VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MACHINE LEARNING - CO3117

Assignment

TEXT CLASSIFICATION

Lecturer: - Nguyễn Đức Dũng
Student: - Đặng Trần Đạt 1852312

HO CHI MINH CITY, DECEMBER 2022

Contents

1 Introduction

With the development of web blogs and Social Networks, many organizations and News providers used to share their news headlines on various websites and web blogs. Nowadays, there are many newsgroups that share their news headlines as short messages in microblogging services such as Twitter. Once these short messages got processed, they may carry out a significant amount of information that will be relevant for many social research areas. Thus, the purpose of this research is to classify news short messages into different groups so that the user could identify their interesting newsgroup. This may be useful to get brief information about the current state of the country, by providing necessary information about development, war, education, etc.

There are several news portals currently available to retrieve short messages due to several reasons as follows:

- Microblogging platforms are used by different people to express their opinion about different topics, thus it is a valuable source of people's opinions.
- News portals contain an enormous number of text posts and it grows every day. The collected corpus can be arbitrarily large.
- The audience varies from regular users to celebrities, company representatives, politicians, and even country presidents. Therefore, it is possible to collect text posts of users from different social and interest groups.
- The audience is represented by users from many countries.

In this project, the short messages will be classified by the system into 5 groups: sport, business, politics, tech and entertainment. These groups were chosen in order to cover the main areas of a general news provider.

With the development of machine learning techniques, nowadays, many researchers tend to use machine learning techniques in text classification [1, 2]. There are 2 types of machine learning techniques including supervised learning [3] (the learning data will be provided by the developer) and unsupervised learning [3] (the method will learn a clustering procedure by observing the distance among data). For the present project, supervised learning techniques will be used as the 5 groups do not change regularly.

In order to classify short messages using machine learning techniques, a proper set of features is required to extract from the short messages. Once created the dataset, it is important to find a suitable classification method in order to classify the short messages. Regarding the classical method, **Support Vector Machine** [4] was used to classify the data as it is capable of dealing with high dimensional datasets. In the aspect of the modern method, we would like to perform a deep learning network technique which is **Long-Short Term Memory** (LSTM) [5] to use the power of the Artificial Neural Network in the text classification task. Based on the result of these methods, we show the comparison between them to get an overview of different algorithms to use for text classification.

2 Dataset

In this project, I use the news article dataset, originating from BBC News [6], provided for use as benchmarks for machine learning research. This dataset consists of 2225 documents from the BBC news website corresponding to stories in five topical areas from 2004-2005. These documents are classified into 5 different categories:

- Business.
- Entertainment.
- Politics.
- Sport.
- Tech.

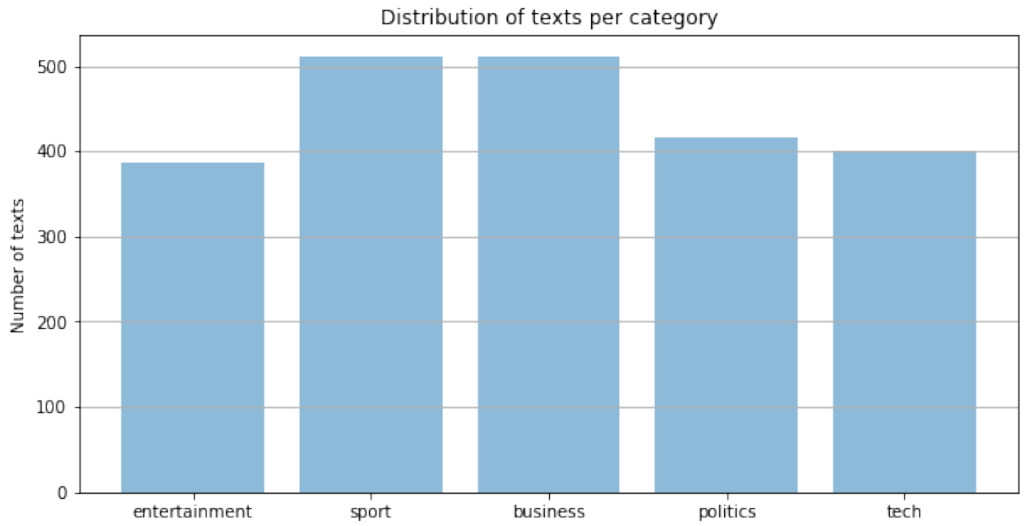

Figure 1: The distribution of texts per category.

