithms

## 3.1 Machine Learning

The study of computer algorithms that can learn and improve on their own is known as machine learning (ML). As a component of artificial intelligence, it is widely accepted. In order to generate predictions without being explicitly trained to do so, machine learning algorithms develop a model based on sample data, known as training data. The development of typical algorithms to carry out the needed tasks would be difficult or impossible in a wide variety of applications, including medicine, email filtering, voice recognition, and computer vision.

An example of machine learning is a system that employs preprogramed algorithmic processes to predict output values in an acceptable range from input data. In the long run, these algorithms get more intelligent as new data is fed into them and their processes are optimised.

To use these machine learning algorithms to solve our problem statement the news articles with their corresponding label is fed to the algorithm to train it. Then the trained algorithm is used to automatically predict the label of the new news articles. The raw text can't be fed directly to these algorithms, so the raw text data is pre-processed to remove noise from it by a series of pre-processing steps. This cleaned text is converted into feature vectors, these vectors can then be used build classifiers which is used to predict the news articles into different tags.

### 3.1.1 Naive Bayes Classifier

Bayesian classification is based on an assumption that all predictors are independent. The assumption of a Naive Bayes classifier is that the presence of one feature in a class has nothing to do with the presence of any other feature whatsoever.

But the assumptions made by Naive Bayes are not generally correct in real-world situations. It turns out, the "independence assumption" is frequently not satisfied, which is why it is referred to as "Naive."

In addition to text analysis, machine learning methods may be used in many other fields. Algorithms need numerical feature vectors with a defined size rather than variable-length raw text documents, hence the raw data cannot be submitted directly to the algorithms. Therefore, we use TF-IDF, Word2Vec, and other techniques to extract characteristics from the text and input them into the algorithm.

Probability may be calculated by using a mathematical formula known as the Bayesian Theorem. According to Bayes' theorem, the mathematical expression is as follows:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Where,

A and B are called events.

P (A | B) is the probability of event A, given the event B is true (has occurred). Event B is also termed as evidence.

P(A) is the priori of A (the prior independent probability, i.e., probability of event before evidence is seen).

P (B | A) is the probability of B given event A, i.e., probability of event B after evidence A is seen.

In our case, we have sports as one of the tags so the probability where the news belongs to sports is calculated as below:

$$P(Sports|\text{A very close game}) = \frac{P(\text{a very close game}|Sports) * P(Sports)}{P(a\ very\ close\ game)}$$

Since for our classifier we're just trying to find out which tag has a bigger probability, we can discard the divisor —which is the same for both tags— and just compare[15][16].

$P(a\ very\ close\ game|Sports) * P(Sports)$

With other categories like

$P(a\ very\ close\ game|Business) * P(Business)$

$P(a\ very\ close\ game|Science) * P(Science)$

$P(a\ very\ close\ game|World) * P(World)$

The we choose the text which has the highest probability with the corresponding tag.

### 3.1.2 Linear Classifier

Logistic regression is a statistical technique to predict a target or independent variable (Y) using one or more predictor or dependent variables(X).

It is used to model a binary outcome, such that the target variable can only have two possible values. In contrast to linear regression, the model's output is a binary value (0 or 1) rather than a numeric number. Logistic Regression can also be modified to predict a target variable which is not binary and has more than 2 classes[17].

Below is an example logistic regression equation:

$$y = \frac{e^{b0+b1*X}}{(1 + e^{b0+b1*X})}$$

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient of the input X.

The logistic regression algorithm's coefficients (Beta values b) must be computed using the training data. This is accomplished using maximum-likelihood estimation. Although it does make assumptions about the distribution of your data, maximum-likelihood estimation is a common learning technique utilised by several machine learning algorithms. Consider the example of spam/No spam categorization. The optimal coefficients would provide a model that predicted a value that was very close to spam for the default class and a value that was very close to Nospam for the other class. The idea behind maximum-likelihood logistic regression is that a search technique seeks coefficient values (Beta values) that minimise the error between the probabilities predicted by the model and those in the data[18]

**Working of Logistic Regression**

Given a data (X, Y), X being an input values and y being the target. The objective is to train the model to predict which class the future values belong to.

Step 1: Primarily, we create a weight matrix with random initialization. Then we multiply it by the input features.

Step 2: Then predict the probability values using the learned weight. Then calculate the loss using the Cross-Entropy between 2 probability distribution.

Step 3: Calculate the gradients and update the weights and repeat step 2 until the loss is very minimal.

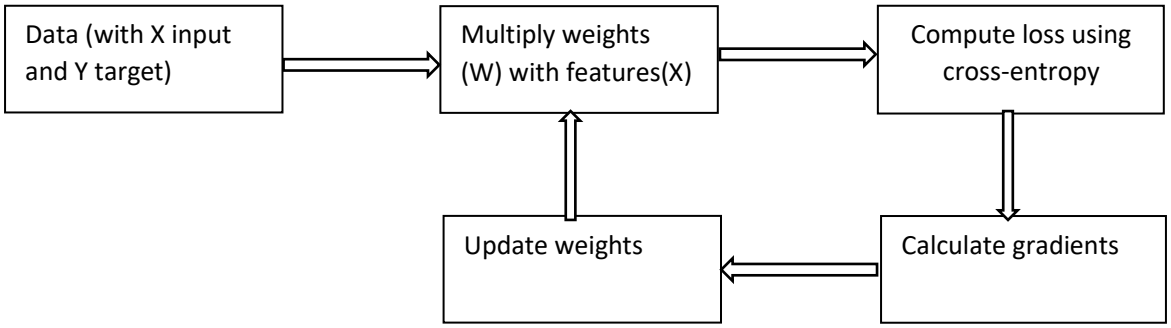



*Figure 8 Logistic Regression Working Flowchart*

Logistic Regression is used majorly to solve binary classification problems, which aims to create a binomial probability distribution function. This means that the class labels have a value of 1 for the positive class or result and 0 for the negative. The model tells us how likely it is that a given case will fall into the first category. There are several situations when logistic regression cannot be applied, such as multi-class classification jobs[18].

In order to accommodate multi-class categorization, it must be modified. When dealing with multi-class classification challenges, one common strategy is to break the problem into several binary classification problems and fit a typical logistic regression model to each subproblem. Examples of this are the one-vs rest and one-vs-one wrapper models.

As an alternative, the logistic regression model may be modified to directly predict multiple class labels. Predicting the likelihood that a given input example corresponds to a known class label, specifically[18].

### 3.1.3 Support Vector Machine

Support vector machines is an algorithm that determines the best decision boundary between vectors that belong to a given group (or category) and vectors that do not belong to it. It can be applied to any kind of vectors which encode any kind of data[19]. This means that in order to leverage the power of SVM text classification, texts must be transformed into vectors. When we have 2 classes to classify our hyperplane will a line. If we have more than 2 classes, we are now in three or more dimensions, our hyperplane can no longer be a line. It must now be a plane as shown in the example above. The idea is that the data will continue to be mapped into higher and higher dimensions until a hyperplane can be formed to segregate it[20].

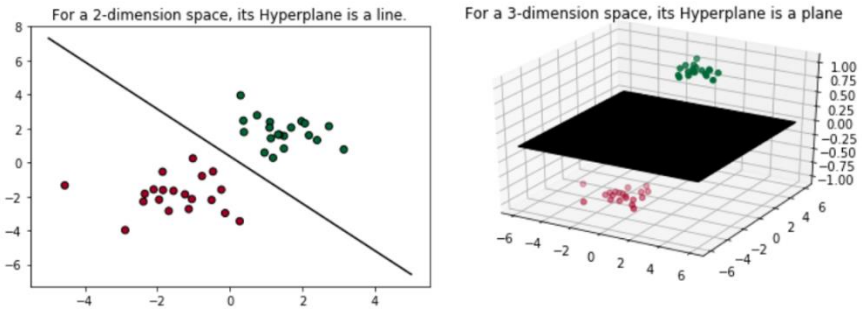



*Figure 9 Visual representation of a simple SVM model*

**What is a hyperplane?**

In an n-dimensional Euclidean space, a hyperplane is a flat, n-1 dimensional subset that divides the space into two separate parts.

**What is a margin?**

The SVM, defines the criterion to be looking for a decision surface that is maximally far away from any data point. The classifier's margin is determined by the distance between the decision surface and the nearest data point.

**SVM in non-separable linear cases**

In the case of linearly separable classes, SVM attempts to find the hyperplane that maximises the margin while ensuring that both classes are correctly categorised. In practise, datasets are mostly unlikely to be linearly separable, therefore the criterion of a hyperplane properly classifying 100 percent of the data will never be satisfied.

Soft Margin and Kernel Tricks are two techniques introduced by SVM to solve non-linearly separable scenarios.

**Soft Margin**: try to identify a line to divide data points but allow for one or two misclassified data points.

**Kernel Trick**: uses existing features, applies transformations, and builds new features. These additional properties are critical for SVM in determining the nonlinear decision boundary.

We can use SVM for the text classification task. To implement SVM to our text classification problem the news articles should be converted into n-dimensional vectors using suitable word embedding techniques (such as Bag of words, tf-idf, etc.). Then these vectors can be represented in a n-dimensional space, and we can use SVM to find an optimal hyperplane that separates these news articles into their respective categories.

### 3.1.4 Bagging Models

#### Bagging Ensemble Algorithm

Bootstrap Aggregation, often known as Bagging, is an ensemble machine learning technique. It is specifically an ensemble of decision tree models, although the bagging approach may also be used to integrate predictions from other types of models. Bootstrap aggregation, as the name implies, is based on the concept of the "bootstrap" sample. A bootstrap sample is a dataset sample with replacement. A sample picked from the dataset is replaced, allowing it to be selected again and maybe several times in the new sample. As a result, the sample may contain duplicate cases from the original dataset. To estimate a population statistic from a small data sample, the bootstrap sampling approach is utilised. This is accomplished by generating numerous bootstrap samples, calculating the statistic for each, and providing the mean statistic for all samples[21].

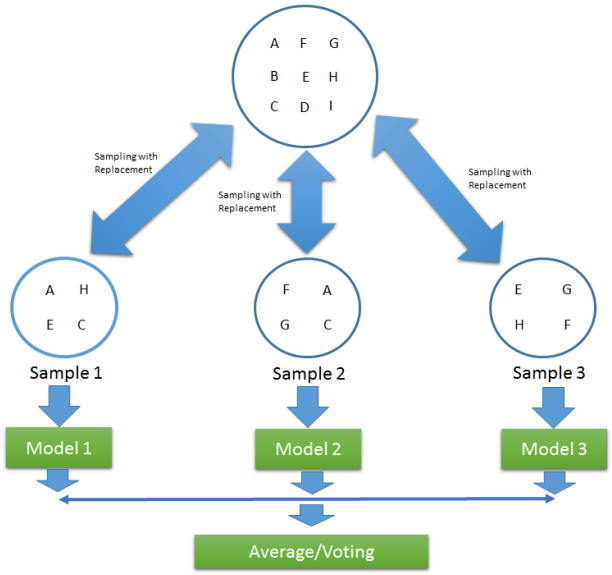



*Figure 10 Bagging algorithms working flowchart*

Bagging predictors is a technique for creating multiple predictors and then combining them to get an aggregated predictor. Making bootstrap replicas of the learning set and utilising these as new learning sets results in a strong prediction. For regression, predictions are created by averaging the predictions across the decision trees. For classification issues, predictions are formed by choosing the majority vote for the classes from across the predictions provided by the decision trees. Bagged decision trees are useful because each tree is trained on a little different training dataset, allowing each tree to have tiny variations and generate slightly different skilful predictions. We think the strategy is effective because the trees have a low correlation between predictions and, consequently, prediction errors[22].

Decision trees, especially unpruned decision trees, are utilised because they have a high variance and somewhat overfit the training data. Other high-variance machine learning techniques, such as k-nearest neighbours with a low k value, can be utilised, however decision trees have shown to be the most successful.

Bagging can be used for text classification problem as the data points after being converted to vectors using ideal vectorization technique can be used to build different classifiers. Then the results of these classifiers can be used to obtain a prediction by majority voting technique.

**The key benefits of bagging include:**

Ease of implementation: Combining the predictions of basic learners or estimators to increase model performance is simple.

Bagging can be used to minimise variation within a learning system. This is especially useful with high-dimensional data, because missing values might contribute to larger variance, increasing the likelihood of overfitting and impeding appropriate generalisation to new datasets[21].

**The key challenges of bagging include:**

Loss of interpretability: Because of the averaging involved among predictions, it is difficult to get precise business insights from bagging. While the output is more precise than any single data point, a more accurate or full dataset may also result in better accuracy inside a single classification or regression model.

Computationally expensive: As the number of iterations grows, bagging slows and becomes more demanding. As a result, it is not well suited for real-time applications.

**Random Forest**

One of the most widely used boosting algorithm to solve different problem statement is Random Forest a supervised learning algorithm. The "forest" it creates is an ensemble of decision trees, which are often trained using the "bagging" approach. The bagging approach is based on the premise that combining learning models improves the final output. Using Random Forest, you can make trees out of several different samples of your training data. It also limits the features that can be used to make the trees, which makes each one unique. This, in turn, can improve performance. In simple random forest, numerous decision trees are built and merged to produce a more accurate and stronger predictor[21].

