DIFFERENT APPROACHES ON NEWS ARTICLES CLASSIFICATION

Subhadip Maity

24.04.2021

Kalyani University

ABSTRACT

Text classification is a popular NLP task that assigns one or more categories to a given piece of text from a larger set of possible categories. In this project, we classify news articles into five different categories viz. 'Sport', 'business', 'politics', 'tech' and 'entertainment'. From basic vectorization approaches, like Bag of Words, TF-IDF, different word embeddings like Word2vec and Doc2vec have been used as advanced vectorization techniques. As well as simple ML classification algorithms, a few deep learning models were employed along with some pre-trained models for these news articles classification tasks. Almost every model performed very well with an accuracy score of more than 90. Finally, we interpreted and explained the classifier's predictions with the Lime and the Shap package.

KEY-WORDS: NLP,TEXT CLASSIFICATION,DOCUMENT CLASSIFICATION,TEXT MINING

1 INTRODUCTION

The machine learning approach to text classification has proven to give good results in numerous articles. In machine learning, classification categorizes a data instance into one or more known classes. The data point can be original of different formats, such as text, speech, image, or numeric. Text classification is a special instance of the classification problem, where the input data point(s) is text and the goal is to categorize the piece of text into one or more buckets (called a class) from a set of predefined buckets (classes). The “text” can be of arbitrary length: a character, a word, a sentence, a paragraph, or a full document. The challenge of text classification is to “learn” this categorization from a collection of examples for each of these categories and predict the categories for new, unseen text data. Text classification is sometimes also referred to as topic classification, text categorization, or document categorization.

Pipeline for Building Text Classification Systems

One typically follows these steps when building a text classification system:

1. Collect or create a labeled dataset suitable for the task.
2. Split the dataset into two (training and test) or three parts: training, validation (i.e., development), and test sets, then decide on evaluation metric(s).
3. Transform raw text into feature vectors.
4. Train a classifier using the feature vectors and the corresponding labels from the training set.
5. Using the evaluation metric(s) from Step 2, benchmark the model performance on the test set.
6. Deploy the model to serve the real-world use case and monitor its performance.

Below is the flowchart of the text classification system.

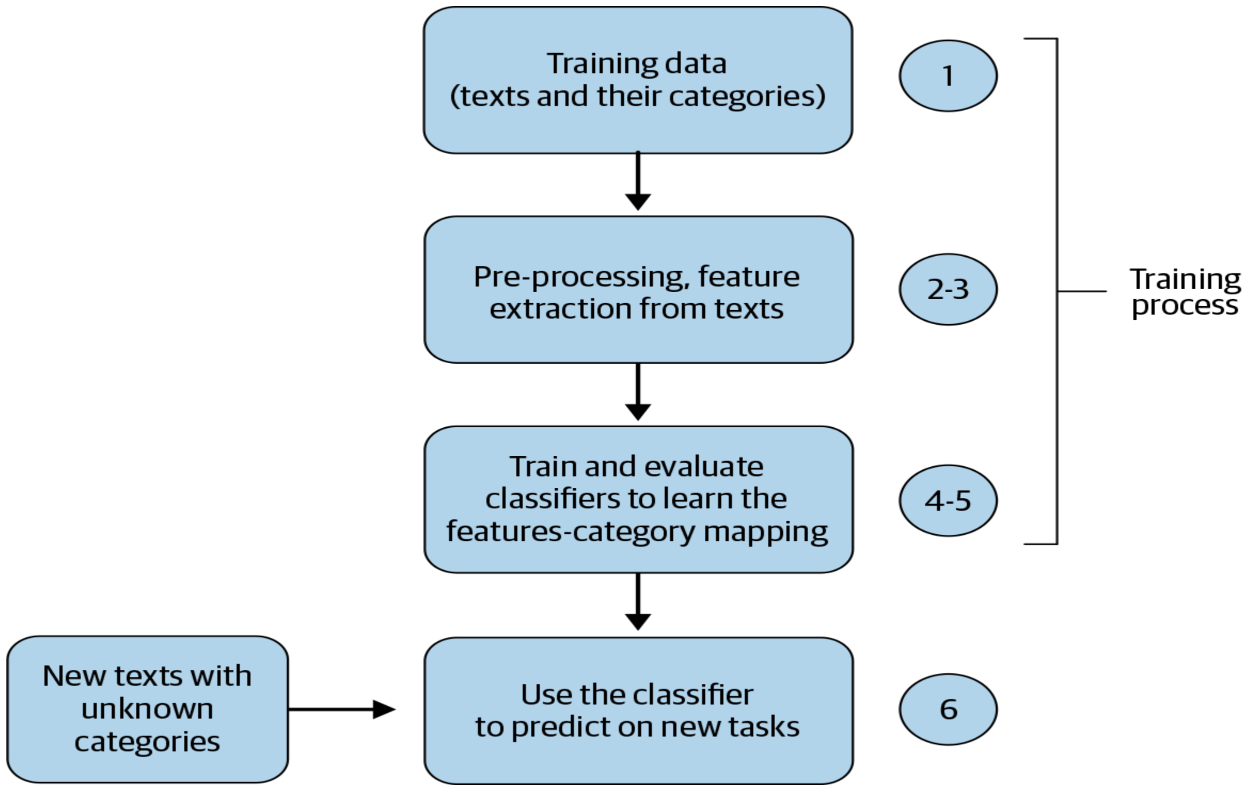

Training data
(texts and their categories)
1
Pre-processing, feature extraction from texts
2-3
Train and evaluate classifiers to learn the features-category mapping
4-5
Training process
New texts with unknown categories
Use the classifier to predict on new tasks
6

Figure showing a general flow chart of the text classification system

Problem Definition

Here in this project, we want to classify some given news articles into 5 different categories:'Sport', 'business', 'politics', 'tech', and 'entertainment'.Since we have labeled data and five different news article categories it is a multiclass supervised classification problem.

Approaches

After the data acquisition, the very first task is text preprocessing. In different classification algorithms, we used different preprocessing techniques like sentence segmentation, word tokenization, stop word removal, stemming and lemmatization, removing digits/punctuation, lowercasing, etc.

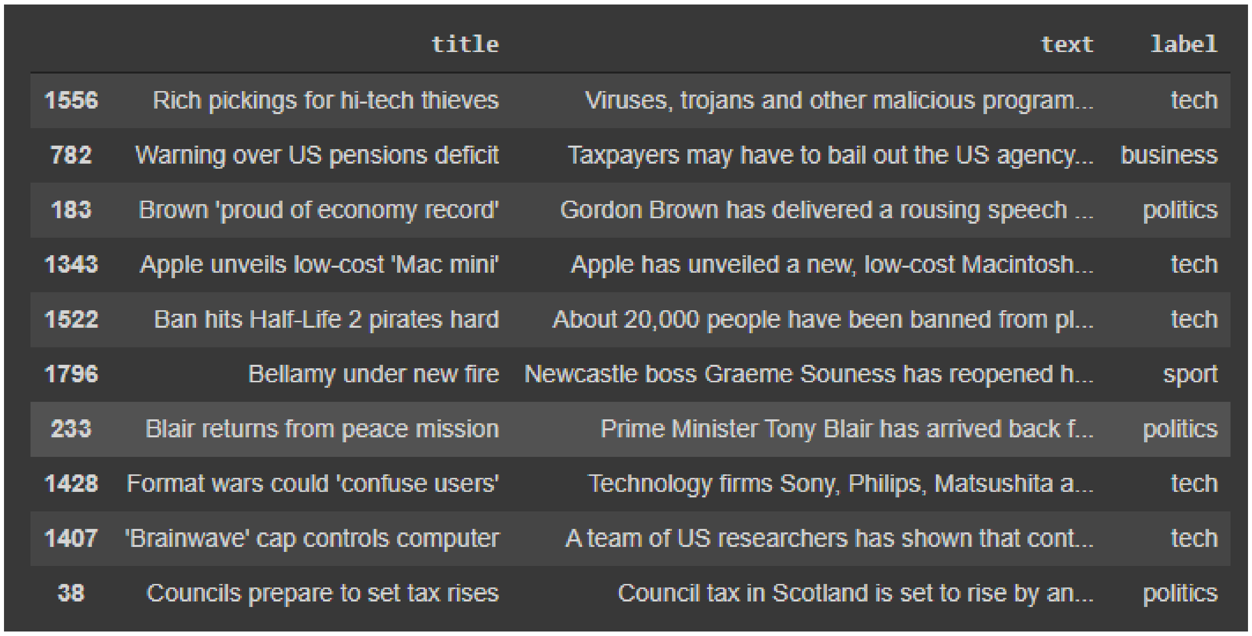
Next, the most challenging part of any NLP task is feature extraction or text representation that is to transform the text into numerical form so that it can be fed into the downstream ML model. We used Bag of Words and TF-IDF as vector space models for text representation. Word2vec and Doc2vec were also implemented for word embeddings.