As you can see in Figure 1, the samples **are not balanced**. This could cause problems during the training but, since they are not highly unbalanced, I have left it as it is.

In other cases, such as fraud detection where the positive classes are very few compared to the negative ones, we must apply techniques to balance it. For example, we could undersample the biggest category.

3 Data Preprocessing

Before starting to clean the data, I calculate the statistics of the dataset based on the length of each document of categories.

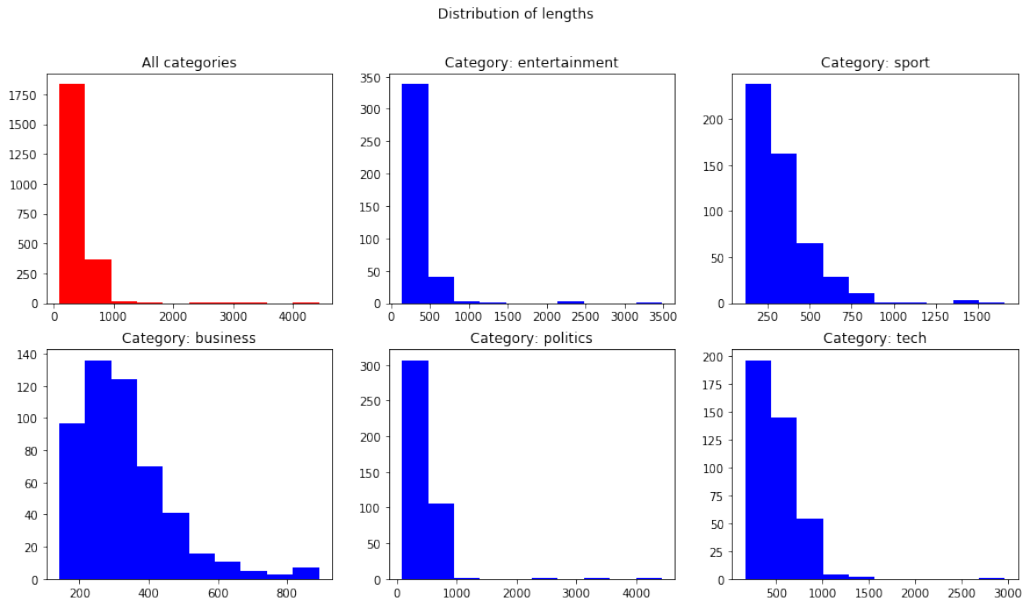

Figure 2: The distribution of document length.

Category	Mean of length	St.dev
entertainment	331.621762	261.795254
sport	330.262231	187.963267
business	329.880392	135.919077
politics	454.973621	300.120275
tech	503.695761	239.849176

Table 1: Statistic about document length of categories.

In these Histograms 2 and Stats 4, we can see that almost all texts contain 1000 or fewer words. Also, we can see that the average length is very different depending on the category.

3.1 Clean characters

In the first step of preprocessing, I would like to remove every character which cannot be used to write words in English (this set of characters is different in other languages). I not only remove all punctuation symbols from all texts, but I also normalize the white spaces.

3.2 Lemmatization

Then, I do the lemmatization over all the texts. To do so, I use the library **Spacy** since it already has detailed dictionaries which the algorithm can look through to link the form back to its lemma.

We can illustrate the method with an example:

Word	Lemma
studying	study
studies	study

This process may look similar to *stemming* (which we are not going to use), but it leads to a different result.

3.3 Stop words

Stop words are a set of commonly used words in any language. For example, in English, "the", "is" and "or", would easily qualify as stop words. These words are not semantically useful in the sentences, thus it is a good idea to remove them and focus on important words instead.

3.4 Term Frequency(TF) — Inverse Dense Frequency(IDF)

TF-IDF is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.[1] It is often used as a weighting factor in searches for information retrieval, text mining, and user modeling. The TF–IDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. It is one of the most popular term-weighting schemes today. TF-IDF can be successfully used for stop-words filtering in various subject fields, including text summarization and classification. In this step, I remove from the dataset those words whose TF-IDF value is less than a certain threshold.