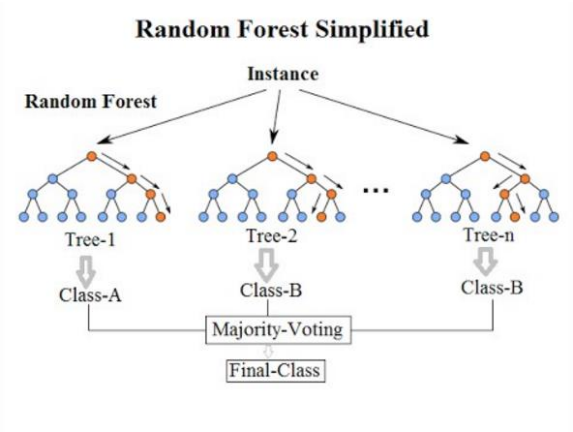



*Figure 11This picture depicts the working of Random Forest algorithm*

The Random Forest algorithm's overall procedure is as follows:

1. For each tree, create a bootstrapped data set.

2. Decision tree is created using the bootstrapped data set but use a random subset of variables or features to split on at each node.

3.Repeat all the processes many numbers of times to create a large forest with a diverse range of trees. This is the difference between a Random Forest from a single decision tree.

Hyperparameters are important because they influence the overall behaviour of a machine learning model. The goal is to find the best combination of hyperparameters to minimise a predefined loss function and produce better outcomes. Random forest hyperparameters are used to boost the prediction capability of the model and to build the model fast. Let's have a look at the hyperparameters that needs to be tuned while building a random forest model.

The first hyperparameter is the number of trees the algorithm builds before doing maximum voting or calculating prediction averages. In general, more trees improve efficiency and makes predictions more stable, but it also slows down the calculation. Another critical hyperparameter is the maximum number of features that random forest will examine when splitting a node. The final critical hyperparameter is the minimum of leaves necessary to separate an internal node[21].

### 3.1.5 Boosting Models

Boosting is a general ensemble approach for creating a strong classifier from a set of weak classifiers. The word 'Boosting' refers to a family of algorithms that transforms weak learners into strong learners. Boosting is an ensemble strategy for enhancing any learning algorithm's model predictions. The aim of boosting is to train weak learners progressively, with each one attempting to correct the one before it. This is accomplished by first developing a model from the training data, followed by the creation of a second model that seeks to rectify the faults in the first model. Models are added until the training set is accurately predicted or until the maximum number of models is reached. Unlike many ML models, which focus on high-quality prediction performed by a single model, boosting algorithms strive to enhance prediction power

by training a series of weak models, each of which compensates for the shortcomings of its predecessors[23].

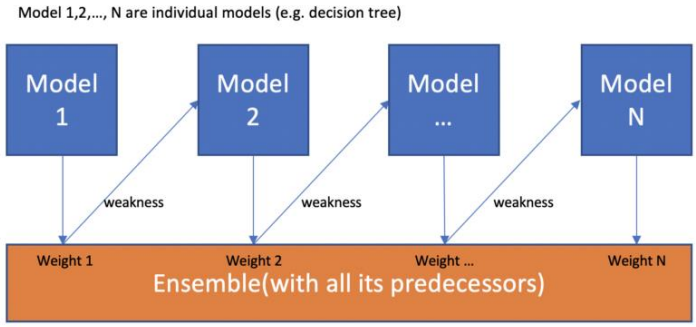



*Figure 12 pictorial representation of a Boosting models working*

## Types of Boosting

### AdaBoost (Adaptive Boosting)

Adaboost creates a single strong learner by combining numerous weak ones. In AdaBoost, the weakest learners are decision stumps, which are decision trees with just one split. In the first decision stump that AdaBoost creates, all observations are weighted the same way. In order to fix the prior mistake, observations that were incorrectly classified now carry a larger weight than those that were properly recognised. Classification and regression issues may be solved using AdaBoost algorithms[23].

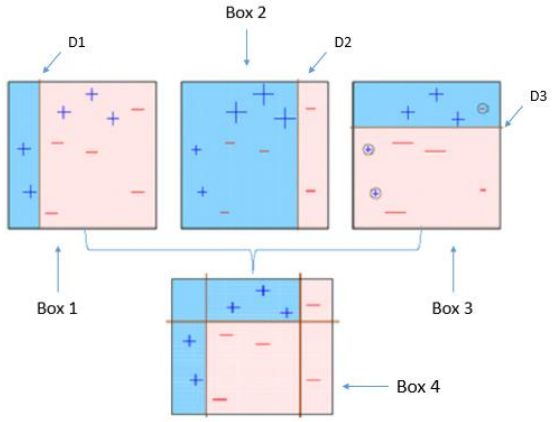



*Figure 13 Simple example to explain how weights are adjusted for the misclassifications in the previous learners*

The first decision stump (D1) may be seen dividing the (+) blue area from the (-) red area in the figure above. Three improperly labelled (+) may be seen in the red section of D1. After the second learner receives the misclassified (+) samples, they will be more important than the previous observations and samples. The model will continue to correct the previous model's mistake until the most accurate predictor is produced, at which point the model will be stop learning.

### Gradient Boosting

In the same way AdaBoost works, Gradient Boosting works by adding more and more predictors to an ensemble, each of which corrects the one before it. While AdaBoost modifies the weights for every wrongly classified observation, Gradient Boosting does not[23].

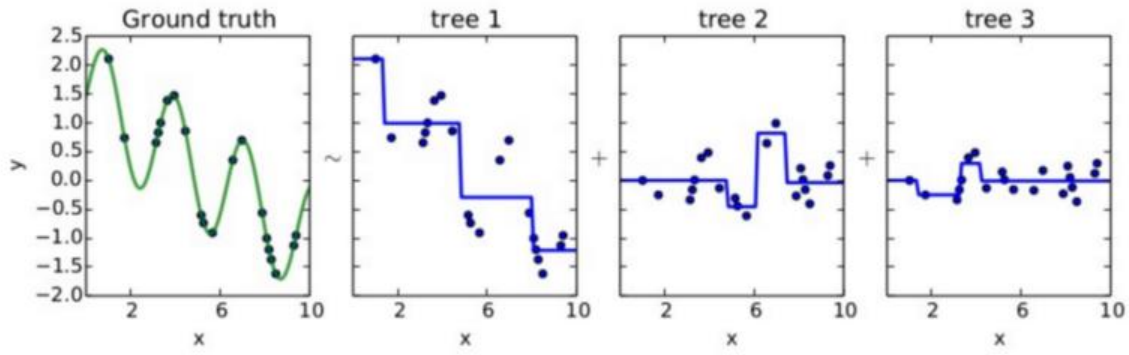



*Figure 14 Boosting Models learning improvement based on its previous errors*

Using the Gradient Descent technique, Gradient Boosting (GBM) identifies the errors in the predictions of the preceding learner. The working of GBM algorithm follows the steps below.

F1(x) = y is a model that fits to the data

The residuals should be used to fit a model, such as h1(x) = yF1 (x)

Create new model, F2(x) = F1(x) + h1(x) (x)

Our final model can account for a large portion of the original model's error and minimise this error over time by adding weak learners one after the other.

### XG Boost (eXtreme Gradient Boosting)

XGBoost is an abbreviation for eXtreme Gradient Boosting. XGBoost is a gradient-boosted decision tree implementation optimised for speed and performance. Gradient boosting machines are notoriously slow to construct due to sequential model training. As a result, they are not particularly scalable. As a result, XGBoost prioritises computational speed and model performance[23].

## 3.2 Deep Learning

Artificial neural networks and representation learning are the foundations of deep learning, a subset of a broader class of machine learning algorithms. Deep Learning can be supervised, semi-supervised, or unsupervised. It goes beyond machine learning by constructing more complicated hierarchical models that mimic how people acquire new knowledge.

In fields like computer vision, speech recognition, natural language processing, machine translation, medical image analysis, and board games, deep-learning architectures such as deep neural networks, recurrent neural networks, convolutional neural networks, and deep

reinforcement learning have been used to produce results comparable to, and in some cases exceeding, human expert performance[24].

Deep learning has numerous benefits over machine learning, including end-to-end training and representation learning. Because of this, deep learning is now a feasible option for natural language processing, as opposed to the more conventional methods of machine learning. No human intervention is needed to build deep learning models from a parallel corpus. In contrast to the classic statistical machine translation method, which mainly relies on feature engineering, this approach has many advantages.

Model interpretability and the requirement for vast amounts of data and high processing resources are increasingly prevalent deep learning challenges. Deep learning typically uses vector data (real-valued vectors), while language data is symbol data, which is distinct from the vector data that is often used. Neural networks are now being used to translate a person's language into vector data, which is then fed back into a language model[25].

### 3.2.1 Convolutional Neural Network (CNN)

The convolutional neural network (CNN) models are predominant in the image data domain. They perform extremely well in computer vision tasks like as image recognition, object detection, and image classification. These neural networks attempt to replicate the human brain and its learning process. The neural network, like the brain, accepts information, processes it, and creates output. These three operations – accepting input, processing information, and producing output – are represented in a neural network by three layers – input, hidden, and output.

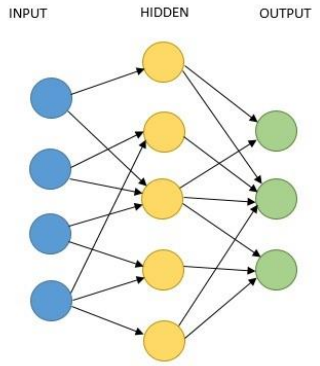



*Figure 15: CNN with a single hidden layer*

When image classification models were first developed, they relied on the raw pixels of an image to categorise it. Cats may be classified based on their fur colour and ear shape using colour histograms and edge detection. Until the approach confronts more complicated versions, it has proven effective. Because the model doesn't account for additional factors, classical picture recognition fails. Using CNNs, we can extract more accurate representations of the content of a picture. When compared to conventional image recognition, CNN takes raw pixel data from a photo, trains the model, then automatically extracts features for categorization[26].

**How CNN works for Image Classification**

CNNs are constructed from neurons with learnable weights and biases. Each neuron gets large number of inputs and then computes a weighted sum of them before passing them through an activation function and responding with an output. The forward and backward propagations are the two fundamental phases in a CNN's learning process. During the forward pass, the network predicts the output based on the weights and biases learnt by the neurons. There are different algorithms that can be used to update the weights of nodes in a neural network so that nodes with higher error rates have lower weights, and vice versa. This process is known as back propagation, and it involves comparing predicted and actual values and propagating the difference back into the neural network.

Input layer is the first layer in a Convolutional Neural Network (CNN), which is made up of artificial input neurons and is responsible for bringing the initial data into the system for processing by following layers of artificial neurons.

The convolutional layer, which does the bulk of the computation is the second layer and is the building block of the architecture. Data or images are convolved using filters or kernels. Filters are small units that is applied to an image through a sliding window. For each sliding movement, the image's filters are combined to produce an element-wise product, and the total of those values is calculated. A 2d matrix would be the output of a convolution with a 3d filter and colour. The activation layer will then use the ReLu (Rectified Linear Unit), and the rectifier function will be used to increase non-linearity in the CNN.

The third layer is the pooling layer. The feature maps' sizes are reduced by pooling layers. As a result, it decreases the number of parameters to learn as well as the amount of computation done in the network. The pooling layer summarises the characteristics in an area of the feature map produced by a convolution layer. Finally, comes the fully connected layer, which requires flattening. This involves reducing the whole pooled feature map matrix to a single column, which is then fed into the neural network for processing. We built a model by combining these characteristics using fully connected layers. Finally, to classify the output, we have an activation function such as SoftMax or sigmoid[27].

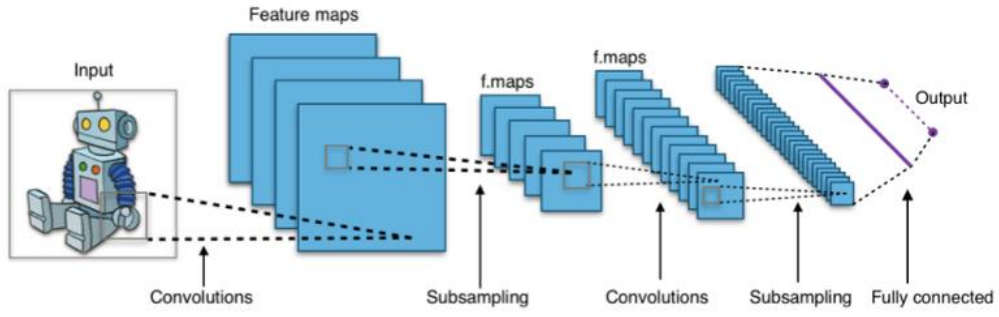



*Figure 16: Working architecture of a CNN model for an image classification problem*

**How to use CNN for text classification**

In the case of NLP tasks, which are applied to text rather than images, we have a one-dimensional array that represents the text. In this case, the ConvNets' architecture is altered to 1D convolutional-and-pooling operations. One of the most common tasks in NLP where ConvNet is employed is sentence classification, which is the process of categorizing a text into a set of pre-determined categories using n-grams, i.e., sequences of words, or sequences of characters.

In image classification we convert the input image a matrix of pixel values which is used as input to the CNN algorithm, to apply CNN to text classification we must convert the input text to a multi-dimensional vector which can be achieved using word embeddings like Word2Vec or GloVe. For instance, we can find the learned embedding of each word and add then according to the sequence to create a matrix where the number of rows represent the number of words, and the number of columns is the dimension of the embedding that has been chosen. This matrix can be then fed into a CNN algorithm as input.