After feature extraction, as baseline ML models, we used Naive Bayes Classifier, Logistic Regression classifier, and Support Vector Machine. For advanced deep learning algorithms, we selected CNN and LSTM. We used several pre-trained models like Glove, FastText, Bert, etc. Every approach achieved a very good result.

Finally, we came up with the interpretation and explanation for one particular prediction with the help of Lime and Shap packages.

2 DATASET

Initially, our data set was a zipped file containing many text files. We converted it into a pandas data frame with columns ‘title’, ‘text’, and ‘label’ where ‘title’ is the headline of news, ‘text’ is the main body of the news, and ‘label’ is one of the labeled categories of the news. We have a total of 2225 news articles categorized into 5 different classes viz. ‘Sport’, ‘business’, ‘politics’, ‘tech’ and ‘entertainment’. Let’s see few rows of our data set.

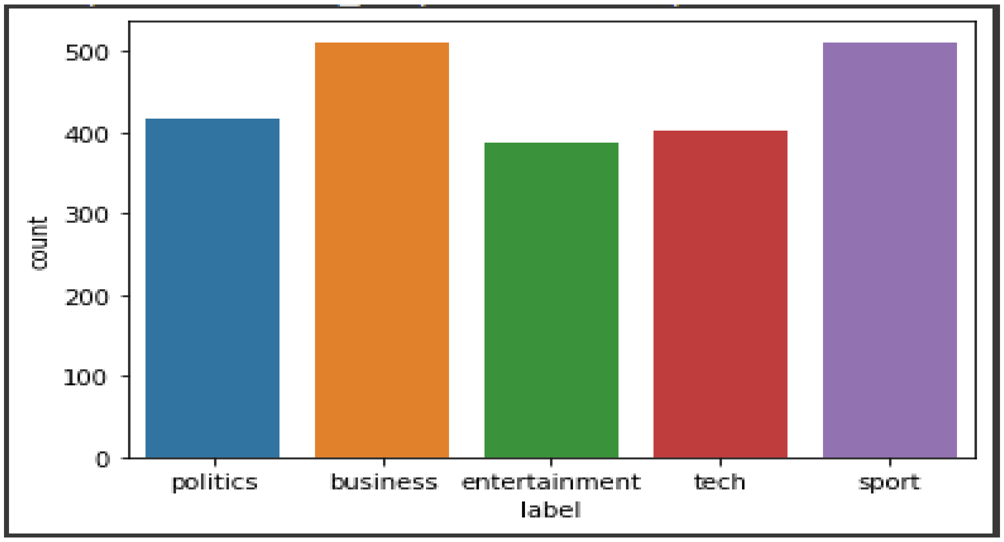


	title	text	label
1556	Rich pickings for hi-tech thieves	Viruses, trojans and other malicious program...	tech
782	Warning over US pensions deficit	Taxpayers may have to bail out the US agency...	business
183	Brown 'proud of economy record'	Gordon Brown has delivered a rousing speech ...	politics
1343	Apple unveils low-cost 'Mac mini'	Apple has unveiled a new, low-cost Macintosh...	tech
1522	Ban hits Half-Life 2 pirates hard	About 20,000 people have been banned from pl...	tech
1796	Bellamy under new fire	Newcastle boss Graeme Souness has reopened h...	sport
233	Blair returns from peace mission	Prime Minister Tony Blair has arrived back f...	politics
1428	Format wars could 'confuse users'	Technology firms Sony, Philips, Matsushita a...	tech
1407	'Brainwave' cap controls computer	A team of US researchers has shown that cont...	tech
38	Councils prepare to set tax rises	Council tax in Scotland is set to rise by an...	politics

Figure showing few rows of the data frame

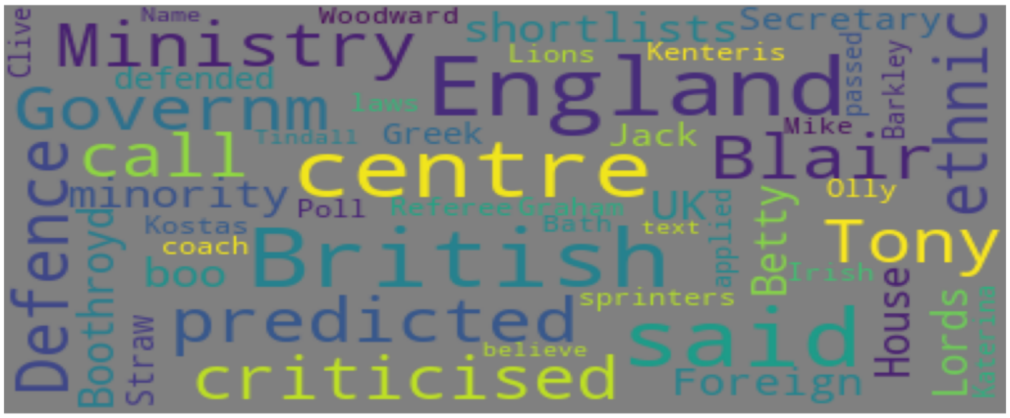
The number of news articles on different categories:

```
sport          511
business         510
politics        417
tech           401
entertainment    386
```

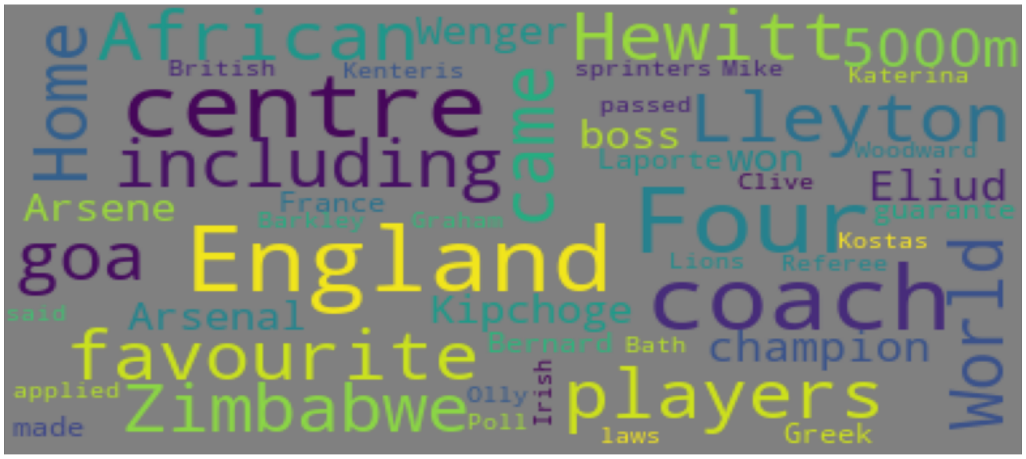


Bar diagram of the different categories of the news articles

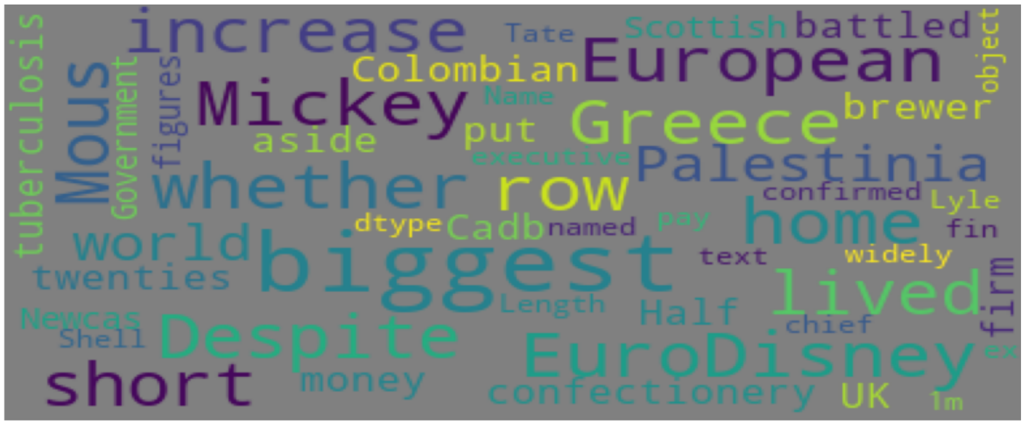
Visualization through word clouds to see the most frequent words:



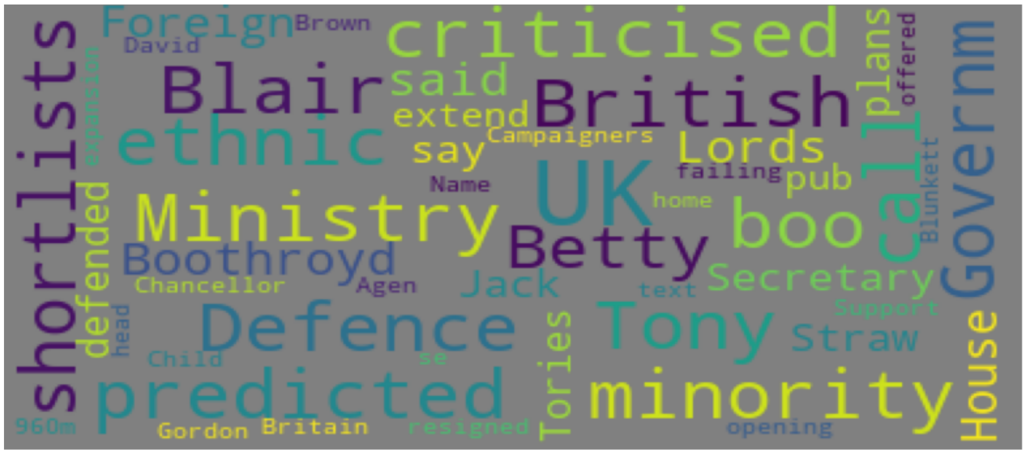
Wordcloud for all news articles categories



Wordcloud for sport related news



Wordcloud for business related news



Wordcloud for politics related news

Technologies
Te
launched
hunting
poor
lawsuit
scam
Theft
series
Dav
avoid
UK
Google
continue
games
mobile
Literally
users
chatrooms
interest
net
face
se
Name
go
rich
owners
writers
told
Apple
break
putting
digital
Grand
Soon
search
mail
giant
technology
text
Auto
records
Computer
divide
issued
web
via
firm
trading
Virus

Wordcloud for tech related news

festive
held
Minister
perform
Potter
Tony
brought
release
REM
BBC
awards
give
industry
Slayer
legend
film
purchased
copi
Fockers
US
watchdogs
urged
Vampire
next
third
Rings
Harry
Meet
Blair
band
announced
Prime
creator
plans
Dustin
Hollywood
Vibe
mor
Comedy
chang
Lord
reggae
topped
will
hit
Hoffman
Buffy
two
refuse
MPs

Wordcloud for entertainment related news

3 TEXT REPRESENTATION

Feature extraction is an important step for any machine learning problem. No matter how good a modeling algorithm you use, if you feed in poor features, you will get poor results. In computer science, this is often called "garbage in, garbage out." In NLP parlance, this conversion of raw text to a suitable numerical form is called text representation.

3.1 Basic Vectorization Approaches

We started with a basic idea of text representation: map each word in the vocabulary (V) of the text corpus to a unique ID (integer value), then represent each sentence or document in the corpus as a V-dimensional vector.

3.1.1 Bag of Words

Bag of words (BoW) is a classical text representation technique that has been used commonly in NLP, especially in text classification problems. The key idea behind it is as follows: represent the text under consideration as a bag (collection) of words while ignoring the order and context. The basic intuition behind it is that it assumes that the text belonging to a given class in the dataset is characterized by a unique set of words. If two text pieces have nearly the same words, then they belong to the same bag (class). Thus, by analyzing the words present in a piece of text, one can identify the class (bag) it belongs to.

Similar to one-hot encoding, BoW maps words to unique integer IDs between 1 and |V|. Each document in the corpus is then converted into a vector of |V| dimensions where, in the i^{th} component of the vector, $i = w_{id}$, is simply the number of times the word w occurs in the document, i.e., we simply score each word in V by their occurrence count in the document.

In this project, we have used Scikitlearn's CountVectorizer for implementing Bag of Words. Initially, the dimension of our feature vector was 31195. Then for better performance, we experimented with reduced feature space with the additional parameter 'max_features=5000'. So, the dimension of the feature vector was 5000.

3.1.2 TF-IDF

In BoW, there's no notion of some words in the document being more important than others.TF-IDF, or term frequency-inverse document frequency, addresses this issue. It aims to quantify the importance of a given word relative to other words in the document and the corpus. It's a commonly used representation scheme for information-retrieval systems, for extracting relevant documents from a corpus for a given text query.

The intuition behind TF-IDF is as follows: if a word w appears many times in a document d_i but does not occur much in the rest of the documents d_j in the corpus, then the word w must be of great importance to the document d_i. The importance of w should increase in proportion to its frequency in d_i, but at the same time, its importance should decrease in proportion to the word's frequency in other documents d_j in the corpus. Mathematically, this is captured using two quantities: TF and IDF. The two are then combined to arrive at the TF-IDF score.

TF (term frequency) measures how often a term or word occurs in a given document. Since different documents in the corpus may be of different lengths, a term may occur more often in a longer document as compared to a shorter document. To normalize these counts, we divide the number of occurrences by the length of the document. TF of a term t in document d is defined as:

$$\text{TF(t,d)=}\frac{(Number\ of\ occurrences\ of\ term\ t\ in\ document\ d)}{(Total\ number\ of\ terms\ in\ the\ document\ d)}$$

IDF (inverse document frequency) measures the importance of the term across a corpus. In computing TF, all terms are given equal importance (weightage). However, it's a well-known fact that stop words like is, are, am, etc., are not important, even though they occur frequently. To account for such cases, IDF weighs down the terms that are very common across a corpus and weighs up the rare terms. IDF of a term t is calculated as follows:

$$\text{IDF(t)=}\ \log_e\frac{(Total\ number\ of\ documents\ in\ the\ corpus)}{(Number\ of\ documents\ with\ term\ t\ in\ them\ )}$$

The TF-IDF score is a product of these two terms. Thus, TF-IDF score = TF * IDF. Let's compute TF-IDF scores for our toy corpus. Some terms appear in only one document, some appear in two, while others appear in three documents.

In this project, we have used Scikitlearn's TfidfVectorizer for implementing Bag of Words. Initially, the dimension of our feature vector was 31195. Then for better performance, we experimented with reduced feature space with the additional parameter 'max_features=5000'. So, the dimension of the feature vector was 5000.

3.2 Distributed Representations

Some key drawbacks are common to all basic vectorization approaches. To overcome these limitations, methods to learn low-dimensional representations were devised. These methods gained momentum in the past six to seven years. They use neural network architectures to create dense, low-dimensional representations of words and texts.

3.2.1 Word2vec

Conceptually, Word2vec takes a large corpus of text as input and "learns" to represent the words in a common vector space based on the contexts in which they appear in the corpus. Given a word w and the words appearing in its context C, how do we find the vector that best represents the meaning of the word? For every word w in the corpus, we start with a vector v_w initialized with random values. The Word2vec model refines the values in v_w by predicting v_w, given the vectors for words in the context C. It does this using a two-layer neural network. We'll dive deeper into this by discussing pre-trained embeddings before moving on to train our own.

3.2.1.1 PRE-TRAINED WORD EMBEDDINGS

Training our own word embeddings is a pretty expensive process (in terms of both time and computing). Thankfully, for many scenarios, it's not necessary to train our own embeddings, and using pre-trained word embeddings often suffices. What are pre-trained word embeddings? Someone has done the hard work of training word embeddings on a large corpus, such as Wikipedia, news articles, or even the entire web, and has put words and their corresponding vectors on the web. These embeddings can be downloaded and used to get the vectors for the words you want. Such embeddings can be thought of as a large collection of key-value pairs, where keys are the words in the vocabulary and values are their corresponding word vectors. Some of the most popular pre-trained embeddings are Word2vec by Google, GloVe by Stanford, and fasttext embeddings by Facebook, to name a few. Further, they're available for various dimensions like d = 25, 50, 100, 200, 300, 600.In this project, all the above-mentioned pre-trained models have been used.

3.2.1.2 TRAINING OUR OWN EMBEDDINGS

Now focused on training our own word embeddings. For this, we looked at two architectural variants that were proposed in the original Word2vec approach. The two variants are:

- Continuous bag of words (CBOW)
- SkipGram

3.2.2 Doc2vec

Doc2vec is based on the paragraph vectors framework and is implemented in Gensim. This is similar to Word2vec in terms of its general architecture, except that, in addition to the word vectors, it also learns a "paragraph vector" that learns a representation for the full text (i.e., with words in context). When learning with a large corpus of many texts, the paragraph vectors are unique for a given text (where "text" can mean any piece of text of arbitrary length), while word vectors will be shared across all texts. The shallow neural networks used to learn Doc2vec embeddings are very similar to the CBOW and SkipGram architecture of Word2vec.