To find the threshold value for each document d, first I did some computations. I have normalized the TF-IDF values as a distribution function. This means that the sum of all TF-IDF values is 1:

$$\sum_{w \in d} \text{tf-idf}(w, d) = 1$$

Then, after I sorted the TF-IDF values in descendant order, I have calculated the value k such as the sum is less than a certain p-value (in this project, I used 0.975).

$$\sum_{i=1}^{k} \text{tf-idf}(w_i, d)$$

Finally, the value $\text{tf-idf}(w_k, d)$ is the threshold that I have used in the document d.

After the preprocessing step, we summerize the dataset again and receive the following result:

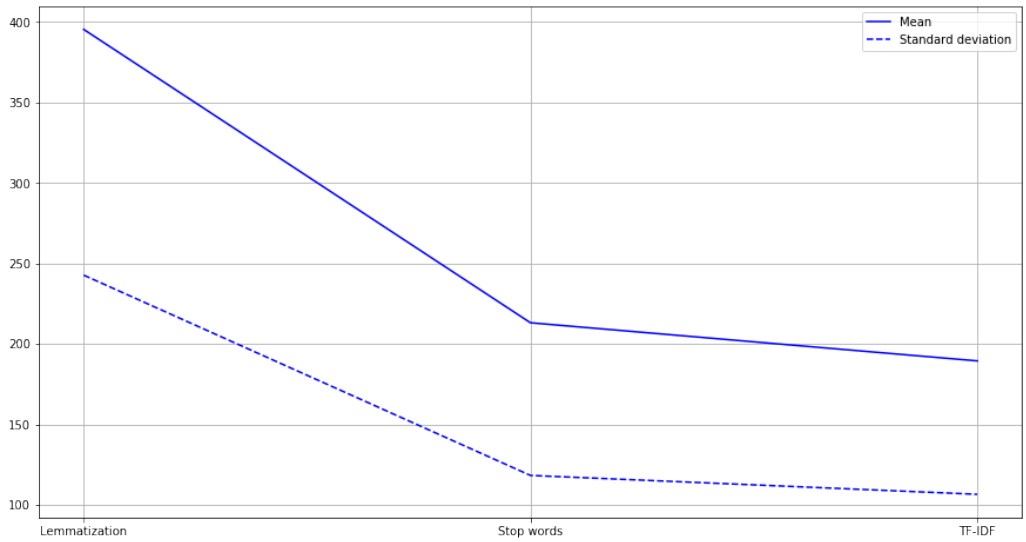

Figure 3: The distribution of document length after preprocessing.

Category	Mean of length	St.dev
entertainment	168.585492	118.648402
sport	159.681018	92.709783
business	172.876471	65.119283
politics	215.347722	128.812873
tech	241.835411	103.570666

Table 2: Statistic about document length of categories after preprocessing.

4 Support Vector Machine method

4.1 Introduction

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

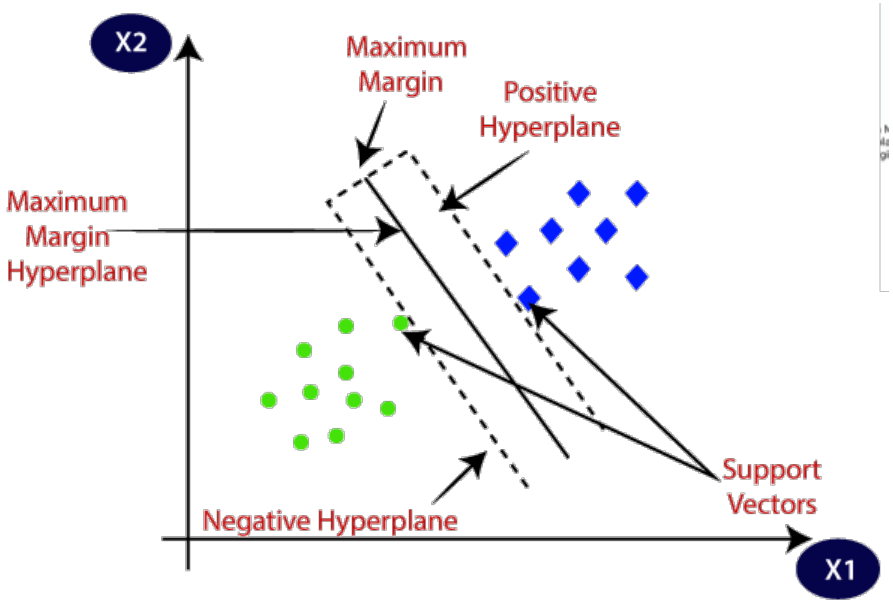

Figure 4: Support Vector Machine example.