The major difference in using CNN for image and text classification is the use of convolutional filters. For image classification, we slide through rows and columns to extract information, but for text classification, each row of the matrix should be considered as a single unit, so the filter is slid over the columns, not over the rows (i.e., only in 1 dimension not in 2 dimensions). This diagram below depicts how the convolutional filters are applied over the input text.

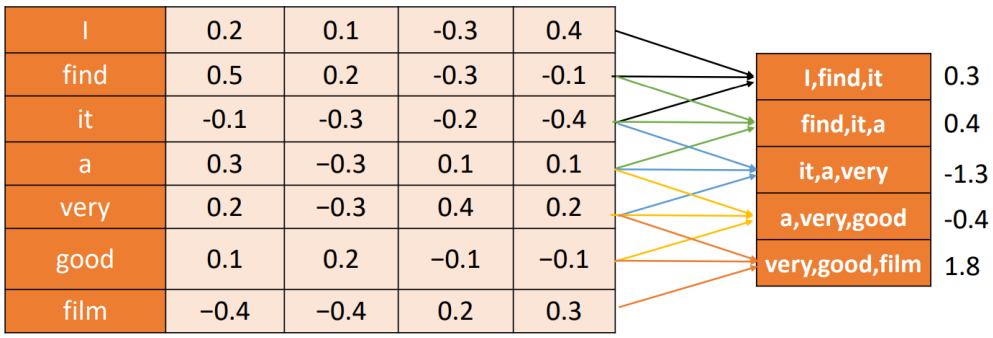



*Figure 17: How a filter is convolved over the data to find patterns*

So, the input text converted as a vector is passed to the input layer then the convolutional filters and activation functions are used to extract pattern from the text then the pooling layers is used to reduce the size of the vector then it is passed to a fully connected layer which uses a softmax or sigmoid function to classify the text into different categories. The working of a CNN for a text classification problem is showed in the below diagram.

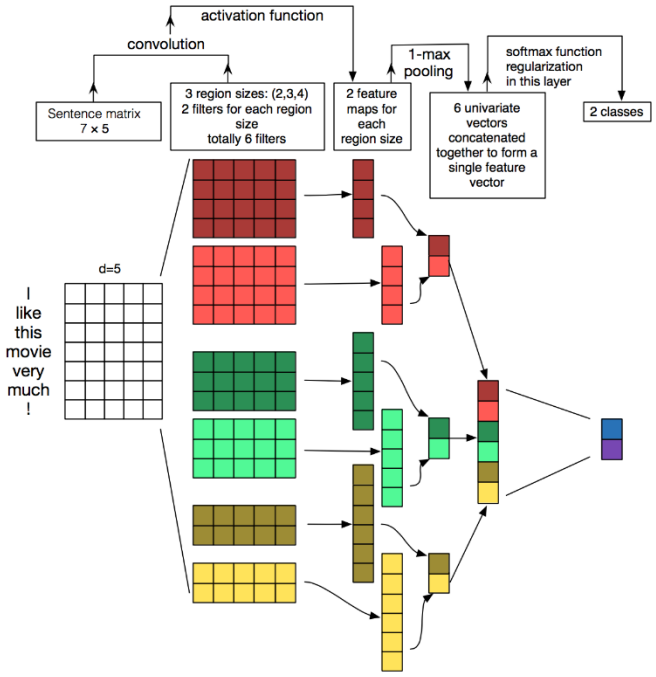



*Figure 18: Working architecture of an CNN model for a text classification task*

### 3.2.2 Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM)

RNNs are a type of artificial neural network in which node connections create a directed graph along a sequence. It is essentially a series of neural network pieces connected like a chain. Each is conveying a message to a successor. CNN is a type of deep, feed-forward artificial neural network in which nodes' connections do not form a cycle. CNNs are often employed in computer vision, but they have also demonstrated promising results when applied to diverse NLP applications. An RNN is taught to detect patterns over time, whereas a CNN is trained to recognise patterns across space[28].

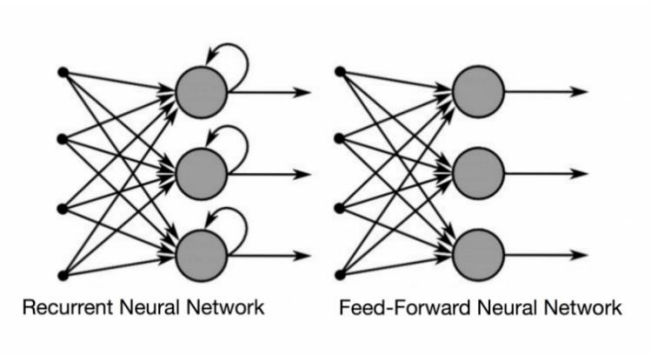



*Figure 19: Structural difference between an CNN and an RNN*

Recurrent neural networks (RNN) are the state-of-the-art algorithm for sequential data. They are used by established organisation like Apple and Google in their voice search applications. It is an ideal choice of deep learning model to solve problems using sequential data, because RNN can be able to remember their inputs due to their internal memory. RNNs may recall

important details about the input they received because of their internal memory, allowing them to predict what will happen next with great accuracy. Therefore, they are the preferred algorithm for time series, voice, text, video and many other types of sequential data. Recurrent neural networks can learn more about a sequence and its context than other algorithms[28].

**Working of Recurrent Neural Network (RNN)**

Three steps are involved in the operation of Recurrent Neural Networks. To begin, it advances through the hidden layer, making a prediction along the way. The loss function is used to compare the predicted value to the actual value in the second stage. A model's loss function shows how well it's doing. Having a lower loss function value indicates an improved model. Back-propagation is used to further compute the gradient for each point in the final step (node). The gradient is used to change the weights of the network at each point.

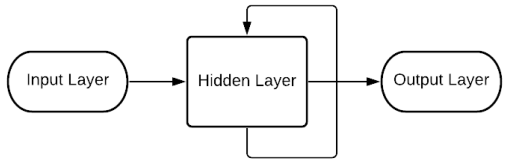



*Figure 20: This flow chart depicts the working flow of an RNN algorithm*

When working with sequential data, recurrent neural networks are widely utilised. The reason for this is that the model uses layers, which provide the model with short-term memory. It can more accurately predict the next data using this memory. As a result, an RNN receives two inputs: the present and the recent past. This is significant because the data sequence provides critical information about what is to come, which is why an RNN can achieve things that other algorithms cannot. A feed-forward neural network, like all other deep learning algorithms, gives a weight matrix to its inputs before producing an output. It is worth noting that RNNs apply weights to both the current and prior input. A recurrent neural network will also adjust the weights for both via gradient descent and backpropagation over time (BPTT).

Forward-propagation is used in neural networks to obtain the output of your model and to determine if this output is accurate or erroneous in order to obtain the error. Backpropagation is just traversing your neural network backwards to obtain the partial derivatives of the error regarding the weights, allowing you to remove this value from the weights[28].

Gradient descent, a technique that may iteratively minimise a given function, uses these derivatives. The weights are then adjusted up or down based on which reduces the inaccuracy. During the training phase, a neural network learns just like this. So, using backpropagation, you simply try to change your model's weights during training. When an RNN is unrolled for each timestep, the process of backpropagation through the unrolled RNN is called BPTT. The picture below shows an RNN that has been unrolled. This visual representation helps you understand what's going on in an RNN network at each timestamp. The RNN is unrolled following the equal sign on the left. There are no cycle follows the equal sign since the discrete time steps are observable and information is passed from one time step to the next.

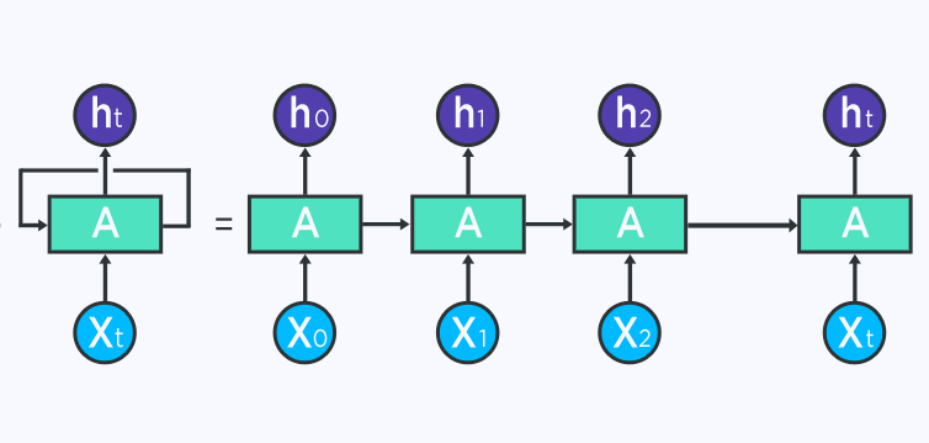



*Figure 21: Depicts the architecture of an RNN model which is unrolled for each timestep*

While unrolling all timesteps, the error in each timestamp is propagated back from the last to the first timestep in BPTT. This enables for the calculation of the error for each timestep, which allows for the updating of the weights. It should be noted that BPTT can be computationally costly when there are many timesteps.

**Two problems with ordinary RNNs**

There are two major challenges that RNNs have had to overcome, but in order to understand them, you must first understand what a gradient is. A gradient is the partial derivative of its inputs. If you're not sure what it implies, consider this: a gradient quantifies how much the output of a function varies when the inputs are changed slightly. A gradient can also be thought of as the slope of a function. The steeper the slope and the faster a model can learn, the higher the gradient. However, if the slope is zero, the model fails to train. A gradient is just the change in all weights in relation to the change in error[28].

**Exploding Gradients**

Exploding gradients occur when the algorithm assigns an absurdly high value to the weights for no obvious reason. Fortunately, by truncating or squashing the gradients, this problem may be effectively solved.

**Vanishing Gradients**

Vanishing gradients occur when the values of a gradient are too small, causing the model to stop learning or to take far too long.

**Long Short-Term Memory (LSTM)**

The Long short-term memory solves the exploding and vanishing gradient problems in the RNN's. Long short-term memory networks (LSTMs) are a type of recurrent neural network extension that basically extends memory. Long Short-Term Memory (LSTM) networks are RNN extensions that increase memory capacity. LSTMs are used to construct the layers of an RNN. LSTMs apply "weights" to data, which allows RNNs to either let new information in, forget information, or give it enough relevance to affect the output[29].

The units of an LSTM are used to build the layers of an RNN, which is also known as an LSTM network. RNNs can remember inputs for a long time because of LSTMs. This is because LSTMs store information in a memory, similar to a computer's memory. The LSTM could read, write, and erase data from its memory.

This memory may be thought of as a gated cell, where the cell decides whether to store or erase information (i.e., whether to open the gates) based on the value it attributes to the information. Weights are used to assign importance, which are also learnt by the algorithm. This basically implies that it learns what information is important and what is not over time. An LSTM has three gates: input, forget, and output. These gates decide whether to allow fresh input (input gate), discard the information because it isn't relevant (forget gate), or let it affect the output at the current timestep (output gate). A RNN with its three gates is seen below[29].

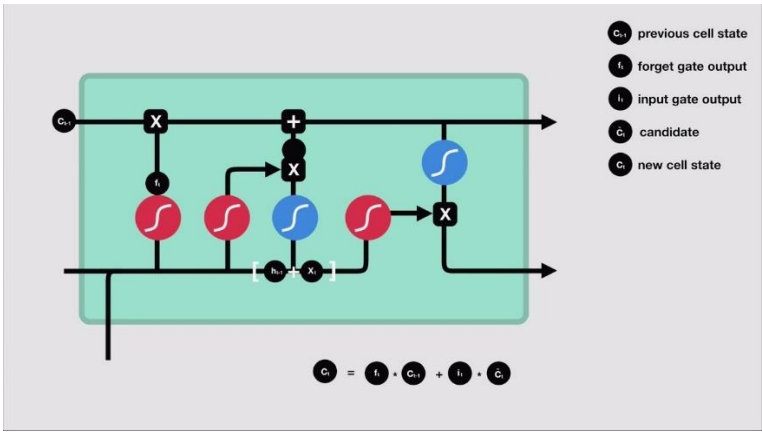



*Figure 22: The working and components of an LSTM gate*

An LSTM's gates are analogue in the form of sigmoid, which means they have a range of zero to one. Because they are analogue, they may do backpropagation. LSTM solves the problem of vanishing gradients by keeping the gradients steep enough to make the training reasonably quick and the accuracy high.

### Implement RNN with LSTM for Text Classification

The encoder is the first layer, and it turns the text to a series of token indices.

Embedding layer comes after the encoder and the embedding layer stores one vector per word. The embedding layer converts the sequence of words into sequence of vectors when called and these vectors are trainable. Words with similar meaning often used to have similar vectors after training with enough data points.

By iterating over the elements, a recurrent neural network (RNN) handles sequence input. RNNs use the outputs from one timestep as input for the following timestep. This forwards and backwards propagates the input across the RNN layer before concatenating the final output[30].

Once the RNN has transformed the sequence into a single vector the two layers are combined. Then a dense layer does some further processing to convert this vector representation to a single logit which is used as a classification result[30].