3.3 BERT

Neural architectures such as recurrent neural networks (RNNs) and transformers were used to develop a large-scale model of language like Bert, which can be used as pre-trained models to get text representations. The key idea is to leverage “transfer learning”—that is, to learn embeddings on a generic task (like language modeling) on a massive corpus and then fine-tune learnings on task-specific data. In this text classification task, we have used the Bert model in two different ways.

4 DIFFERENT CLASSIFICATION APPROACHES AND THEIR RESULTS

4.1 Basic ML Classifiers with one common pipeline

We built a classification system with a common pipeline but different ML classifiers. We mapped different categories of news articles to numeric values as 'sport':1, 'business':2,'politics':3,'tech':4,'entertainment':5. Then as a text preprocessing technique, we are performing the following steps: removing br tags, punctuation, numbers, and stopwords. We used Scikitlearn’s train test split method to split our data set into the training(75%) and testing(25%) parts i.e. we had 1668 news articles for training and 557 for testing.

4.1.1 Naive Bayes Classifier

Naive Bayes is a probabilistic classifier that uses Bayes’ theorem to classify texts based on the evidence seen in training data. It estimates the conditional probability of each feature of a given text for each class based on the occurrence of that feature in that class and multiplies the probabilities of all the features of a given text to compute the final probability of classification for each class. Finally, it chooses the class with maximum probability.

Using BoW vectorization (with the dimension of the feature vector 31195) we get the following result:

```
Accuracy:  0.9748653500897666
                precision    recall  f1-score   support

           1       1.00      0.99      1.00       118
           2       0.98      0.95      0.97       133
           3       0.95      0.99      0.97       108
           4       0.98      0.98      0.98        93
           5       0.97      0.96      0.97       105

    accuracy                           0.97       557
   macro avg       0.97      0.98      0.97       557
weighted avg       0.98      0.97      0.97       557
```

Using BoW vectorization (with the dimension of the feature vector 5000) we get the following result:

```
Accuracy:  0.9748653500897666
              precision    recall  f1-score   support

           1       1.00      0.99      1.00       118
           2       0.97      0.96      0.97       133
           3       0.97      0.98      0.98       108
           4       0.97      0.96      0.96        93
           5       0.96      0.98      0.97       105

    accuracy                           0.97       557
   macro avg       0.97      0.97      0.97       557
weighted avg       0.97      0.97      0.97       557
```

Using TF-IDF vectorization (with the dimension of the feature vector 31195) we get the following result:

```
Accuracy:  0.9676840215439856
              precision    recall  f1-score   support

           1       0.98      0.99      0.99       118
           2       0.95      0.98      0.96       133
           3       0.95      0.99      0.97       108
           4       0.99      0.95      0.97        93
           5       0.98      0.92      0.95       105

    accuracy                           0.97       557
   macro avg       0.97      0.97      0.97       557
weighted avg       0.97      0.97      0.97       557
```

Using TF-IDF vectorization (with the dimension of the feature vector 5000) we get the following result:

```
Accuracy:  0.9694793536804309
              precision    recall  f1-score   support
```

```
           1       0.99      0.99      0.99       118
           2       0.96      0.96      0.96       133
           3       0.95      0.98      0.97       108
           4       0.98      0.95      0.96        93
           5       0.97      0.96      0.97       105

    accuracy                           0.97       557
   macro avg       0.97      0.97      0.97       557
weighted avg       0.97      0.97      0.97       557
```

4.1.2 Logistic Regression

Logistic regression is an example of a discriminative classifier and is commonly used in text classification, as a baseline in research, and as an MVP in real-world industry scenarios. Unlike Naive Bayes, which estimates probabilities based on feature occurrence in classes, logistic regression "learns" the weights for individual features based on how important they are to make a classification decision. The goal of logistic regression is to learn a linear separator between classes in the training data to maximize the probability of the data. This "learning" of feature weights and a probability distribution over all classes is done through a function called "logistic" function, and (hence the name) logistic regression.

Using BoW vectorization (with the dimension of the feature vector 31195) we get the following result:

```
Accuracy:  0.9712746858168761
              precision    recall  f1-score   support

           1       0.97      0.99      0.98       118
           2       0.98      0.97      0.97       133
           3       0.96      0.95      0.96       108
           4       0.99      0.98      0.98        93
           5       0.96      0.96      0.96       105

    accuracy                           0.97       557
   macro avg       0.97      0.97      0.97       557
weighted avg       0.97      0.97      0.97       557
```

Using BoW vectorization (with the dimension of the feature vector 5000) we get the following result:

```
Accuracy:  0.9748653500897666
```

```
              precision    recall  f1-score   support

           1       0.97      0.99      0.98       118
           2       0.97      0.98      0.97       133
           3       0.97      0.95      0.96       108
           4       0.99      0.99      0.99        93
           5       0.97      0.96      0.97       105

    accuracy                           0.97       557
   macro avg       0.98      0.97      0.98       557
weighted avg       0.97      0.97      0.97       557
```

Using TF-IDF vectorization (with the dimension of the feature vector 31195) we get the following result:

```
Accuracy:  0.9784560143626571
              precision    recall  f1-score   support

           1       1.00      0.99      1.00       118
           2       0.98      0.97      0.97       133
           3       0.99      0.96      0.98       108
           4       0.97      0.98      0.97        93
           5       0.95      0.99      0.97       105

    accuracy                           0.98       557
   macro avg       0.98      0.98      0.98       557
weighted avg       0.98      0.98      0.98       557
```

Using TF-IDF vectorization (with the dimension of the feature vector 5000) we get the following result:

```
Accuracy:  0.9748653500897666
              precision    recall  f1-score   support

           1       1.00      0.99      1.00       118
           2       0.97      0.97      0.97       133
           3       0.97      0.97      0.97       108
           4       0.97      0.97      0.97        93
           5       0.96      0.97      0.97       105

    accuracy                           0.97       557
   macro avg       0.97      0.97      0.97       557
weighted avg       0.97      0.97      0.97       557
```

4.1.3 Support Vector Machine

A support vector machine (SVM) is a discriminative classifier like logistic regression. However, unlike logistic regression, it aims to look for an optimal hyperplane in a higher-dimensional space, which can separate the classes in the data by a maximum possible margin. Further, SVMs are capable of learning even non-linear separations between classes, unlike logistic regression. However, they may also take longer to train.

Using BoW vectorization (with the dimension of the feature vector 31195) we get the following result:

```
Accuracy:  0.9820466786355476
              precision    recall  f1-score   support

           1       1.00      0.99      1.00       118
           2       0.98      0.98      0.98       133
           3       0.99      0.98      0.99       108
           4       0.99      0.97      0.98        93
           5       0.95      0.99      0.97       105

    accuracy                           0.98       557
   macro avg       0.98      0.98      0.98       557
weighted avg       0.98      0.98      0.98       557
```

Using BoW vectorization (with the dimension of the feature vector 5000) we get the following result:

```
Accuracy:  0.9766606822262118
              precision    recall  f1-score   support

           1       1.00      0.99      1.00       118
           2       0.96      0.98      0.97       133
           3       0.98      0.98      0.98       108
           4       0.99      0.96      0.97        93
           5       0.96      0.97      0.97       105

    accuracy                           0.98       557
   macro avg       0.98      0.98      0.98       557
weighted avg       0.98      0.98      0.98       557
```

Using TF-IDF vectorization (with the dimension of the feature vector 31195) we get the following result:

```
Accuracy:  0.9856373429084381
              precision    recall  f1-score   support
```

```
           1       1.00      0.99      1.00       118
           2       0.98      0.97      0.97       133
           3       0.99      0.99      0.99       108
           4       0.99      0.99      0.99        93
           5       0.97      0.99      0.98       105

    accuracy                           0.99       557
   macro avg       0.99      0.99      0.99       557
weighted avg       0.99      0.99      0.99       557
```

Using TF-IDF vectorization (with the dimension of the feature vector 5000) we get the following result:

```
Accuracy:  0.9838420107719928
              precision    recall  f1-score   support

           1       1.00      0.99      1.00       118
           2       0.98      0.97      0.97       133
           3       0.98      1.00      0.99       108
           4       0.99      0.99      0.99        93
           5       0.97      0.97      0.97       105

    accuracy                           0.98       557
   macro avg       0.98      0.98      0.98       557
weighted avg       0.98      0.98      0.98       557
```

