Text Classification on BBC news data

Text Mining and Classification

Abstract

Use the provided data to perform text classification across 5 classes in the dataset, that are:-
[Sports, Business, Politics, Tech, Entertainment]
The classification should be done only using Deep learning.

Shivam Baslas
Shivam72baslas@gmail.com

INDEX

Chapter 1
Introduction

1.1. Problem Statement and Project Description

We have to perform text classification on the BBC text,The provided data includes articles and corresponding classes for each sample article.
We have to use a “Deep Learning” model to achieve classification on the following dataset.

Instructions:
1. Use the provided data to perform text classification across 5 classes in the dataset, that are:-[Sports, Business, Politics, Tech, Entertainment]
2. The classification should be done only using Deep learning.
3. Use a “70% Train and 30% Test” dataset split.
4. Your accuracy on the Test Set will determine your results.
5. Report the plots of loss and accuracy across epochs for both training and test data.

1.2 Data

We have 2 variables and 2225 observations in given data set. The variable text contain text string and the variable category having 5 categories that are:- (Sports, Business, Politics, Tech, Entertainment) it is also our target variable to develop the model top of it. As target variable is categorical it is a classification problem.

Table 1.1: Sample Data

	category	text
0	tech	tv future in the hands of viewers with home th...
1	business	worldcom boss left books alone former worldc...
2	sport	tigers wary of farrell gamble leicester say ...
3	sport	yeading face newcastle in fa cup premiership s...
4	entertainment	ocean s twelve raids box office ocean s twelve...

(2225, 2)

1.3 Expletory Data Analysis

Exploratory Data Analysis (EDA) is an approach to analysing data sets and summarize their main characteristics.

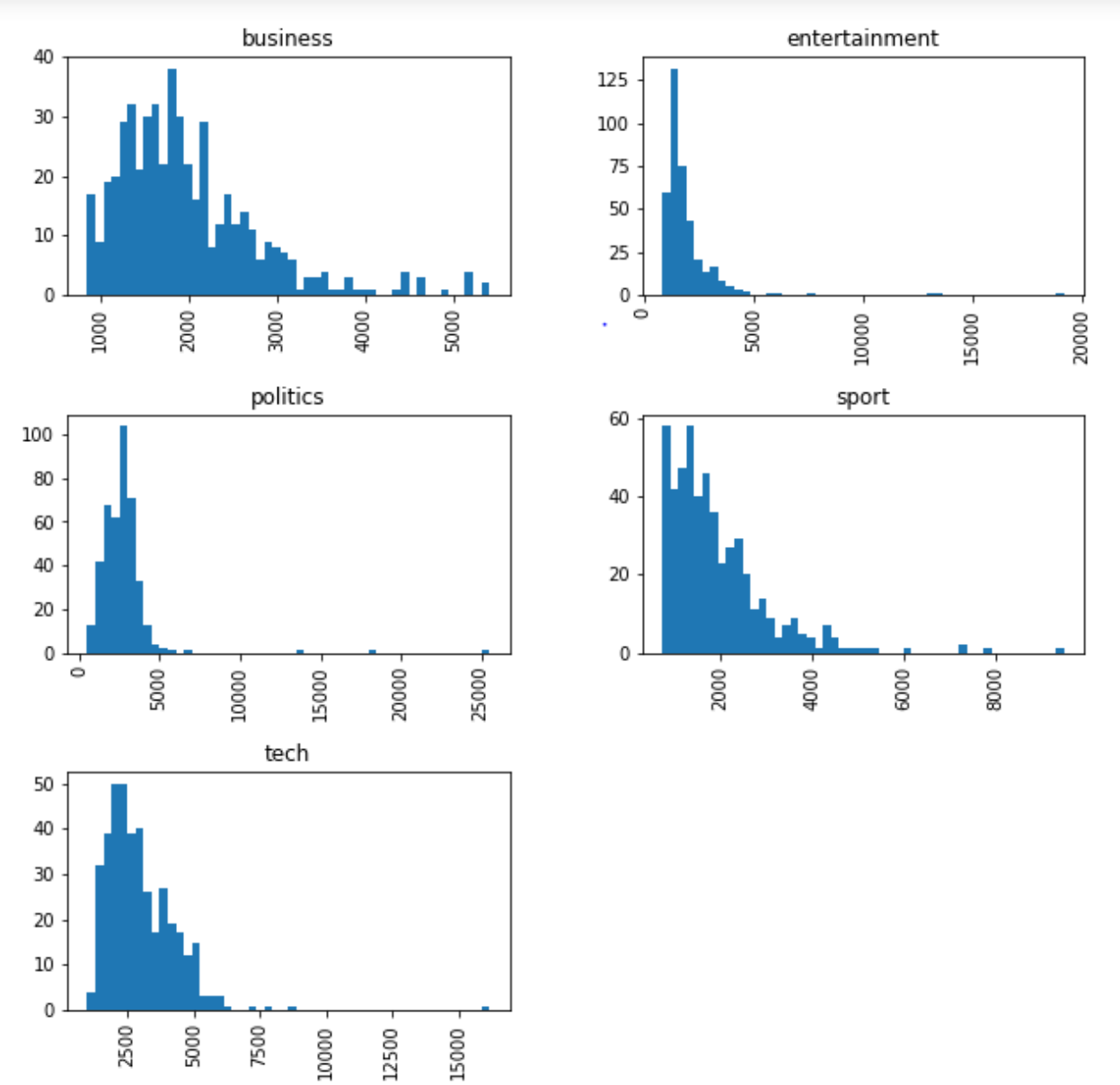
Table 1.2: Data summary group by category

category		text
business	count	510
	unique	503
	top	jobs growth still slow in the us the us create...
	freq	2
entertainment	count	386
	unique	369
	top	famed music director viotti dies conductor mar...
	freq	2
politics	count	417
	unique	403
	top	blair dismisses quit claim report tony blair h...
	freq	2
sport	count	511
	unique	504
	top	spain coach faces racism inquiry spain s footb...
	freq	2
tech	count	401
	unique	347
	top	millions to miss out on the net by 2025 40% o...
	freq	2