The diagram below describes the model architecture and implementation of text classification using LSTM based RNN architecture.

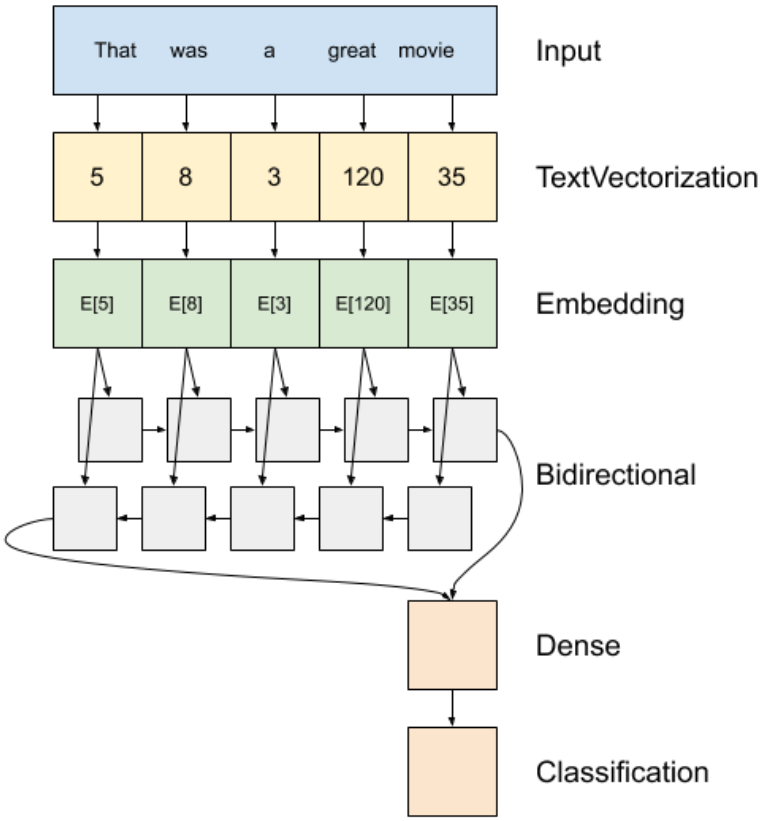



*Figure 23: Working of an RNN architecture with LSTM layers for a text classification task*

### 3.2.3 Bidirectional Encoder Representations from Transformers (BERT)

It was developed by researchers at Google AI Language and is known as Bidirectional Encoder Representations from Transformers (BERT). Natural language processing (NLP) researchers have deemed it a "ground-breaking" technology since it is the first ever bidirectional and entirely unsupervised method for representing language. The Transformer encoder, unlike directional models, reads the complete sequence of words at once instead than sequentially (left to right or right to left). Because of this, it is deemed bidirectional, yet it would be more correct to state that it is non-directional. Allows the model to learn a word's context from all its surrounding context (left and right of the word)[31].

**Why BERT?**

A Seq2Seq model is one that takes one sequence of objects (words, letters, time series, etc.) and produces another sequence of items. The model is made up of two parts: an encoder and a decoder. The encoder stores the context of the input sequence as a hidden state vector and delivers it to the decoder, which generates the output sequence. Because the job is sequence-based, both the encoder and decoder tend to utilize RNNs, LSTMs, GRUs, and so on. The output sequence is largely reliant on the context established by the hidden state in the encoder's

final output, making it difficult for the model to deal with extended words. In the case of extended sequences, the beginning context is almost certainly gone by the end of the series. This issue in the seq2seq models is solved using technique called "Attention", and BERT is built using Attention mechanism as its core.

The lack of high-quality training data is a major obstacle in the field of natural language processing (NLP). However, in order to produce task-specific datasets from the vast quantity of text data available, it is necessary to partition that pile into the many different areas. There are thus just a small number of training cases that have been human labelled. It's unfortunate that deep learning-based NLP models need a lot more data to work well; they see considerable advances when trained on millions, even billions, of annotated training examples. An enormous quantity of unannotated content on the web has been used by academics to train general-purpose language representation models, which has helped close the data gap. When dealing with tasks like question answering and sentiment analysis, BERT's pre-trained model may be fine-tuned on smaller task-specific datasets. When compared to starting from scratch and training on smaller task-specific datasets, this method yields much higher accuracy outcomes[31].

**Understanding How BERT works?**

BERT uses attention mechanisms that learn contextual links between words (or sub-words) in a text, such as Transformer. There are two separate processes in transformers: an encoder that reads the text input and a decoder that produces a prediction for the task. The encoder method is all that is needed for BERT since the goal is to develop a language model. Instead, of reading the text input sequentially, the Transformer encoder reads the complete sequence of words (left-to-right or right-to-left). However, it should be defined as non-rather than bidirectional because of this fact. It allows the model to learn the meaning of a word based on the context in which it is used (left and right of the word).

The Transformer encoder is detailed in the chart below. Vectors are used to embed tokens in the input before neural network processing begins. For each input token, a sequence of H-dimensional vectors is returned. While training language models, it is difficult to set a specific goal for the model's predictions. There are many methods that predict the next word in the sequence (such as saying, "The child returned from..."), which restricts context learning fundamentally.

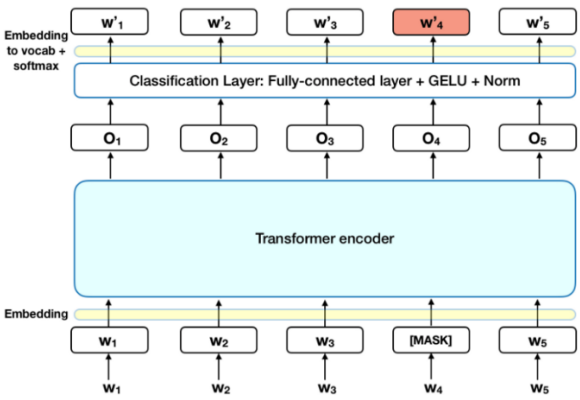



*Figure 24: The above picture depicts the architecture of BERT architecture*

BERT uses two training strategies, to overcome this challenge:

- Masked LM (MLM)
- Next Sentence Prediction (NSP)

**Masked LM (MLM)**

A [MASK] token is substituted for 15% of each word sequence before it is sent into BERT. The model then tries to estimate the original value of the masked words based on the context supplied by the other, non-masked phrases in the sequence. For the output word prediction to work, the following is required[31]:

On top of the encoder output, a classification layer is added.

The output vectors are multiplied by the embedding matrix to get the vocabulary dimension.

The probability of each word in the lexicon is calculated using a Softmax function.

The BERT loss function only examines the prediction of masked values and ignores non-masked words. Convergence is slower than in directed models because of the enhanced context knowledge[31].

**Next Sentence Prediction (NSP)**

It is at this phase of BERT training that the model learns how to predict whether the second sentence in a pair will appear next in the original text or not. This means that half of the sentences in the training corpus are paired, whereas the other half are just random phrases from the corpus. Because of this assumption, the second phrase may be considered an independent entity[32]. To assist the model in distinguishing between the two phrases during training, the input is handled as follows before entering the model:

[CLS] and [SEP] tokens are placed at the beginning of each sentence, respectively. Each token has a sentence embedding that indicates either Sentence A or Sentence B. Token embeddings with a vocabulary of 2 are similar in concept to sentence embeddings. Each token has a positional embedding added to it to show where it is in relation to the others[32]. In the Transformer paper, positional embedding is explained in detail. The following procedures are taken to forecast whether the second phrase is related to the first:

The Transformer model is applied to the full input sequence. Using a basic classification layer, the [CLS] token output is transformed into a 21 shaped vector (learned matrices of weights and biases).

Using softmax to calculate the likelihood of IsNextSequence. Masked LM and Next Sentence Prediction are trained simultaneously in the BERT model, with the objective of minimizing the combined loss function of the two techniques.

**How to use BERT for the text classification task**

Bidirectional Encoder Representations from Transformers is the acronym for BERT. It aims to pre-train deep bidirectional representations from unlabelled text by conditioning on the right and left side of the context at the same time[33]. Therefore, the pre-trained BERT model may be fine-tuned with just one extra output layer to provide cutting-edge models for a variety of NLP tasks. One of the most effective methods would be to fine-tune it using your own task and work-specific data. The embeddings from BERT can then be used as embeddings in our text documents. BERT (Bidirectional Encoder Representations from Transformers) is a large neural network design with many parameters ranging from 100 million to over 300 million. Overfitting would arise from training a BERT model from scratch on a small dataset. And BERT is heavily GPU dependent, so it is not possible to train a BERT model from scratch with large dataset in a laptop or personal computer. As a result, it is preferable to start with a pre-trained BERT model that has been trained on a large dataset. Model fine-tuning is the process of further training the model using a smaller dataset[33].

**Different Fine-Tuning Techniques**

**Train the entire architecture** – We may train the full pre-trained model on our dataset and then feed the result to a softmax layer for additional processing. The model's pre-trained weights are then re-evaluated using error on the new dataset the updated weights will be propagated throughout the architecture during backpropagation.

**Train some layers while freezing others** – Another option is to partly train a pre-trained model. While retraining higher layers, we may keep early layers of the model's weights fixed. We can experiment to see how many frozen layers and how many trained layers are optimal.

**Freeze the entire architecture** – To train a new model, we can even freeze all the model's layers and add a few neural network layers of our own. During model training, just the weights of the associated layers will be changed[34].

**Steps to implement BERT for text classification**

1. Install Transformers Library
2. Import BERT Model and BERT Tokenizer
3. Load the data
4. Tokenize the Sentences
5. Define Model Architecture (higher model layers, optimizers, loss functions etc.,)
6. Train the model for our problem statement.

# Chapter 4

# Experiments & Results

## 4.1 Experiment Setup

For evaluating the performance of the machine learning and deep learning models on the text classifications tasks, I have used the AG news classification dataset which comprises news articles under 4 different categories such as world, sports, business, and science and technology. The dataset comprises 127600 data points and the distribution is uniform, where there is an even number of data points in all 4 classes. There is no skewness in the data and the data is perfectly balanced.

For the building and evaluating the machine learning classifiers using different word embeddings such as count vectorizer and TF-IDF I will be using 80% of the data points which is around 102,080 samples for training the classifiers and the remaining 25,520 samples for testing the performance of the classifiers. To find the optimal hyper-parameters for all the different classifiers I have used grid search. Since the grid search to find optimal parameters using the whole train set took a lot of time to run, I used a sample from the train set and used it for hyper-parameter tuning. Then used those parameters were to train the classifiers with the complete training data.

On the other hand, the Deep learning tasks are computationally expensive, and I did not use any GPU or cloud infrastructure to train the deep learning models. So, I have used a sample from the whole dataset to train the deep learning architectures. I have used 5000 samples from each class which is less than 1/3 of the whole dataset. It is proven that deep learning performs better with more data but due to computational complexity, I have used a small sample to build and evaluate these models.

For the machine learning classifiers, I have used the sci-kit learn library one of the most robust libraries for machine learning in Python. To build the deep learning models I have used the Keras library from TensorFlow. This section will be about the results and evaluation of the classifiers and comparing their performances.

## 4.2 Evaluation Metrics

The metrics that are commonly used when evaluating supervised classifiers are accuracy and macro-average metrics which usually consist of scores of precision, recall and a combination of the two called an F1-score. We can also evaluate classifiers using AUC (Area Under Receiver Operator Characteristics (ROC))[35].

**Accuracy**

Accuracy is an important classification metric which is easy to understand and interpret. And it can be applied for both binary and multiclass classification problem. It is the percentage of outcomes that are accurate compared to the total number of cases studied[35].

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

where TN, FN, FP and FN stand for:

TP – True Positive, TN – True Negative, FP – False Positive, FN – False Negative

**Drawback**

For classification problems where the data is imbalanced and skewed, accuracy is an invalid metric to use.

**Precision**

What fraction of anticipated Positives is really Positive? When we want to be certain of our prediction, precision is a good statistic to use. For instance, in a system to predict a loan defaulter we must be certain of our predicting fraudulent defaulter, we should not predict a loyal customer as a defaulter which causes consumer dissatisfaction[35].

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

**Recall**

what proportion of actual Positives is correctly classified? When we aim to catch as many positives as possible, recall is a good option of assessment statistic. For instance, in a system to predict whether a person has heart related risk or not, we want to capture the condition even if we are not certain about the condition.

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

**F1-Score**

The F1 score is the harmonic mean of precision and recall and it usually a number between 0 and 1. We get a better F1 score if the classifier has a good balance between precision and recall. Hence it proves to be a great metric to measure a classifier[35].

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

**AUC**

How successfully the positive and negative probability are separated is shown by the Area Under the ROC curve. Using our classifier, we were able to calculate the probabilities. We can be used to find the sensitivity and specificity values which is plotted for different threshold values to get a ROC curve[36].

Where True positive rate is the proportion of true values, that is captured by the classifier.

$$sensitivity = True\ Positive\ Rate\ (TRP) = recall = TP/(TP + FN)$$

Where True positive rate is the proportion of false values, that is captured by the classifier.

$$1 - specificity = False\ Positive\ Rate\ (FDR) = \frac{FP}{TN + FP}$$

Like log loss, AUC has the advantage of being classification-threshold-invariant. It measures the model's ability to make accurate predictions regardless of the threshold used for classification, in contrast to F1 score and accuracy, which are tied to the threshold used. The below diagram depicts a sample ROC curve and how to understand the ROC curve[36].