4.2 Word2vec with pre-trained Google News vectors

Loading and pre-processing the text data remains a common step. However, instead of vectorizing the texts using BoW-based features, we'll now rely on neural embedding models. As mentioned earlier, we'll use a pre-trained embedding model. There are several pre-trained Word2vec models trained on large corpora available on the internet. Here, we have used the one from Google. This is a large model that can be seen as a dictionary where the keys are words in the vocabulary and the values are their learned embedding representations. Given a query word, if the word's embedding is present in the dictionary, it will return the same. There are multiple ways to use this pre-learned embedding to represent features. We have used a simple approach of averaging the embeddings for individual words in the text. Note that it uses embeddings only for the words that are present in the dictionary. It ignores the words for which embeddings are absent. Also, note that averaging created a single vector with DIMENSION(=300) components. After this feature engineering, the final step is similar to what we did in the previous section. When trained with a logistic regression classifier, these features gave a classification accuracy of 95% on our dataset. Following is the classification report of this approach

```
Accuracy:  0.9569120287253142
              precision    recall  f1-score   support

           1       0.98      1.00      0.99       124
           2       0.96      0.95      0.95       132
           3       0.91      0.93      0.92       102
           4       0.93      0.95      0.94       104
           5       0.99      0.95      0.97        95

    accuracy                           0.96       557
   macro avg       0.96      0.96      0.96       557
weighted avg       0.96      0.96      0.96       557
```

4.3 Subword embeddings with FastText

Word2vec suffers from the fact of out of vocabulary(OOV) problem i.e. if a word in our dataset was not present in the pre-trained model's vocabulary, it can not represent this word. Fasttext embeddings by Facebook provide a solution. They're based on the idea of enriching word embeddings with subword-level information. Thus, the embedding representation for each word is represented as a sum of the representations of individual character n-grams. While this may seem like a longer process compared to just estimating word-level embeddings, it has two advantages:

- This approach can handle words that did not appear in training data (OOV).
- The implementation facilitates extremely fast learning on even very large corpora.

While fastText is a general-purpose library to learn the embeddings, it also supports off-the-shelf text classification by providing end-to-end classifier training and testing; i.e., we don't have to handle feature extraction separately. The training and test sets are provided as CSV files in this dataset. Using Sklearn's train test split we split the data set into two CSV files train and test. Train file contains 80% of the data set i.e. a total of 1780 news articles and the rest of the news articles i.e. 445 news articles are contained in the test CSV file. So, the first step involves reading these files into your Python environment and cleaning the text to remove extraneous characters, similar to what we did in the pre-processing steps for the other classifier examples we've seen so far. Once this is done, the process to use fastText is quite simple. Fasttext is extremely fast to train and achieves a very good recall score for our classification task. The result of the 4 iterations are as follows :

```
Test Samples: 445 Precision@1 : 96.8539 Recall@1 : 96.8539
Test Samples: 445 Precision@2 : 49.6629 Recall@2 : 99.3258
Test Samples: 445 Precision@3 : 33.3333 Recall@3 : 100.0000
Test Samples: 445 Precision@4 : 25.0000 Recall@4 : 100.0000
Test Samples: 445 Precision@5 : 20.0000 Recall@5 : 100.0000
```

4.4 Document Embeddings

In the Doc2vec embedding scheme, we learn a direct representation for the entire document (sentence/paragraph) rather than each word. Just as we used word and character embeddings as features for performing text classification, we can also use Doc2vec as a feature representation mechanism. Since there are no existing pre-trained models that work with the latest version of Doc2vec, We have built our own Doc2vec model and use it for text classification. Data preparation and preprocessing of the text and splitting the data set into training, testing remains the same as earlier approaches. The first part of this process involves converting the data into a format readable by the Doc2vec implementation, which can be done using the TaggedDocument class. It's used to represent a document as a list of tokens, followed by a "tag," which in its simplest form can be just the filename or ID of the document. However, Doc2vec by itself can also be used as the nearest neighbor classifier for both multiclass and multilabel classification problems using. We have used the model parameters as:

```
 model = Doc2Vec(vector_size=50, alpha=0.025, min_count=5, dm =1,
epochs=100)
```

vector_size refers to the dimensionality of the learned embeddings; alpha is the learning rate; min_count is the minimum frequency of words that remain in vocabulary; dm, which stands for distributed memory, is one of the representation learners implemented in Doc2vec (the other is dbow, or distributed bag of words); and epochs are the number of training iterations. There are a few other parameters that can be customized.Doc2vec's infer_vector function can be used to infer the vector representation for a given text using a pre-trained model. Since there is some amount of randomness due to the choice of hyperparameters, the inferred vectors differ each time we extract them. For this reason, to get a stable representation, we ran it multiple times (called steps) and aggregated the vectors. We have used the learned model to infer features for our data and train a logistic regression classifier. The performance of the model is as follows:

```
precision    recall  f1-score   support

           1       0.99      0.99      0.99       116
           2       0.87      0.97      0.92       134
           3       0.93      0.82      0.87       104
           4       0.97      0.91      0.94       114
           5       0.89      0.93      0.91        89

    accuracy                           0.93       557
   macro avg       0.93      0.92      0.93       557
weighted avg       0.93      0.93      0.93       557

Accuracy:  0.9281867145421903
```

4.5 Deep Learning Models

Over the past few years, it has shown remarkable improvements on standard machine learning tasks, such as image classification, speech recognition, and machine translation. This has resulted in widespread interest in using deep learning for various tasks, including text classification. Two of the most commonly used neural network architectures for text classification are convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Long short-term memory (LSTM) networks are a popular form of RNNs. In this project, we tried both CNN and LSTM. After splitting the data set as earlier methods, we implemented the following steps to convert training and test data into a format suitable for the neural network input layers:

1. Tokenize the texts and convert them into word index vectors.
2. Pad the text sequences so that all text vectors are of the same length.
3. Map every word index to an embedding vector. We do that by multiplying word index vectors with the embedding matrix. The embedding matrix can either be populated using pre-trained embeddings or it can be trained for embeddings on this corpus.
4. Use the output from Step 3 as the input to neural network architecture.

Once these are done, we can proceed with the specification of neural network architectures and training classifiers with them. We have used Keras, a Python-based DL library. For pre-trained embeddings we converted the train and test data into an embedding matrix as we did in the earlier examples with Word2vec and fastText, we have to download them and use them to convert our data into the input format for the neural networks. Here we have done with pre-trained GloVe with 100 as dimension. The input layer is an embedding layer and the output layer is a softmax layer as our problem is a multiclass classification. We have defined a CNN with three convolution-pooling layers using the Sequential model class in Keras, which allows us to specify DL models as a sequential stack of layers—one after another. For CNN we set the parameters as loss='categorical_crossentropy',optimizer='rmsprop',metrics=accuracy'.We set optimizer='adam' for LSTM.For both CNN and LSTM, we set the number of epochs equals 15. We got the following results

4.5.1 1D CNN Model with pre-trained embedding

```
Define a 1D CNN model.

Epoch 1/15

12/12 [==============================] - 34s 97ms/step - loss: 1.9515 -
acc: 0.2444 - val_loss: 1.3973 - val_acc: 0.3876

Epoch 2/15

12/12 [==============================] - 0s 39ms/step - loss: 1.2513 -
acc: 0.4832 - val_loss: 0.5584 - val_acc: 0.8764

Epoch 3/15

12/12 [==============================] - 0s 40ms/step - loss: 0.5515 -
acc: 0.8179 - val_loss: 0.7505 - val_acc: 0.6826

Epoch 4/15

12/12 [==============================] - 0s 40ms/step - loss: 0.3593 -
acc: 0.8858 - val_loss: 0.3885 - val_acc: 0.8876

Epoch 5/15

12/12 [==============================] - 0s 39ms/step - loss: 0.2774 -
acc: 0.9007 - val_loss: 0.3506 - val_acc: 0.8652

Epoch 6/15

12/12 [==============================] - 0s 39ms/step - loss: 0.1937 -
acc: 0.9345 - val_loss: 0.6473 - val_acc: 0.7865

Epoch 7/15

12/12 [==============================] - 0s 39ms/step - loss: 0.2640 -
acc: 0.9135 - val_loss: 0.2763 - val_acc: 0.9129

Epoch 8/15

12/12 [==============================] - 0s 39ms/step - loss: 0.1682 -
acc: 0.9387 - val_loss: 0.1086 - val_acc: 0.9663

Epoch 9/15

12/12 [==============================] - 0s 39ms/step - loss: 0.0860 -
acc: 0.9691 - val_loss: 0.1549 - val_acc: 0.9522
```

```
Epoch 10/15

12/12 [==============================] - 0s 39ms/step - loss: 0.1420 -
acc: 0.9633 - val_loss: 0.4118 - val_acc: 0.8792

Epoch 11/15

12/12 [==============================] - 0s 40ms/step - loss: 0.1375 -
acc: 0.9606 - val_loss: 0.5160 - val_acc: 0.8539

Epoch 12/15

12/12 [==============================] - 0s 38ms/step - loss: 0.2980 -
acc: 0.9308 - val_loss: 0.6392 - val_acc: 0.8343

Epoch 13/15

12/12 [==============================] - 0s 39ms/step - loss: 0.2726 -
acc: 0.9347 - val_loss: 0.0575 - val_acc: 0.9888

Epoch 14/15

12/12 [==============================] - 0s 39ms/step - loss: 0.0230 -
acc: 0.9971 - val_loss: 0.0982 - val_acc: 0.9579

Epoch 15/15

12/12 [==============================] - 0s 39ms/step - loss: 0.0167 -
acc: 0.9986 - val_loss: 0.0732 - val_acc: 0.9803

14/14 [==============================] - 0s 12ms/step - loss: 0.1089 -
acc: 0.9573

Test accuracy with CNN: 0.9573033452033997
```