Attach tables describe the categories mentioned in the data set –

- Total count(count),
- Unique values (unique)
- Observation which having high occurrence (top)
- Occurrence frequency(freq).

Table 1.2: Histogram of the length of each document group by category



Maximum news text having the length of the strings b/w 1000 to 2500. The news on politics having the highest length of the news.

Chapter 2

Methodology

2.1. Text Preprocessing (Text Mining)

Text mining is the process of extracting interesting and non-trivial information and knowledge from unstructured text.

2.1.1. Pre-processing steps:

- **Punctuation Marks** – Remove punctuation marks
- **Numbers** – Remove numbers
- **Case folding**- Reduce all letters to lower case (exception: upper case in mid-sentence)
- **Stop words**- Remove stop words (e.g. function words: “a”, “the”, “in”, “to”; pronouns: “I”, “he”, “she”, “it”).
- **Stemming** - Reduce tokens to “root” form of words to recognize morphological variation.

2.1.2. Vectorization

Currently, we have the messages as lists of tokens and now we need to convert each of those messages into a vector the SciKit Learn's algorithm models can work with.

Now we'll convert each message, represented as a list of tokens (lemmas) above, into a vector that machine learning models can understand.

We'll do that in three steps using the bag-of-words model:

1. Count how many times does a word occur in each message (Known as term frequency)
2. Weigh the counts, so that frequent tokens get lower weight (inverse document frequency)
3. Normalize the vectors to unit length, to abstract from the original text length (L2 norm)

Let's begin the first step:
Each vector will have as many dimensions as there are unique words in the SMS corpus. We will first use SciKit Learn's CountVectorizer. This model will convert a collection of text documents to a matrix of token counts.

We can imagine this as a 2-Dimensional matrix. Where the 1-dimension is the entire vocabulary (1 row per word) and the other dimension are the actual documents, in this case a column per text message.

For example:

	Message 1	Message 2	...	Message N
Word 1 Count	0	1	...	0
Word 2 Count	0	0	...	0
...	1	2	...	0
Word N Count	0	1	...	1

Since there are so many messages, we can expect a lot of zero counts for the presence of that word in that document. Because of this, SciKit Learn will output a Sparse Matrix.

2.1.3. TF-IDF Metrics

TF-IDF stands for *term frequency-inverse document frequency*, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model.

Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

 # Note all the outputs are save in Jupyter Notebook file.

2.2 Model Development

With Text represented as vectors, we can finally train our text classifier. Now we can actually use almost any sort of classification algorithms. For a variety of reasons, the Naive Bayes classifier algorithm is a good choice.

2.2.1. Data split into train and test

As instructed to Use a “70% Train and 30% Test” dataset split.

```
Documents in training set are : 1557
Documents in test set are : 668
Total documents in courpus are : 2225
```

2.2.2. Creating a Data Pipeline

Now run our model again and then predict off the test set. We will **use SciKit Learn's pipeline** capabilities to store a pipeline of workflow. This will allow us to set up all the transformations that we will do to the data for future use.

Now we can directly pass message text data and the pipeline will do our pre-processing for us

```
Pipeline(memory=None,
         steps=[('bow',
                 CountVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>, encoding='utf-8',
                                 input='content', lowercase=True, max_df=1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 1), preprocessor=None,
                                 stop_words=None, strip_accents=None,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, vocabulary=None)),
                ('tfidf',
                 TfidfTransformer(norm='l2', smooth_idf=True,
                                  sublinear_tf=False, use_idf=True)),
                ('classifier',
                 MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True))],
         verbose=False)
```

2.2.3 Model Evaluation

Now we have a classification report for our model on a true testing set!!!!

Confusion metrics:

It is the Error matrix which describe the performance of classification Model. Row represents actual class/values, Column represent predicted class/values.

```
[[140   0   1   0   1]
 [  4 110   5   0   0]
 [  1   0 119   0   1]
 [  0   0   0 157   0]
 [  1   0   1   3 124]]
```

Classification report:

	precision	recall	f1-score	support
business	0.96	0.99	0.97	142
entertainment	1.00	0.92	0.96	119
politics	0.94	0.98	0.96	121
sport	0.98	1.00	0.99	157
tech	0.98	0.96	0.97	129
accuracy			0.97	668
macro avg	0.97	0.97	0.97	668
weighted avg	0.97	0.97	0.97	668

The terms in the classification reports are-

Accuracy:

It describe that how accurately model can able to classify

Accuracy = TP+TN/ Total observations

Recall:

It is the proportion of actual positive cases which are correctly identified.Also known as Sensitivity or True positive rate

Recall= TP/TP+FN