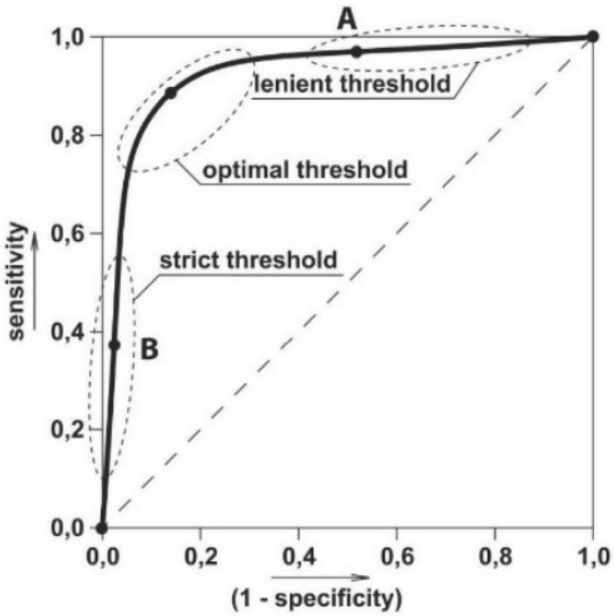


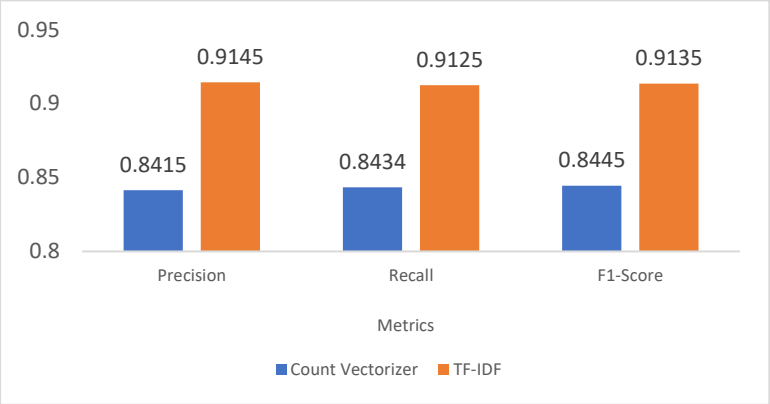

*Figure 25: How to plot and understand an ROC curve*

### 4.2.1 Machine Learning Models

**Naïve Bayes**

I have built the naïve bayes classifier using two word embedding techniques one is bag of words and the other is TF-IDF. I have evaluated the classifier using the evaluation techniques such as precision, recall and

F1-score on these two embedding techniques. From the metrics it's clear that TF-IDF embedding performs better for this text classification task. The performance metrics and comparisons are mentioned below.




| Naïve Bayes | | |
|---|---|---|
| **Metrics** | **Count Vectorizer** | **TF-IDF** |
| Precision | 0.8415 | 0.9145 |
| Recall | 0.8434 | 0.9125 |
| F1-Score | 0.8445 | 0.9135 |

*Figure 26: The figure and the table display the performance metrics of the Naive Bayes classifier for 2-word embeddings*

## Error Analysis

Below I have attached the results of the confusion matrix and the ROC curves which is useful to analysis the mispredictions of the classifier. I have tried to analysis the miss predicted data samples to find out the root cause.

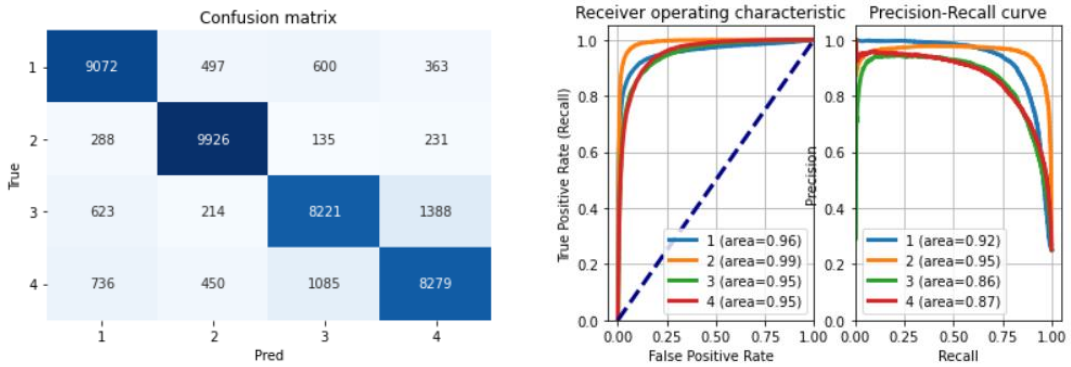



*Figure 27: The confusion matrix and ROC curves results of the classifier is displayed above*

There are miss classifications among all the class but between Business and Science & Technology there are more miss classifications than the other classes.

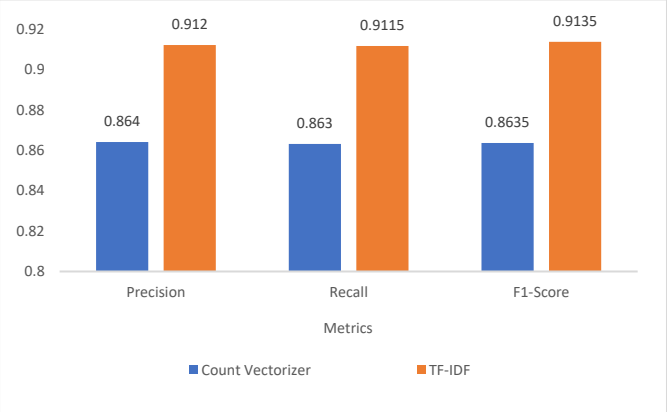
| Actual | Predicted | Text |
|---|---|---|
| Business | Science & Technology | Short-sellers, Wall Street's dwindling band of ultra-cynics, are seeing green again |
| Business | Science & Technology | Tearaway world oil prices, toppling records and straining wallets, present a new economic menace barely three months before the US presidential elections. |
| Science & Technology | Business | America Online on Thursday said it plans to sell a low-priced PC targeting low-income and minority households who agree to sign up for a year of dialup Internet service. |

*Figure 28: The table shows few misclassified samples of the classifier*

Above table contains few samples of news text which got mis-predicted. From my understanding both texts have different meanings, but the vectorization technique which I have used does not take semantic into consideration and works on the word count. Both these categories have similar words. This might be a reason for a heavy misclassification rate between these classes. In deep learning classifiers I will be implement vectorization techniques that works on the semantics of the words. I will analyse how the deep learning methods classify text content in these two classes.

## Support Vector Machine (SVM)

Predominant machine learning approaches like as support vector machines (SVMs) may be used for a variety of tasks, including classification, regression, and even the identification of outliers. High-dimensional spaces are well-suited to SVMs, which are often used in classification tasks of many kinds. There are three classes in Scikit-learn that can do multi-class classification: SVC, NuSVC, and LinearSVC. The implementation is based on libsvm and uses a C-support vector classification system. Scikit-learn makes use of the sklearn.svm.SVC module. In accordance with a one-vs-one system, this class takes care of multiclass support[37]. I have implemented the SVC implementation using the bag of words and TF-IDF implementation and using the optimal parameters identified using the grid search. The TF-IDF implementation yields better results compared to the bag of words embedding. The evaluation metrics can be found in the below tables and charts.




| Support Vector Machine (SVM) | | |
|---|---|---|
| **Metrics** | **Count Vectorizer** | **TF-IDF** |
| Precision | 0.864 | 0.912 |
| Recall | 0.863 | 0.9115 |
| F1-Score | 0.8635 | 0.9135 |

*Figure 29: The figure and the table display the performance metrics of the Support Vector Machine (SVM) classifier for 2-word embeddings*

## Error Analysis

Below I have attached the results of the confusion matrix and the ROC curves which is useful to analysis the mispredictions of the classifier. I have tried to analysis the miss predicted data samples to find out the root cause.

In SVM again the misclassification is high between science and Business classes, but it is slightly better compared to Naïve Bayes classifier.

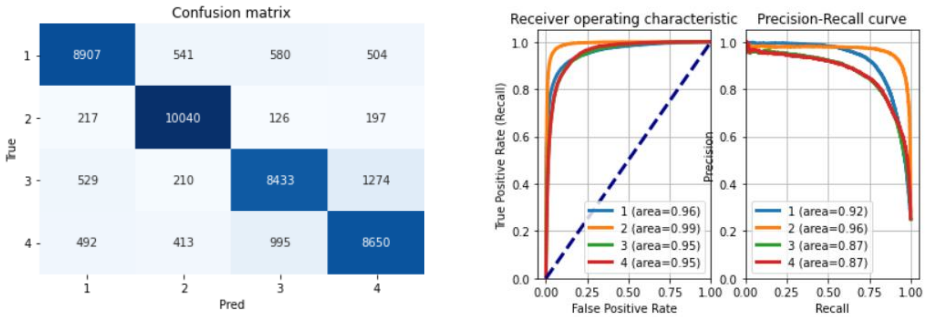
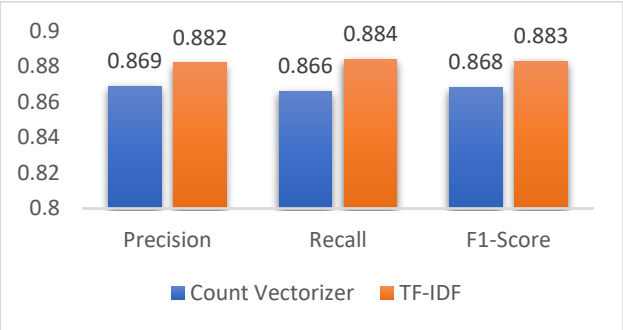



*Figure 30: The confusion matrix and ROC curves results of the classifier is displayed above*

## Logistic Regression

I have built the Logistic Regression using implementations available in sklearn using both Bag of Words and TF-IDF word embedding techniques. I have evaluated the classifiers performance using the evaluation techniques such as precision, recall and F1-score on both embedding techniques implementations. The classifiers seem to perform slightly better using the TF-IDF vectorization technique. The performance metrics and comparisons are mentioned below.




| Logistic Regression | | |
|---|---|---|
| **Metrics** | **Count Vectorizer** | **TF-IDF** |
| Precision | 0.869 | 0.882 |
| Recall | 0.866 | 0.884 |
| F1-Score | 0.868 | 0.883 |

*Figure 31: The figure and the table display the performance metrics of the Logistic Regression classifier for 2-word embeddings*

## Error Analysis

I've included the outputs of the confusion matrix and the ROC curves, which are helpful for analysing the classifier's mispredictions. I attempted to analyse the miss predicted data samples in order to determine the root cause.

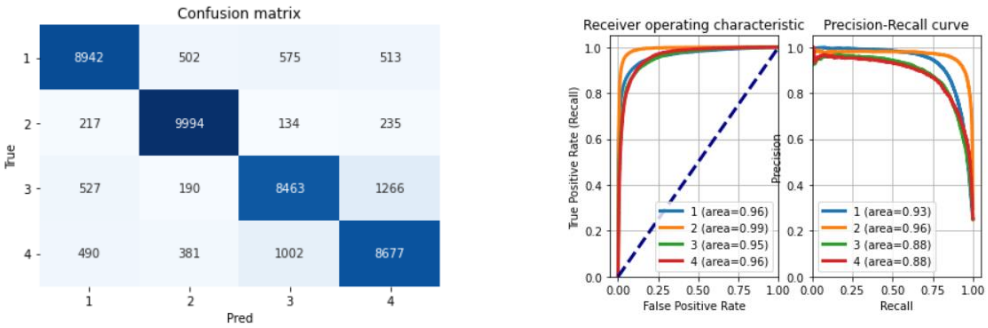
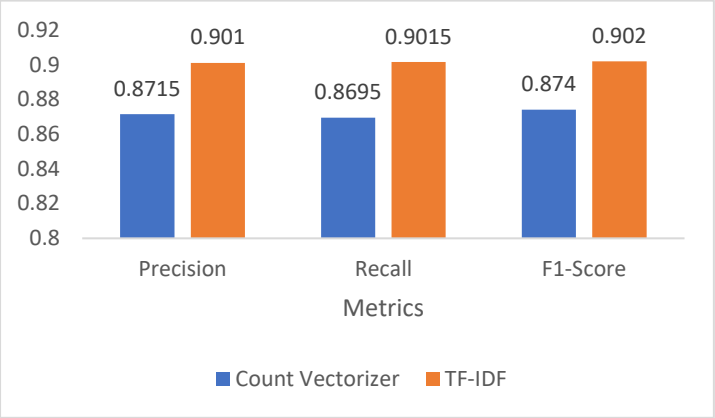



*Figure 32: The confusion matrix and ROC curves results of the classifier is displayed above*

## Bagging

I am using Random Forest one of the most powerful bagging algorithms. Random Forests are improved versions of bagged decision trees in terms of performance. In order to reduce error, decision trees use a greedy method to choose the variable to split on. It's possible that decision trees to have a lot of structural similarity and good correlation in predictions even if you use Bagging. If the predictions from the sub-models are uncorrelated or at the very least weakly correlated, combining them in ensembles performs better[38]. I have used sklearn to build the random forest classifier and built it using optimal parameters obtained using grid search.




| Random Forest | | |
|---|---|---|
| **Metrics** | **Count Vectorizer** | **TF-IDF** |
| Precision | 0.8715 | 0.901 |
| Recall | 0.8695 | 0.9015 |
| F1-Score | 0.874 | 0.9025 |

*Figure 33: The figure and the table display the performance metrics of the Random Forest classifier for 2-word embeddings*

## Error Analysis

I've included the outputs of the confusion matrix and the ROC curves, which are helpful for analysing the classifier's mispredictions. I attempted to analyse the miss predicted data samples in order to determine the root cause.