4.2 Data preparation

In this method, I just need to load the prepared data and the remaining steps such as vectorizing the data are done by a tool called **CountVectorizer**. It is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors (for use in further text analysis). Let me consider a few sample texts from a document (each as a list element):

document = ["One Geek helps Two Geeks", "Two Geeks help Four Geeks", "Each Geek helps many other Geeks at GeeksforGeeks."]

CountVectorizer creates a matrix in which each unique word is represented by a column of the matrix, and each text sample from the document is a row in the matrix. The value of each cell is nothing but the count of the word in that particular text sample. This can be visualized as follows:

	at	each	four	geek	...	other	two
document[0]	0	0	0	1	...	0	1
document[1]	0	0	1	0	...	0	1
document[2]	1	1	0	1	...	1	0

Key Observations:

1. There are 12 unique words in the document, represented as columns of the table.

2. There are 3 text samples in the document, each represented as rows of the table.
3. Every cell contains a number, that represents the count of the word in that particular text.
4. All words have been converted to lowercase.
5. The words in columns have been arranged alphabetically.

Inside CountVectorizer, these words are not stored as strings. Rather, they are given a particular index value. In this case, 'at' would have index 0, 'each' would have index 1, 'four' would have index 2 and so on. The actual representation has been shown in the table 3 below:

0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	1	1	0	0	1	0	1	0	1
0	0	1	0	2	0	1	0	0	0	0	1
1	1	0	1	1	1	0	1	1	0	1	0

Table 3: A Sparse Matrix

4.3 Model generation

First of all, I create a pipeline of transforms with a final estimator including **CountVertorizer** and **SGDClasiffier**. This is a linear classifier (SVM, logistic regression, a.o.) optimized by the SGD. These are two different concepts. While SGD is an optimization method, Logistic Regression or Support Vector Machine is a machine learning algorithm/model.

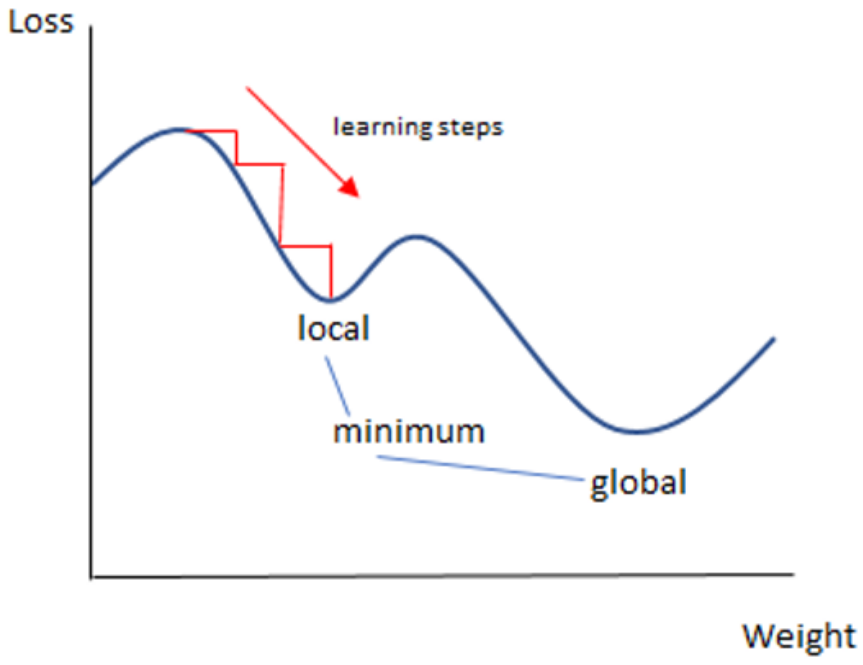

Figure 5: Gradient descent in general.

In a nutshell, gradient descent is used to minimize a cost function. Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. But we can also use these kinds of algorithms to optimize our linear classifiers such as Support Vector Machines.

SGD allows minibatch (online/out-of-core) learning. Therefore, it makes sense to use SGD for large-scale problems where it's very efficient. In addition, SVM does not work if we cannot keep the record in RAM, while SGD Classifier continues to work.