4.5.2 1D CNN model with training our own embedding

```
Defining and training a CNN model, training embedding layer on the fly
instead of using pre-trained embeddings

Epoch 1/15

12/12 [==============================] - 2s 98ms/step - loss: 1.6144 -
acc: 0.2084 - val_loss: 1.6037 - val_acc: 0.2219

Epoch 2/15

12/12 [==============================] - 1s 74ms/step - loss: 1.5887 -
acc: 0.2571 - val_loss: 1.5111 - val_acc: 0.4298

Epoch 3/15

12/12 [==============================] - 1s 75ms/step - loss: 1.3779 -
acc: 0.4156 - val_loss: 0.9612 - val_acc: 0.6096
```

```
Epoch 4/15

12/12 [==============================] - 1s 75ms/step - loss: 0.7992 -
acc: 0.6551 - val_loss: 0.6104 - val_acc: 0.7444

Epoch 5/15

12/12 [==============================] - 1s 74ms/step - loss: 0.4952 -
acc: 0.8072 - val_loss: 0.3394 - val_acc: 0.8876

Epoch 6/15

12/12 [==============================] - 1s 74ms/step - loss: 0.3335 -
acc: 0.8861 - val_loss: 0.2443 - val_acc: 0.9298

Epoch 7/15

12/12 [==============================] - 1s 75ms/step - loss: 0.0756 -
acc: 0.9922 - val_loss: 0.9671 - val_acc: 0.7163

Epoch 8/15

12/12 [==============================] - 1s 88ms/step - loss: 0.3689 -
acc: 0.8524 - val_loss: 0.1704 - val_acc: 0.9382

Epoch 9/15

12/12 [==============================] - 1s 74ms/step - loss: 0.0135 -
acc: 0.9992 - val_loss: 0.4593 - val_acc: 0.8596

Epoch 10/15

12/12 [==============================] - 1s 74ms/step - loss: 0.0130 -
acc: 1.0000 - val_loss: 0.1561 - val_acc: 0.9466

Epoch 11/15

12/12 [==============================] - 1s 74ms/step - loss: 0.0019 -
acc: 1.0000 - val_loss: 0.1628 - val_acc: 0.9522

Epoch 12/15

12/12 [==============================] - 1s 74ms/step - loss: 9.0633e-04 -
acc: 1.0000 - val_loss: 0.1355 - val_acc: 0.9494

Epoch 13/15

12/12 [==============================] - 1s 75ms/step - loss: 3.8545e-04 -
acc: 1.0000 - val_loss: 0.1439 - val_acc: 0.9522

Epoch 14/15

12/12 [==============================] - 1s 75ms/step - loss: 1.6840e-04 -
acc: 1.0000 - val_loss: 0.1453 - val_acc: 0.9551
```

```
Epoch 15/15

12/12 [==============================] - 1s 76ms/step - loss: 7.4340e-05 -
acc: 1.0000 - val_loss: 0.1443 - val_acc: 0.9551

14/14 [==============================] - 0s 11ms/step - loss: 0.2343 -
acc: 0.9416

Test accuracy with CNN: 0.9415730237960815
```

4.5.3 LSTM Model with training our own embedding

```
Training the RNN

Epoch 1/15

45/45 [==============================] - 153s 3s/step - loss: 1.5991 -
accuracy: 0.2436 - val_loss: 1.3158 - val_accuracy: 0.4579

Epoch 2/15

45/45 [==============================] - 153s 3s/step - loss: 1.2323 -
accuracy: 0.5011 - val_loss: 0.8974 - val_accuracy: 0.6264

Epoch 3/15

45/45 [==============================] - 151s 3s/step - loss: 0.6322 -
accuracy: 0.8421 - val_loss: 0.8893 - val_accuracy: 0.6657

Epoch 4/15

45/45 [==============================] - 151s 3s/step - loss: 0.4049 -
accuracy: 0.9157 - val_loss: 0.5836 - val_accuracy: 0.8006

Epoch 5/15

45/45 [==============================] - 151s 3s/step - loss: 0.1341 -
accuracy: 0.9853 - val_loss: 0.4863 - val_accuracy: 0.8511

Epoch 6/15

45/45 [==============================] - 152s 3s/step - loss: 0.0375 -
accuracy: 0.9902 - val_loss: 0.2910 - val_accuracy: 0.9101

Epoch 7/15
```

```
45/45 [==============================] - 151s 3s/step - loss: 0.0053 -
accuracy: 1.0000 - val_loss: 0.2211 - val_accuracy: 0.9466

Epoch 8/15

45/45 [==============================] - 150s 3s/step - loss: 0.0029 -
accuracy: 1.0000 - val_loss: 0.2294 - val_accuracy: 0.9354

Epoch 9/15

45/45 [==============================] - 148s 3s/step - loss: 0.0034 -
accuracy: 1.0000 - val_loss: 0.2671 - val_accuracy: 0.9270

Epoch 10/15

45/45 [==============================] - 148s 3s/step - loss: 0.0024 -
accuracy: 1.0000 - val_loss: 0.2915 - val_accuracy: 0.9101

Epoch 11/15

45/45 [==============================] - 148s 3s/step - loss: 0.0024 -
accuracy: 1.0000 - val_loss: 0.2827 - val_accuracy: 0.9270

Epoch 12/15

45/45 [==============================] - 147s 3s/step - loss: 0.0018 -
accuracy: 1.0000 - val_loss: 0.2260 - val_accuracy: 0.9438

Epoch 13/15

45/45 [==============================] - 146s 3s/step - loss: 0.0046 -
accuracy: 0.9988 - val_loss: 0.6297 - val_accuracy: 0.8118

Epoch 14/15

45/45 [==============================] - 146s 3s/step - loss: 0.0274 -
accuracy: 0.9986 - val_loss: 0.4213 - val_accuracy: 0.8764

Epoch 15/15

45/45 [==============================] - 146s 3s/step - loss: 0.0067 -
accuracy: 1.0000 - val_loss: 0.4059 - val_accuracy: 0.8708

14/14 [==============================] - 3s 212ms/step - loss: 0.4893 -
accuracy: 0.8764
```

```
Test accuracy with RNN: 0.8764045238494873
```

4.5.4 LSTM Model using pre-trained Embedding Layer

```
Training the RNN

Epoch 1/15

45/45 [==============================] - 135s 3s/step - loss: 1.4461 -
accuracy: 0.3766 - val_loss: 0.4655 - val_accuracy: 0.8455

Epoch 2/15

45/45 [==============================] - 135s 3s/step - loss: 0.6568 -
accuracy: 0.7649 - val_loss: 1.4735 - val_accuracy: 0.5590

Epoch 3/15

45/45 [==============================] - 136s 3s/step - loss: 0.7171 -
accuracy: 0.7707 - val_loss: 0.3881 - val_accuracy: 0.8736

Epoch 4/15

45/45 [==============================] - 134s 3s/step - loss: 0.5668 -
accuracy: 0.8118 - val_loss: 0.5175 - val_accuracy: 0.8427

Epoch 5/15

45/45 [==============================] - 135s 3s/step - loss: 0.4986 -
accuracy: 0.8524 - val_loss: 0.8271 - val_accuracy: 0.7584

Epoch 6/15

45/45 [==============================] - 132s 3s/step - loss: 0.7851 -
accuracy: 0.7499 - val_loss: 0.3434 - val_accuracy: 0.9073

Epoch 7/15

45/45 [==============================] - 133s 3s/step - loss: 0.5214 -
accuracy: 0.8456 - val_loss: 0.3797 - val_accuracy: 0.8961

Epoch 8/15

45/45 [==============================] - 133s 3s/step - loss: 0.3177 -
accuracy: 0.9145 - val_loss: 0.2315 - val_accuracy: 0.9382
```

```
Epoch 9/15

45/45 [==============================] - 135s 3s/step - loss: 0.3197 -
accuracy: 0.8976 - val_loss: 0.2669 - val_accuracy: 0.9242

Epoch 10/15

45/45 [==============================] - 135s 3s/step - loss: 0.2097 -
accuracy: 0.9473 - val_loss: 0.5168 - val_accuracy: 0.8062

Epoch 11/15

45/45 [==============================] - 134s 3s/step - loss: 0.3374 -
accuracy: 0.8811 - val_loss: 0.2271 - val_accuracy: 0.9466

Epoch 12/15

45/45 [==============================] - 134s 3s/step - loss: 0.1867 -
accuracy: 0.9454 - val_loss: 0.2302 - val_accuracy: 0.9354

Epoch 13/15

45/45 [==============================] - 134s 3s/step - loss: 0.1874 -
accuracy: 0.9483 - val_loss: 0.2042 - val_accuracy: 0.9242

Epoch 14/15

45/45 [==============================] - 133s 3s/step - loss: 0.2222 -
accuracy: 0.9362 - val_loss: 0.2624 - val_accuracy: 0.9298

Epoch 15/15

45/45 [==============================] - 134s 3s/step - loss: 0.1582 -
accuracy: 0.9519 - val_loss: 0.1437 - val_accuracy: 0.9663

14/14 [==============================] - 3s 215ms/step - loss: 0.2127 -
accuracy: 0.9348

Test accuracy with RNN: 0.934831440448761.
```