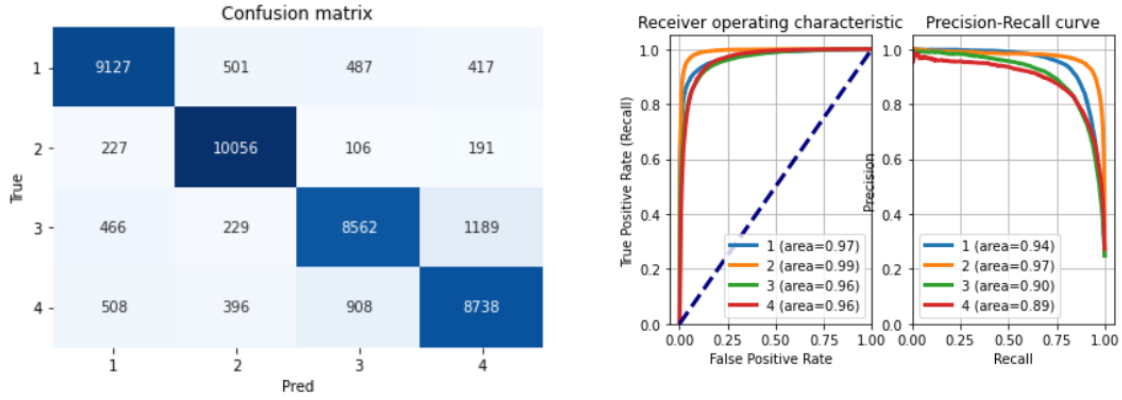
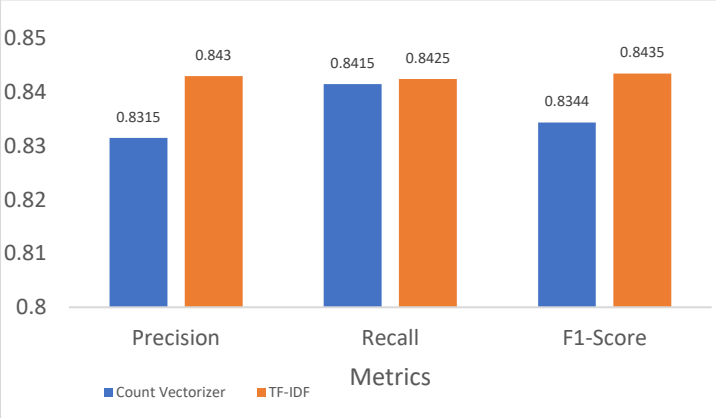



*Figure 34: The confusion matrix and ROC curves results of the classifier is displayed above*

Like other classifier results, using Random Forest classifier the misclassification between Business and Science & Technology classes is high compared to other classes. But the overall misclassification rate between these classes is less compared to other classifiers. Also, the misclassification rate between the world news and sports news is less compared to other classifiers.

## Boosting

Gradient boosting machine (GBM) relies on the intuition that the best possible next model, when combined with previous models, reduces the overall prediction error rate. Due to the sequential nature of GBM, they are generally difficult to train quickly. As a result, they have limited scalability. As a result, XGBoost is primarily concerned with improving model speed and accuracy. I have implemented Gradient Boosting using sklearn implementation, but it took a lot of time to train using the whole train set. So, I have a used a sample of the data to train the GBM model. But we can use XGBoost to improve the computational performance, which I have not done in my project. Gradient Boosting classifiers perform marginally better with TF-IDF embedding compared to bag of words embedding.




| Gradient Boosting | | |
|---|---|---|
| **Metrics** | **Count Vectorizer** | **TF-IDF** |
| Precision | 0.8315 | 0.843 |
| Recall | 0.8415 | 0.8425 |
| F1-Score | 0.8344 | 0.8435 |

*Figure 35: The figure and the table display the performance metrics of the Boosting classifier for 2-word embeddings*

## Error Analysis

I've included the outputs of the confusion matrix and the ROC curves, which are helpful for understanding the classifier results. I attempted to analyse the classifiers performance using its classification metrics.

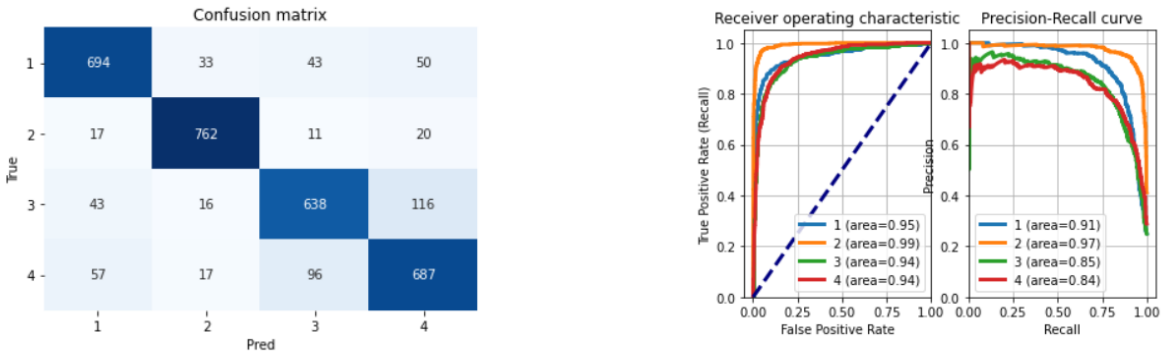



*Figure 36: The confusion matrix and ROC curves results of the classifier is displayed above*

Like all the classifiers even in gradient boosting the misclassification between the business and science and technology classes were high. The GBM classifier has better recall metrics than the precision. These scores would have been better if the classifier had been trained on the whole set of data instead of a small sample.

### 4.2.2 Deep Learning Models

#### Convolutional Neural Network (CNN)

I have used keras with tensorFlow backend to create the CNN models. I have trained the models using two different pre-trained word embedding techniques like Word2Vec and GloVe. I have used the same model architecture but with different embeddings to compare the performance of the embeddings. I have built a CNN model starting with an input layer followed by an embedding layer. Then I have used 3 convolutional filters and the corresponding pooling layers. I have used max pooling. Then I have flattened the vectors obtained from max layer to a fully connected dense layer obtain the classification results. I have used dropout regularisation technique to avoid the overfitting problem. The model summary can be seen below.

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | [(None, 1000)] | 0 | [] |
| embedding (Embedding) | (None, 1000, 300) | 9819000 | ['input_1[0][0]'] |
| reshape (Reshape) | (None, 1000, 300, 1) | 0 | ['embedding[0][0]'] |
| conv2d (Conv2D) | (None, 998, 1, 512) | 461312 | ['reshape[0][0]'] |
| conv2d_1 (Conv2D) | (None, 997, 1, 512) | 614912 | ['reshape[0][0]'] |
| conv2d_2 (Conv2D) | (None, 996, 1, 512) | 768512 | ['reshape[0][0]'] |
| max_pooling2d (MaxPooling2D) | (None, 1, 1, 512) | 0 | ['conv2d[0][0]'] |
| max_pooling2d_1 (MaxPooling2D) | (None, 1, 1, 512) | 0 | ['conv2d_1[0][0]'] |
| max_pooling2d_2 (MaxPooling2D) | (None, 1, 1, 512) | 0 | ['conv2d_2[0][0]'] |
| concatenate (Concatenate) | (None, 3, 1, 512) | 0 | ['max_pooling2d[0][0]', 'max_pooling2d_1[0][0]', 'max_pooling2d_2[0][0]'] |
| flatten (Flatten) | (None, 1536) | 0 | ['concatenate[0][0]'] |
| dropout (Dropout) | (None, 1536) | 0 | ['flatten[0][0]'] |
| dense (Dense) | (None, 5) | 7685 | ['dropout[0][0]'] |

Total params: 11,671,421
Trainable params: 1,852,421
Non-trainable params: 9,819,000

*Figure 37: Model summary of the CNN architecture*

I have used “adam” as the optimizer and categorical cross entropy has the loss function because the target variable has more than 2 classes in it. I have trained the model for 10 epochs and used the batch size as 500 samples each. The training and validation loss and accuracy values are plotted below for the reference. The CNN architecture with GloVe embedding gave better results compared to the CNN model with Word2Vec embedding. The precision and recall scores were better in the GloVe version.

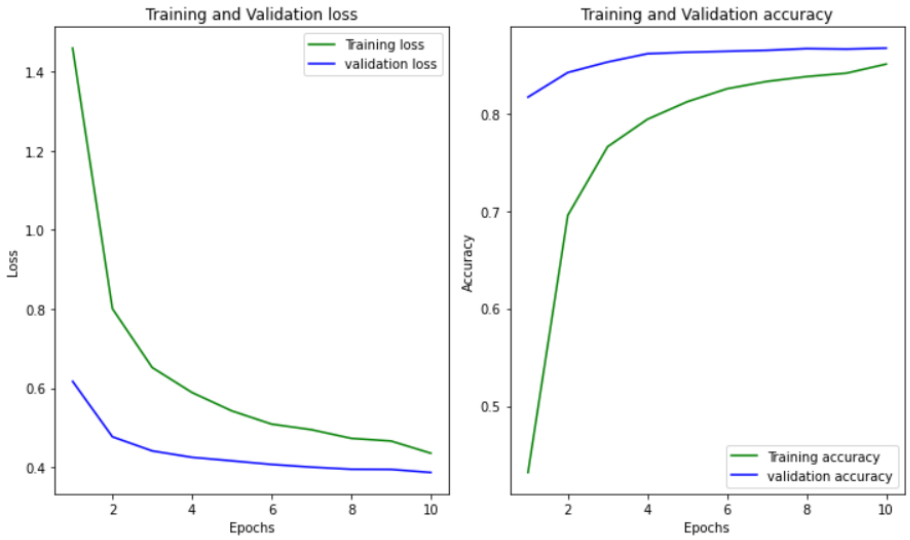



*Figure 38: Train and validation loss and accuracy plots of the CNN model*

## Model Evaluation

Below you can find the precision, recall and F1-score for the CNN classifier for both the word embeddings and the confusion matrix of the CNN model with GloVe embedding.

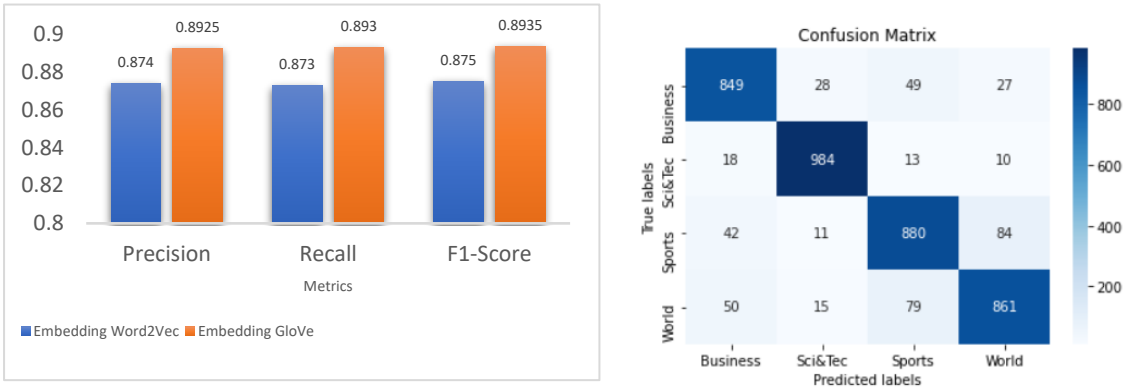



*Figure 39: The precision, recall, F1-score plot and the confusion matrix of the CNN model*

Similar to the machine learning classifiers the misclassification rate was high between the business and science & technology class compared to other classes. Even the word embedding used in the deep learning models take semantic meanings of the words into consideration the misclassification still happens. The reason this can be of the content in these two classes being close each other's. This makes it difficult for the deep learning models to distinguish them accordingly.