In this classifier, there are many hyperparameters that need to be tuned for better results including loss function, penalty, a constant that multiplies the regularization term, stopping criterion and random_state used for shuffling the data. These hyperparameters are one of the important factors in the performance of the model, once we set appropriate values for these hyperparameters, the performance of a model can improve significantly. In this project, we would like to optimize values for the hyperparameters of a model by using **GridSearchCV**. Moreover, GridSearchCV tries all the combinations of the values passed in the dictionary and evaluates the model for each combination using the **Cross-Validation** (I define 10-fold in this project) method. Hence after using this function we get accuracy/loss for every combination of hyperparameters and we can choose the one with the best performance.

4.4 Model evaluation

It took very little time to train the model and the results are pretty good. This approach got a global accuracy of 96.86%, which is slightly more than the LSTM model and, in comparison to that model, SVM is much faster to train.

I evaluate this model based on three metrics as shown in the table below:

	precision	**recall**	**f1-score**	**support**
business	0.93	0.97	0.95	141
entertainment	0.99	0.97	0.98	108
politics	0.99	0.94	0.96	149
sport	0.99	0.98	0.99	154
tech	0.94	0.98	0.96	116
accuracy			0.97	668
macro avg	0.97	0.97	0.97	668
weighted avg	0.97	0.97	0.97	668

Table 4: Statistic about document length of categories.

For more detail, I also create a confusion matrix as follow:

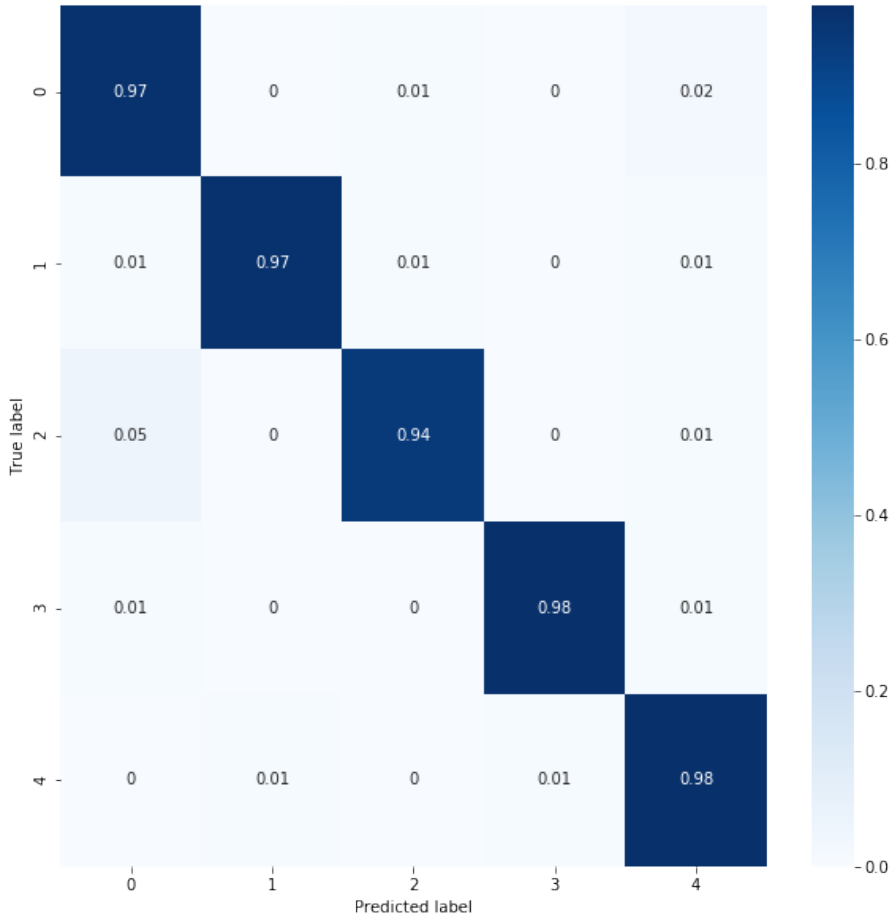

Figure 6: The confusion matrix of SVM method.