4.6 Pre-Trained Language Model BERT

We use BERT, a pre-Trained NLP model open-sourced by Google in late 2018 that can be used for Transfer Learning on textual data. We have also used ktrain, a lightweight wrapper to train and use pre-trained DL models using the TensorFlow library Keras. ktrain provides a straightforward process for all steps, from obtaining the dataset and the pre-trained BERT to fine-tuning it for the classification task. For BERT, We have got the following result

```
Epoch:   0%|          | 0/4 [00:00<?, ?it/s]Train loss: 0.4282651076403757

Epoch:  25%|██▌       | 1/4 [00:38<01:55, 38.40s/it]Validation Accuracy:
0.9776785714285714

Train loss: 0.11378734011853499

Epoch:  50%|█████     | 2/4 [01:16<01:16, 38.38s/it]Validation Accuracy:
0.9821428571428571

Train loss: 0.06280491074975876

Epoch:  75%|███████▌  | 3/4 [01:54<00:38, 38.28s/it]Validation Accuracy:
0.9821428571428571

Train loss: 0.024466724118350873

Epoch: 100%|██████████| 4/4 [02:32<00:00, 38.22s/it]Validation Accuracy:
0.9776785714285714
```

For ktrain, We have the following result

```
begin training using onecycle policy with max lr of 2e-05...

Epoch 1/4

334/334 [==============================] - 364s 1s/step - loss: 1.2441 -
accuracy: 0.5311 - val_loss: 0.1531 - val_accuracy: 0.9462

Epoch 2/4

334/334 [==============================] - 345s 1s/step - loss: 0.0745 -
accuracy: 0.9810 - val_loss: 0.0774 - val_accuracy: 0.9821

Epoch 3/4

334/334 [==============================] - 345s 1s/step - loss: 0.0279 -
accuracy: 0.9925 - val_loss: 0.0762 - val_accuracy: 0.9821

Epoch 4/4

334/334 [==============================] - 346s 1s/step - loss: 0.0060 -
accuracy: 0.9993 - val_loss: 0.0759 - val_accuracy: 0.9776

<tensorflow.python.keras.callbacks.History at 0x7fd245836e10>
```

5 Interpreting Text Classification Models

As ML models started getting deployed in real-world applications, interest in the direction of model interpretability grew. Recent research resulted in usable tools for interpreting model predictions (especially for classification). Lime is one such tool that attempts to interpret a black-box classification model by approximating it with a linear model locally around a given training instance. The advantage of this is that such a linear model is expressed as a weighted sum of its features and easy to interpret. For example, if there are two features, f1 and f2, for a given test instance of a binary classifier with classes A and B, a Lime linear model around this instance could be something like -0.3 × f1 + 0.4 × f2 with a prediction B. This indicates that the presence of feature f1 will negatively affect this prediction (by 0.3) and skew it toward A explains this in more detail. Let's now look at how Lime can be used to understand the predictions of a text classifier.

Let's take a few examples. At first, We have used the Lime on Logistic Regression Classifier we previously used. Here, We have taken an example news article. The article is as follows :

```
“The former head of US medical services firm HealthSouth overstated
earnings and assets to boost the company's share price, it was
claimed in court.  Richard Scrushy, 52, is accused of "directing" a
$2.7bn (£1.4bn) accounting fraud at the company he co-founded in
Alabama in 1984. Prosecutors said he was motivated by wealth -
spending about $200m between 1996 and 2002 while earning much less.
Defense lawyers said Mr. Scrushy had been deceived by other
executives. Several former HealthSouth employees have already pleaded
guilty to fraud and are expected to give evidence against Mr.
Scrushy.  "We will present evidence that Richard Scrushy knew about
the conspiracy, that he participated in the conspiracy and that he
profited," prosecutor Alice Martin told the court. Mr. Scrushy is the
first chief executive to be tried for breaching the Sarbanes Oxley
Act - a law introduced in the wake of the Enron and WorldCom frauds
which obliges corporate bosses to vouch for the accuracy of their
companies' results. Among the charges he faces are conspiracy to
commit fraud, filing false statements, and money laundering. After
federal agents raided HealthSouth's offices in March 2003, the
```

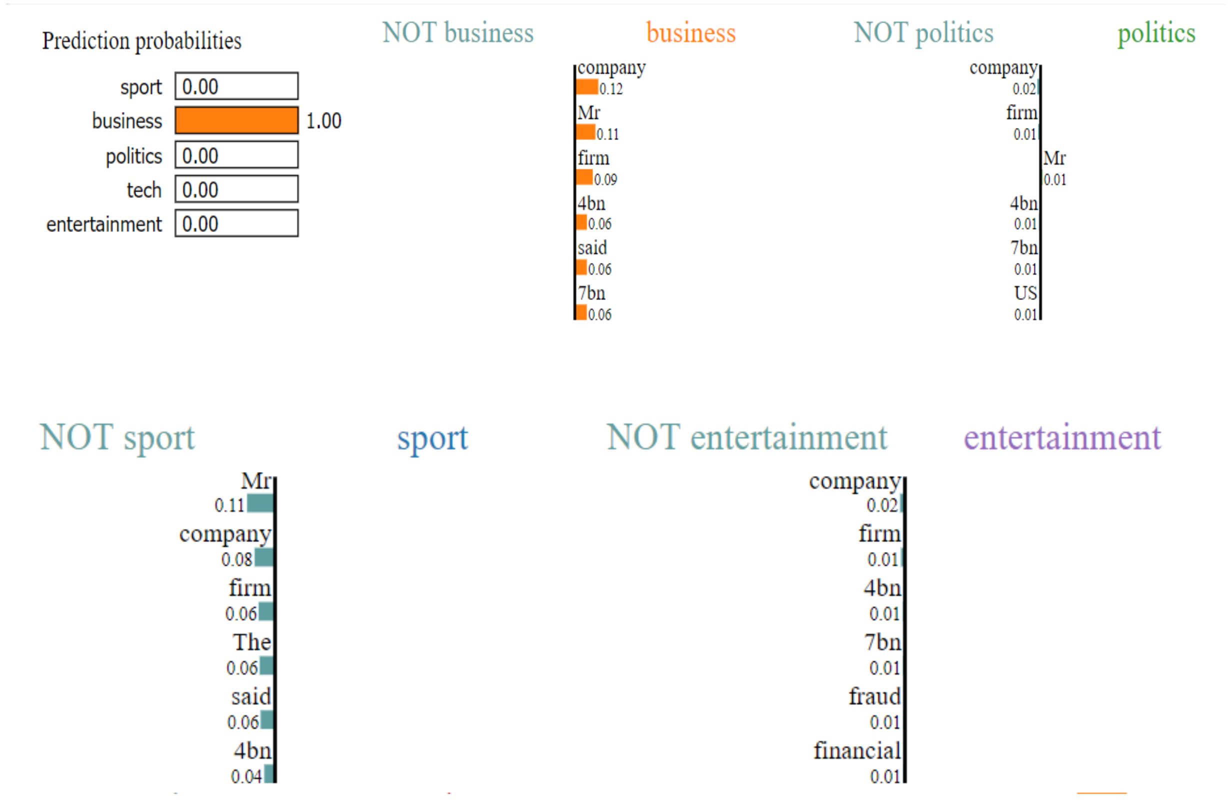
```
company said none of its past financial statements could be relied
on. The firm has since reorganized its board and management team and
currently operates about 1,400 health clinics."
```