## Recurrent Neural Network (RNN) with LSTM

For RNN also I have used keras with tensorFlow backend to create the models. I have trained the models using two different pre-trained word embedding techniques like Word2Vec and GloVe. I have used the same model architecture but with different embeddings to compare the performance of the embeddings. I have built a RNN model starting with an input layer followed

by an embedding layer. Then I have used 2 bidirectional LSTM layers. Then I have flattened the vectors obtained from LSTM layers to corresponding dense layers. Which uses softmax function to obtain the classification results. I have used dropout regularisation technique to avoid the overfitting problem. The model summary can be seen below.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 1000, 300)         9719400

bidirectional_2 (Bidirectio  (None, 1000, 128)         186880
nal)

bidirectional_3 (Bidirectio  (None, 64)                41216
nal)

dense_2 (Dense)              (None, 64)                4160

dropout_1 (Dropout)          (None, 64)                0

dense_3 (Dense)              (None, 5)                 325

=================================================================
Total params: 9,951,981
Trainable params: 232,581
Non-trainable params: 9,719,400
```

*Figure 40: Model summary of the RNN with LSTM architecture*

I have used "adam" as the optimizer and categorical cross entropy has the loss function because the target variable has more than 2 classes in it. I have trained the model for 10 epochs and used the batch size as 500 samples each. The training and validation loss and accuracy values are plotted below for the reference. The CNN architecture with GloVe embedding results was comparatively higher than the Word2Vec embedding. The precision and recall scores were also better in the GloVe version.

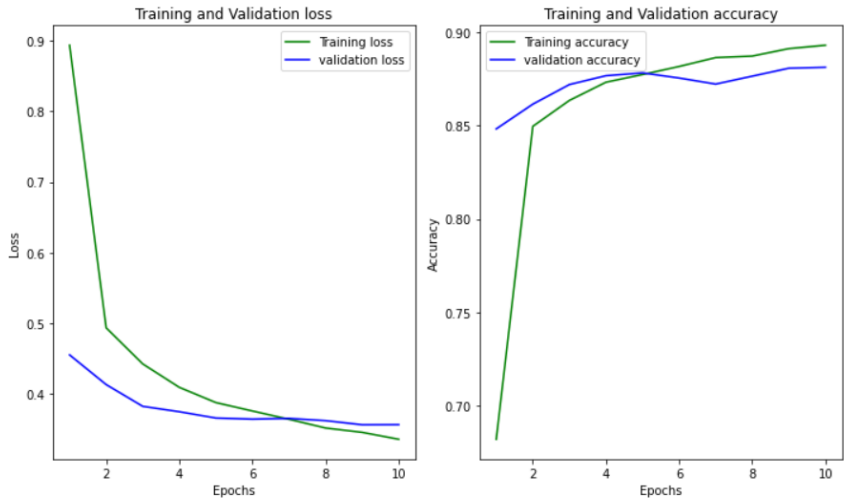



*Figure 41: Train and validation loss and accuracy plots of the RNN model*

## Model Evaluation

Below you can find the precision, recall and F1-score for the CNN classifier for both the word embeddings and the confusion matrix of the CNN model with GloVe embedding.

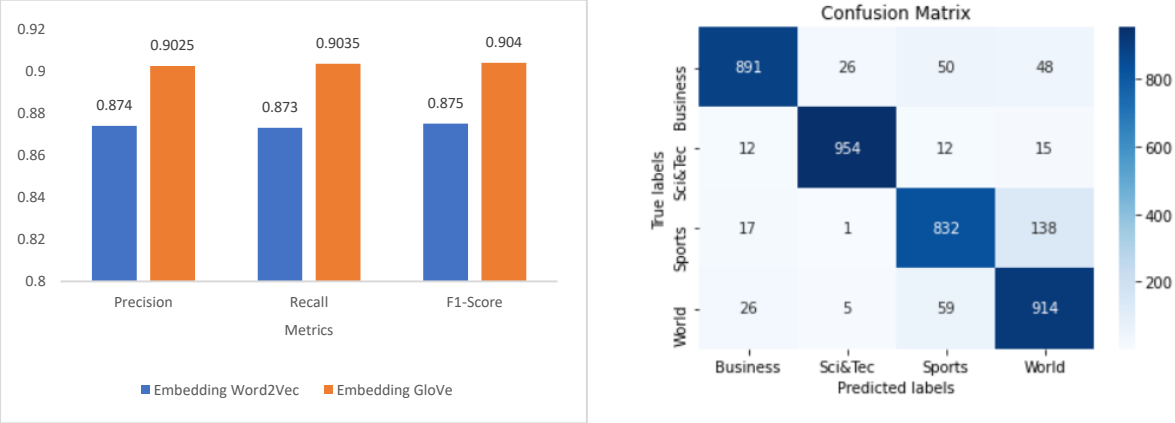



*Figure 42: The precision, recall, F1-score plot and the confusion matrix of the RNN model*

In RNN the precision and recall score has improved compared to the CNN model, but the misclassified samples between the business and science and technology class have increased. The improvement in the precision, recall score is due to the reduced misclassification in the remaining classes.

### Bidirectional Encoder Representations from Transformers (BERT)

I have used one of the pre-trained BERT models and fine-tuned it for my text classification task. I have used the embedding from BERT as embedding layer for my data. I have used the pre-trained model along with few changes in the final layer of the model to adjust it for the classification task. Training Transformer models is heavily dependent on computational resources, so it is not possible to train a BERT model from scratch. The model summary can be seen in the below picture.

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_word_ids (InputLayer)  [(None, 128)]             0           []

input_mask (InputLayer)      [(None, 128)]             0           []

segment_ids (InputLayer)     [(None, 128)]             0           []

keras_layer (KerasLayer)     [(None, 768),             109482241   ['input_word_ids[0][0]',
                              (None, 128, 768)]                      'input_mask[0][0]',
                                                                     'segment_ids[0][0]']

tf.__operators__.getitem (Slic  (None, 768)            0           ['keras_layer[0][1]']
ingOpLambda)

dense (Dense)                (None, 64)                49216       ['tf.__operators__.getitem[0][0]'
                                                                    ]

dropout (Dropout)            (None, 64)                0           ['dense[0][0]']

dense_1 (Dense)              (None, 32)                2080        ['dropout[0][0]']

dropout_1 (Dropout)          (None, 32)                0           ['dense_1[0][0]']

dense_2 (Dense)              (None, 5)                 165         ['dropout_1[0][0]']

==================================================================================================
Total params: 109,533,702
Trainable params: 109,533,701
Non-trainable params: 1
```

*Figure 43: model summary of the BERT pre-trained model*

**Model Evaluation**

I have mentioned the loss function and the optimizer which must be used while training the model which are “adam” and categorical cross entropy, because the target variable has more than 2 classes in it. I have trained the model for 10 epochs and used the batch size as 64 samples each. The training and validation loss and accuracy values are plotted below for the reference. The precision and recall scores can also be found below.

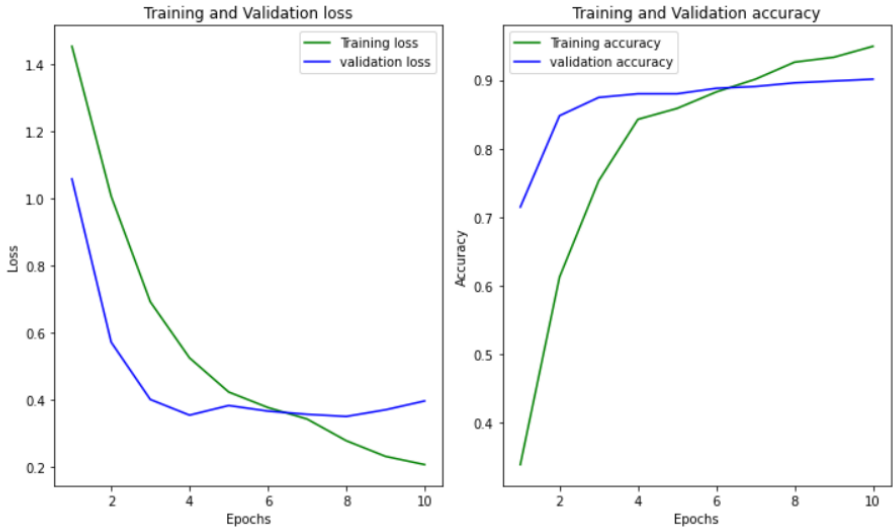



*Figure 44: Training and validation loss and accuracy plot*

From the classification metrics and confusion matrix results attached below, it seems that the classifier has a good precision and recall metrics, but its slightly lesser than the RNN classifier. For a dataset of just 2000 training samples which is trained using a pre-trained model which was developed for generic NLP purpose these metrics are far better. Even by using this pre-trained BERT model the misclassification rate between the business and science and technology classes are not reduced.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Business | 0.88 | 0.84 | 0.86 | 130 |
| Sci&Tec | 0.93 | 0.98 | 0.96 | 113 |
| Sports | 0.84 | 0.88 | 0.86 | 136 |
| World | 0.87 | 0.82 | 0.84 | 121 |
| accuracy | | | 0.88 | 500 |
| macro avg | 0.88 | 0.88 | 0.88 | 500 |
| weighted avg | 0.88 | 0.88 | 0.88 | 500 |

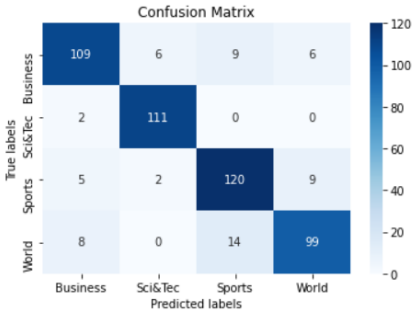



*Figure 45: Precision, recall metrics and confusion matrix for BERT model*

## 4.3 Comparison

Among the machine learning classifiers, the Naïve Bayes, SVM and Random Forest gave better results compared to Logistic Regression and Gradient Boosting with bag of words and TF-IDF embeddings. In general, all the classifiers performed better with TF-IDF embedding. The accuracy of all the machine learning and deep learning classifiers is mentioned below.

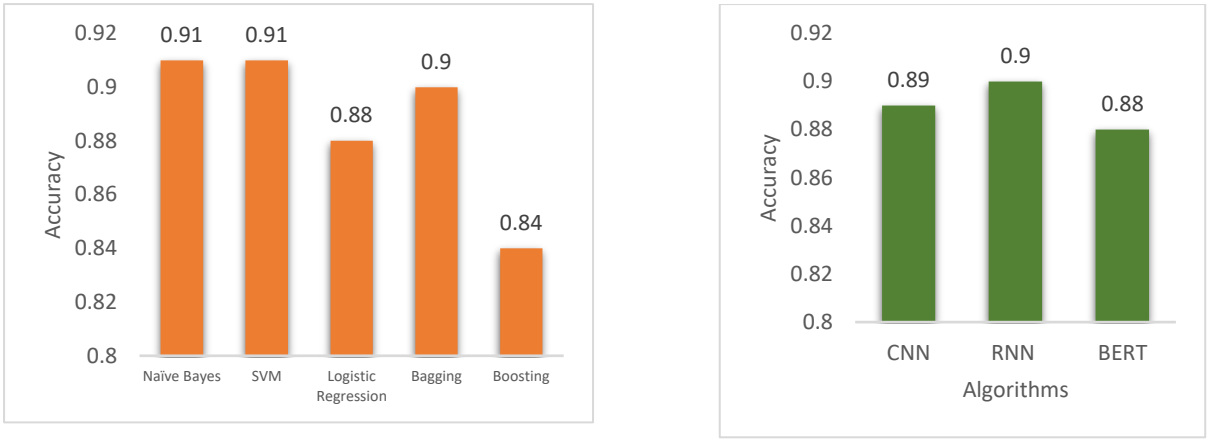



*Figure 46: Comparison of the machine learning and deep learning classifiers*

From the performance metrics of the deep learning models, it is evident that the RNN model yields slightly better results compared to CNN and Transformer models. Both CNN and RNN architecture were build using 5000 samples from each class which is less than 1/3 of the data available. With more training data these models would have yielded even better performance metrics for this classification task. The BERT model is not trained from scratch, the pre-trained model has been used with a few changes in the final layer to fine-tune it for the classification task. Using just 500 samples in each class the results which we got from BERT is very surprising and promising.

From this experiment, it is evident that deep learning methods yield good results in this text classification tasks. The experiment result shows that both machine learning and deep learning models yield similar performance metrics, but the major difference is that the deep learning models are trained with 1/3 of the data that was used to train the machine learning models. It is proven that deep learning models improve with more training data but due to the computational complexity, I was not able to train the deep learning models on the huge corpus of data that was available.

Since text data is sequential in nature the Recurrent Neural Network (RNN) variants provide great results in text classification tasks, but the biggest hiccup in training these models is the computational power.

With large training data and with GPU-based computational resources we can build robust deep learning algorithms like RNN's which can provide great results on these text classification tasks.

In particular, Bidirectional Encoder Representations from Transformers (BERT) models gave great results with a very small training set which comprises 500 samples from each class. Transfer learning with large pre-trained transformer-based language models like BERT has become a dominating approach for most NLP tasks. These BERT models are very useful when we must implement text classification tasks with very little data in hand.

# Chapter 5

# Conclusion

## 5.1 Conclusion

I have presented a report on the performance of the most used machine learning and deep learning algorithms on the multi-class text classification task. My goal was to evaluate and compare different machine learning and deep learning classifiers using different embedding techniques and hyperparameters to find models that perform better on this text classification task. The TF-IDF word embedding outperforms the bag of words word embedding technique across all metrics for all the machine learning classifiers implement and evaluated. And GloVe embedding with all the deep learning architectures gave better results than the Word2Vec word embedding technique. These results may vary if a different set of pre-processing techniques were used, if the data is further analysed carefully more strong set text pre-processing can be done which may improve the results further.

Hyperparameters also play a major role in building a robust classifier. Hyperparameter tuning is a laborious task but needs to be carried out to find the optimal parameters of the algorithms. But there is no promise that a set of parameters that made the classifier perform better in this can good results for some other classification tasks. For each problem statement, the optimal parameters must be found using data of that problem statement. The best choice of classifier always depends on what classifications task it is going to be used for. If the priority lies with not producing false negatives, then the models with good precision scores should be used. If, however, not producing false positives is the highest priority, then a classifier with a good recall score should be used.