[]

5 Long-Short Term Memory method

5.1 Introduction

Recurrent Neural Network (RNN), a branch of Artificial Intelligence, was developed in the 1986 by Rumelhart [7] and improved by Werbos (1988) [8] and Elman (1990) [9] for enhancing the learning performance on sequence data like texts, documents, etc. In general, the construction of an RNN is similar to an FNN, with the distinction that a presence of connections between hidden layers, spanning through adjacent time steps. Figure 7 presents the basic architecture of an RNN, which has physically one layer. At the time t, this network will produce output $y^{(t)}$ from the input $x^{(t)}$. However, the output of this network at the previous iteration will also be used as part of the input of the next step, or *recurrent input*, together with new actual input. Similar to a typical *Multi-Layer Perceptron* (MLP) [10], RNN uses some layers of the perceptron to learn suitable weights in the backpropagation manner when processing input, output and recurrent input, denoted as $\mathbf{W}_{ax}$, $\mathbf{W}_{ya}$ and $\mathbf{W}_{aa}$, respectively [7]. Thus, when handling a

sequence of data $x^{(t)}$, an RNN can be logically unfolded as a recurrent multilayer network, as depicted in Figure 8.

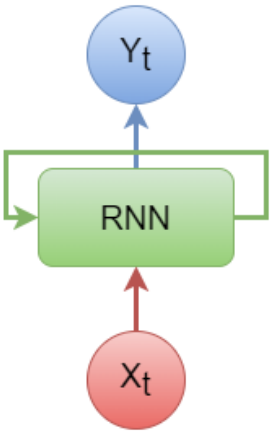

Figure 7: The physical architecture of an RNN

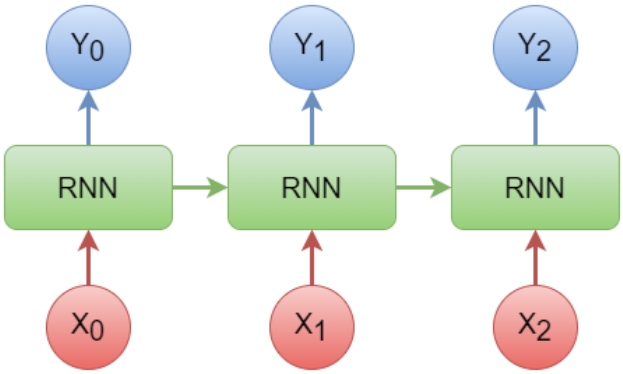

Figure 8: An unfolded RNN

Although the RNN is theoretically a powerful model, it has problems in learning long-term dependencies, caused by two well-known issues which are *vanishing* and *exploding gradient* [11]. The vanishing gradient will become worse when a *sigmoid* [7] activation function is used, whereas a *Rectified Linear Unit* (ReLU) can easily lead to an exploding gradient. Fortunately, a formal thorough mathematical explanation of the vanishing and exploding gradient problems was represented by Bengio [12], analyzing conditions under which these problems may appear.

To overcome the limitations of RNN, a successor of RNN was introduced by Hochreiter and Schmidhuber in 1997, called *Long Short Term Memory* (LSTM) [13]. LSTM can deliver higher performance by using a hidden layer as a memory cell instead of a recurrent cell (see Figure 9). This LSTM model contains computational blocks which are repeated over many timesteps with the goal of determining what information will be added or removed. This process is controlled by three gates named *input gate*, *output gate*, and *forget gate*. Controlling the flow of information inside an LSTM model is calculated using equation (1a)-(1e).