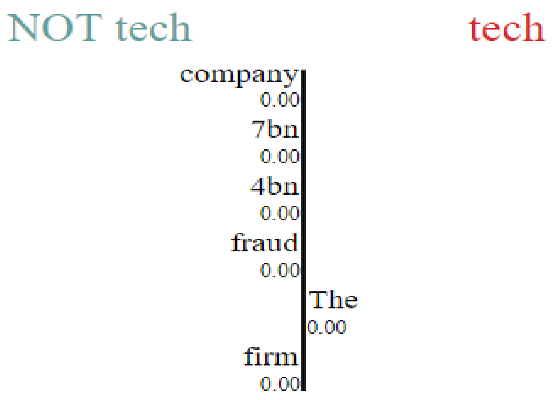
Our classifier's result was:

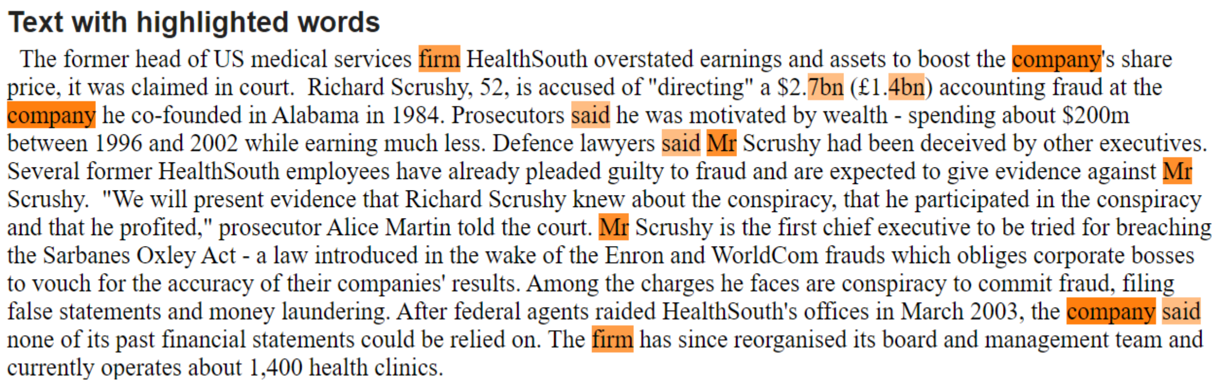
```
True class: business

Predicted class: business
```

Now, our question is why our model is predicting this news article to be in business class? Lime provides the answer that some particular feature i.e. words are carrying too many weights(positive/negative) than the others and that is why it is falling into business class and not in other class. Firm, company, Mr., said, etc. are the strong words which are dragging this news article to business class. If we remove these words this news article might fall into another class. Let's see the visualization provided by Lime







Text with highlighted words

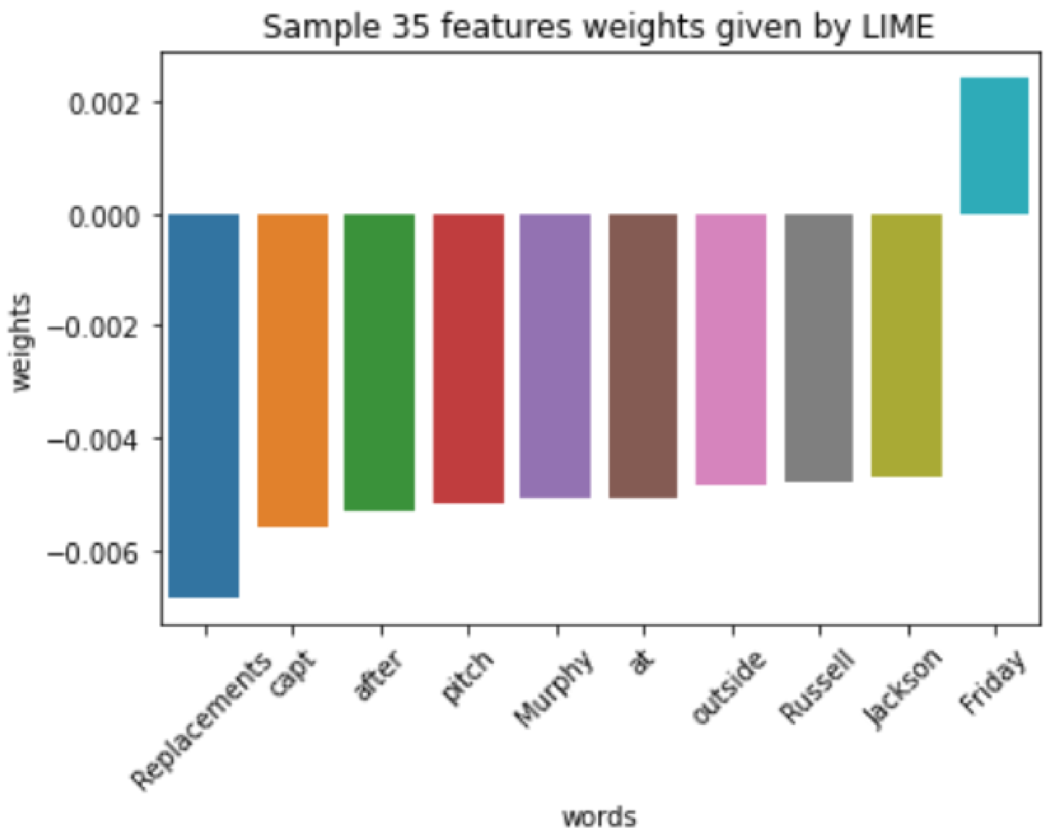
The former head of US medical services firm HealthSouth overstated earnings and assets to boost the company's share price, it was claimed in court. Richard Scrushy, 52, is accused of "directing" a $2.7bn (£1.4bn) accounting fraud at the company he co-founded in Alabama in 1984. Prosecutors said he was motivated by wealth - spending about $200m between 1996 and 2002 while earning much less. Defence lawyers said Mr Scrushy had been deceived by other executives. Several former HealthSouth employees have already pleaded guilty to fraud and are expected to give evidence against Mr Scrushy. "We will present evidence that Richard Scrushy knew about the conspiracy, that he participated in the conspiracy and that he profited," prosecutor Alice Martin told the court. Mr Scrushy is the first chief executive to be tried for breaching the Sarbanes Oxley Act - a law introduced in the wake of the Enron and WorldCom frauds which obliges corporate bosses to vouch for the accuracy of their companies' results. Among the charges he faces are conspiracy to commit fraud, filing false statements and money laundering. After federal agents raided HealthSouth's offices in March 2003, the company said none of its past financial statements could be relied on. The firm has since reorganised its board and management team and currently operates about 1,400 health clinics.

We again used Lime on a previously done RNN model. Our test news article is as follows

"Headingley Friday, 25 February 2000 GMT The Tykes have brought in Newcastle prop Ed Kalman and Tom McGee from the Borders on the loan while fly-half Craig McMullen has joined from Narbonne. Raphael Ibanez is named at hooker for Saracens in one of four changes. Simon Raiwalui and Ben Russell are also selected in the pack while Kevin Sorrell comes in at the outside center. - Friday's game at Headingley got the go-ahead on Friday after passing an early pitch inspection. Leeds: Balshaw; Rees, Christophers, Bell, Doherty; McMullen, Dickens; McGee, Rawlinson, Gerber; Murphy, Palmer (capt), Morgan, Parks, Popham. Replacements: Kalman, Regan, Hyde, Rigney, McMillan, Rock, Vickerman. Saracens: Bartholomeusz; Castaignede, Sorrell, Harris, Vaikona; Jackson, Bracken; Yates, Ibanez, Visagie; Raiwalui, Fullarton; Randell, Russell, Vyvyan (capt). Replacements: Cairns, Lloyd, Broster, Chesney, Johnston, Rauluni, Little."

Lime gives the following prediction

```
Probability(sport) = 0.9874618649482727
Probability(business) = 0.0006117829470895231
Probability(politics) = 0.001372863189317286
Probability(tech) = 0.007293272763490677
Probability(entertainment) = 0.0032602122519165277
True class: sport
157/157 [==============================] - 44s 283ms/step
```



6 Conclusion

For this news article classification task, we have used several approaches from a simple classification model to a complex deep learning model. Every single approach is doing a very fine job of classifying our text data.TF-IDF and BoW are not creating too much difference. Notably, simple baseline ML models like Naive Bayes classifier, Logistic Regression, and Support Vector Machine are beating complex deep learning models like

CNN and RNN in the context of accuracy measure or time. This is happening because our data is simple enough to extract the pattern by simple models and the amount of data is not too big. Deep learning models are data-hungry. Although the pre-trained models are bulky ,they are doing very well too.