From my experiment, I believe that to build a robust classifier for a text classification task other than the algorithms, pre-processing, and word embedding the volume of training data and computational resources also plays a vital role. If we have a large training corpus and computational capability, we can build RNN models or even train transformer-based models from scratch to solve the problem, on the other hand, if we have less training data and computational resources, we can train machine learning models or do transfer learning using pre-trained models like BERT for solving the problem.

Furthermore, the structure of the data that is to be used is also an important factor to take into consideration when choosing the right classifier. The dataset used for this work was perfectly balanced but in the real world, the data will be highly imbalanced for any classification problem. For this dataset, the best overall result was achieved was using Naive Bayes and SVM using the TF-IDF method and RNN with LSTM gave great results using the GloVe word embedding technique. But these may vary for a different classification task. But always the data, pre-processing, and embedding technique plays a very important role in building a best text classification algorithm.

## 5.2 Self Evaluation

It can be difficult to predict the course of a project at the outset. However, a clear work plan and associated timetable are usually necessary. After reading a few articles on various machine learning and deep learning techniques used for text categorization, I decided to get down to work. The most critical step in solving the problem statement was deciding which data to use. During this phase, I gained an understanding of the data I was working with and how to apply it to achieve my objectives. Data pre-processing and feature extraction was aided by my prior expertise working on several machine learning use cases. As a result, I was able to put together a variety of feature extraction techniques that were effective in achieving my aims. To get the best results, I invested a lot of effort in pre-processing and feature extraction.

Advanced deep learning methods and topics in NLP were explored in the last phase of my research study. I made a concerted effort to ensure that the report as a whole was consistent in both its format and its substance. Even though certain issues arose, in general, I believe I was able to stick to my timetable and that my report will be valuable for future study in this area.

## 5.3 Future Work

Future works could include evaluating more ensemble methods such as bagging and boosting to understand how they perform. There are many other variants of deep learning architectures available such as RNN with Gated Recurrent Unit (GRU), Recurrent Convolutional Neural Network (RCNN), and other Variants of Deep Neural Networks which can be built and evaluated for this problem statement. The deep learning models can be explored and evaluated more by using different architectures, optimizers, and different regularizations techniques.

With the recent advancements in the field of NLP with top players like Google and Facebook proposing advanced concepts like transformers with attention mechanisms and state-of-the-art pre-trained models which can be fine-tuned for different problem statements, there are many algorithms and different variants that can be explored to solve a problem. Other transformer-based algorithms and different pre-trained BERT models can be implemented and evaluated for the text classification problem statement. Different BERT models can be explored with ease these days thanks to Hugging Face, a start-up in the Natural Language Processing (NLP) domain, offering its library to all[39].

## Bibliography


[1] "ESSAYS.SE: Comparing Feature Extraction Methods and Effects of Pre-Processing Methods for Multi-Label Classification of Textual Data." https://www.essays.se/essay/99b8b7f33c/ (accessed Dec. 12, 2021).

[2] G. Alain and Y. Bengio, "What regularized auto-encoders learn from the data-generating distribution," *Journal of Machine Learning Research*, vol. 15, pp. 3563–3593, Jan. 2015.

[3] "Keyword Extraction Techniques using Python | by Ajay J Alex | Analytics Vidhya | Medium." https://medium.com/analytics-vidhya/keyword-extraction-techniques-using-python-edea5fc35678 (accessed Dec. 12, 2021).

[4] "An overview of Text Summarization in Natural Language Processing." https://www.impelsys.com/an-overview-of-text-summarization-in-natural-language-processing/ (accessed Dec. 12, 2021).

[5] "A Comprehensive Guide to Understand and Implement Text Classification in Python." https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/ (accessed Dec. 12, 2021).

[6] A. H. Doan *et al.*, "The case for a structured approach to managing unstructured data," *CIDR 2009 - 4th Biennal Conference on Innovative Data Systems Research*, 2009.

[7] "NLP Text Preprocessing: A Practical Guide and Template | by Jiahao Weng | Towards Data Science." https://towardsdatascience.com/nlp-text-preprocessing-a-practical-guide-and-template-d80874676e79 (accessed Dec. 12, 2021).

[8] "Stemming vs Lemmatization. Truncate a word to its root or base… | by Aditya Beri | Towards Data Science." https://towardsdatascience.com/stemming-vs-lemmatization-2daddabcb221 (accessed Dec. 12, 2021).

[9] "A Beginner's Guide to Bag of Words & TF-IDF | Pathmind." https://wiki.pathmind.com/bagofwords-tf-idf (accessed Dec. 12, 2021).

[10] "Bag-of-words vs TFIDF vectorization –A Hands-on Tutorial." https://www.analyticsvidhya.com/blog/2021/07/bag-of-words-vs-tfidf-vectorization-a-hands-on-tutorial/ (accessed Dec. 12, 2021).

[11] H. Liang, X. Sun, Y. Sun, and Y. Gao, "Text feature extraction based on deep learning: a review," *Eurasip Journal on Wireless Communications and Networking*, vol. 2017, no. 1, Dec. 2017, doi: 10.1186/S13638-017-0993-1.

[12] Dipanjan Sarkar, "Implementing Deep Learning Methods and Feature Engineering for Text Data: The Continuous Bag of Words (CBOW)," 2018.

[13] "Word2Vec For Word Embeddings -A Beginner's Guide - Analytics Vidhya." https://www.analyticsvidhya.com/blog/2021/07/word2vec-for-word-embeddings-a-beginners-guide/ (accessed Dec. 12, 2021).

[14] Dipanjan Sarkar, "Implementing Deep Learning Methods and Feature Engineering for Text Data: The Skip-gram Model," 2018.

[15] "A practical explanation of a Naive Bayes classifier." https://monkeylearn.com/blog/practical-explanation-naive-bayes-classifier/ (accessed Dec. 12, 2021).

[16] "Text Classification Using Naive Bayes: Theory & A Working Example | by Serafeim Loukas | Towards Data Science." https://towardsdatascience.com/text-classification-using-naive-bayes-theory-a-working-example-2ef4b7eb7d5a (accessed Dec. 12, 2021).

[17] C. Y. J. Peng, K. L. Lee, and G. M. Ingersoll, “An introduction to logistic regression analysis and reporting,” *Journal of Educational Research*, vol. 96, no. 1, pp. 3–14, 2002, doi: 10.1080/00220670209598786.

[18] “Logistic Regression for Machine Learning.” https://machinelearningmastery.com/logistic-regression-for-machine-learning/ (accessed Dec. 12, 2021).

[19] T. Evgeniou and M. Pontil, “Support vector machines: Theory and applications,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2049 LNAI, pp. 249–257, 2001, doi: 10.1007/3-540-44673-7_12.

[20] H. Tayade, C. Shetty, R. Jankar, and A. Pande, “Segregation of Live News Articles Based on Location Using Machine Learning,” *International Journal of Scientific Research in Computer Science, Engineering and Information Technology © 2020 IJSRCSEIT*, vol. 6, no. 3, pp. 2456–3307, 2020, doi: 10.32628/CSEIT206380.

[21] “Bagging and Random Forest Ensemble Algorithms for Machine Learning.” https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/ (accessed Dec. 12, 2021).

[22] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.

[23] “A Quick Guide to Boosting in ML. This post will guide you through an… | by Jocelyn D’Souza | GreyAtom | Medium.” https://medium.com/greyatom/a-quick-guide-to-boosting-in-ml-acf7c1585cb5 (accessed Dec. 12, 2021).

[24] Y. Lecun *et al.*, “Deep learning for natural language processing: advantages and challenges,” *National Science Review*, vol. 5, no. 1, pp. 24–26, Jan. 2018, doi: 10.1093/NSR/NWX110.

[25] Y. Theis and W. Hsu, “Learning to detect named entities in bilingual code-mixed open speech corpora”.

[26] L. Alzubaidi *et al.*, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *Journal of Big Data 2021 8:1*, vol. 8, no. 1, pp. 1–74, Mar. 2021, doi: 10.1186/S40537-021-00444-8.

[27] “Introduction to how CNNs Work. Introduction to how CNNs work | by Simran Bansari | DataDrivenInvestor.” https://medium.datadriveninvestor.com/introduction-to-how-cnns-work-77e0e4cde99b (accessed Dec. 12, 2021).

[28] “Recurrent Neural Networks (RNN): What It Is & How It Works | Built In.” https://builtin.com/data-science/recurrent-neural-networks-and-lstm (accessed Dec. 12, 2021).

[29] H. H. Sak, A. Senior, and B. Google, “Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling”.

[30] “Text classification with an RNN | TensorFlow.” https://www.tensorflow.org/text/tutorials/text_classification_rnn (accessed Dec. 12, 2021).

[31] “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding | Papers With Code.” https://paperswithcode.com/paper/bert-pre-training-of-deep-bidirectional (accessed Dec. 13, 2021).

[32] “BERT: State of the Art NLP Model, Explained - KDnuggets.” https://www.kdnuggets.com/2018/12/bert-sota-nlp-model-explained.html (accessed Dec. 13, 2021).

[33] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*

- *Proceedings of the Conference*, vol. 1, pp. 4171–4186, Oct. 2018, Accessed: Dec. 13, 2021. [Online]. Available: https://arxiv.org/abs/1810.04805v2

[34] "A Comprehensive Guide to Understand and Implement Text Classification in Python." https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/ (accessed Dec. 13, 2021).

[35] "The 5 Classification Evaluation metrics every Data Scientist must know | by Rahul Agarwal | Towards Data Science." https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226 (accessed Dec. 12, 2021).

[36] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159, Jul. 1997, doi: 10.1016/S0031-3203(96)00142-2.

[37] "sklearn.svm.SVC — scikit-learn 1.0.1 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (accessed Dec. 12, 2021).

[38] "Bagging Research Papers - Academia.edu." https://www.academia.edu/Documents/in/Bagging (accessed Dec. 12, 2021).

[39] "Hugging Face – The AI community building the future." https://huggingface.co/ (accessed Dec. 13, 2021).

# Appendix

## A1 Installing and running the code

The code developed can be found in directory code/, which constitutes four jupyter notebook files with the implementation of machine learning and deep learning models. code depends on the following Python packages: Numpy, SciPy, Scikit-learn, Matplotlib, TensorFlow. The code was written and tested on windows using Python version 3.7.

pip

It is a useful tool to easily install Python packages without the need of visiting package website or manually cloning remote repositories. It can be installed by following the instructions at http://pip.readthedocs.org/en/

NumPy, SciPy

Can be installed by downloading binaries from http://www.scipy.org/

scipylib/download.html. Otherwise, by using pip, from a command line:

pip install --user NumPy

and

pip install --user SciPy

Each line will raise an exception only if the package is not installed. After all the requirements are installed, the code will run, and the results will be seen below the cell. Below I have attached a screen shot which shows results of a cell which is ran and results are displayed.

In [1]:
```
import pandas as pd
import numpy as np
import nltk
## for plotting
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
data['Class Index'] = data['Class Index'].astype('str')
classes = {"1" : "World",
                    "2" : "Sports",
                    "3" : "Business",
                    "4" : "Sci&Tech"}

data['Class'] = data['Class Index'].map(classes)
data
```

Out[3]:

| | Class Index | Title | Description | Class |
|---|---|---|---|---|
| 0 | 3 | Wall St. Bears Claw Back Into the Black (Reuters) | Reuters - Short-sellers, Wall Street's dwindli... | Business |
| 1 | 3 | Carlyle Looks Toward Commercial Aerospace (Reu... | Reuters - Private investment firm Carlyle Grou... | Business |
| 2 | 3 | Oil and Economy Cloud Stocks' Outlook (Reuters) | Reuters - Soaring crude prices plus worries\ab... | Business |
| 3 | 3 | Iraq Halts Oil Exports from Main Southern Pipe... | Reuters - Authorities have halted oil export\f... | Business |
| 4 | 3 | Oil prices soar to all-time record, posing new... | AFP - Tearaway world oil prices, toppling reco... | Business |

## A2 Download the data and embedding

### Data

The data which was used to train the models can be downloaded from the official site which has a DB version and xml version of the AG news article corpus.

http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html.

I have used the csv version of the data which is available in separate train and test csv files which can be downloaded from the link below.

https://www.kaggle.com/amananandrai/ag-news-classification-dataset

Since the size of the dataset very large, uploading the data and embedding along with the report will exceed 100MB (maximum upload limit), only the links to download the data and the embeddings are mentioned.

### Embeddings

The GloVe pre-trained embedding which was used while building model can be found in the link below.

https://nlp.stanford.edu/projects/glove/

There are many versions with different dimensions are available I have used the following embedding glove.6B.100d.txt

The Word2Vec pre-trained embedding which was used while building model can be found in the link below.

https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit

There are many versions with different dimensions are available I have used the following embedding glove.6B.100d.txt GoogleNews-vectors-negative300.bin

Once the data and the embedding are downloaded the path in the code as to be replaced with the new file path and the code can be run.

The code contains the all the results of the machine learning and deep learning models which was already ran, and the model results can be found in the code's output consoles. If needed the code can be ran again to obtain the results again.

The report contains the detailed comparison of these deep learning and machine learning models using their evaluation metrics. The also contains detailed analysis of these algorithms and their performance on this text classification task.