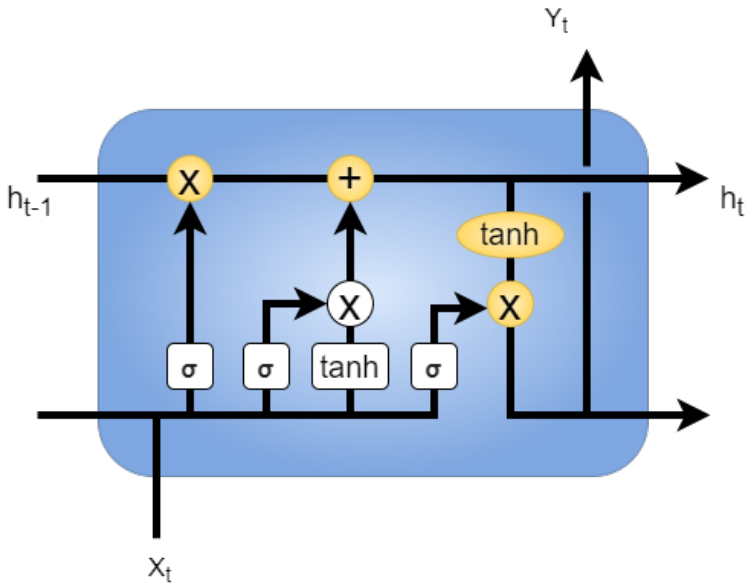

Figure 9: The structure of an LSTM cell

$$
\begin{cases}
i_t = \sigma(\mathbf{W}_i x_t + \mathbf{W}_{hi} h_{t-1} + \mathbf{b}_i) & \text{(1a)} \\
f_t = \sigma(\mathbf{W}_f x_t + \mathbf{W}_{hf} h_{t-1} + \mathbf{b}_f) & \text{(1b)} \\
o_t = \sigma(\mathbf{W}_o x_t + \mathbf{W}_{ho} h_{t-1} + \mathbf{b}_o) & \text{(1c)} \\
(C)_t = f_t \otimes (C)_{t-1} + i_t \otimes tanh(\mathbf{W}_C x_t + \mathbf{W}_{hC} h_{t-1} + \mathbf{b}_C) & \text{(1d)} \\
h_t = o_t \otimes tanh((C)_{t-1}) & \text{(1e)}
\end{cases}
$$

In equation (1a)-(1e), i, f, o, (C), h denote input gate, forget gate, output gate, internal state, and hidden layer, respectively. Here, $\mathbf{W}_i$, $\mathbf{W}_f$, $\mathbf{W}_o$, $\mathbf{W}_C$, and $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o$, and $\mathbf{b}_C$ represent the weights and bias of three gates and a memory cell, in the order given. The main success of LSTM is storing information of a sequence in a memory cell and updating it with learned two input and forget gate. Moreover, the next hidden state is controlled using the output gate, which makes more useful output. Because of above techniques, LSTM has overcome limitations of RNN, enhancing the ability to remember values over an arbitrary time interval by regulating the flow of information inside the memory cell.

5.2 Data preparation

Since the LSTM layer processes sequences of numbers, I needed to transform the texts into numbers. I used Keras Tokenizer to transform them. The Tokenizer contains a dictionary word number, so every text we tokenize will be the same. In addition, the prediction of the network is a numeric format, so I also transform each category into a number as follow:

- 0: entertainment
- 1: sport

- 2: business
- 3: politics
- 4: tech

On the other hand, the length is different among documents, so I fixed the *maximum length* of them as 1000 elements. I have **padded the short texts** with zeros and, in case there are texts longer than a maximum length, I cut them. So all texts are the same length. After that, I split the dataset into a ratio of 7:3 for training and testing sets. An example of a transform step is shown as follows:

['musicians tackle red tape musicians.....']
=> [0,...., 0, 3970, 19458, ..., 19459, 384, 2334]

5.3 Model generation

I have used the loss function *SparseCategoricalCrossentropy* since I have a multi-class problem. This model returns a vector of probabilities where each value is the probability for each category (hence I must pick the highest one).

In the first layer of the model, I used a pre-trained set of embedding vectors with a dimension of 100 to get more accurate results. These embeddings have been trained over Wikipedia 2014 database using the top $400K$ words. However, some words might not be present in that embedding set, so I have initialized the vectors (of those "missing words") with zeros.

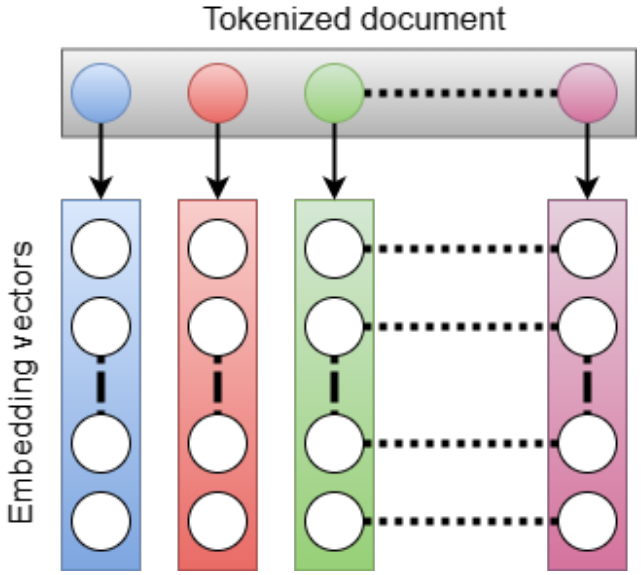

Figure 10: Word embedding step.

Then I drop out some elements of the input sequence before passing the LSTM network with the output of 64 units. Finally, it passes the feed-forward layer with the output of 5 (number of categories) and is activated by the softmax function for the probability of each class.

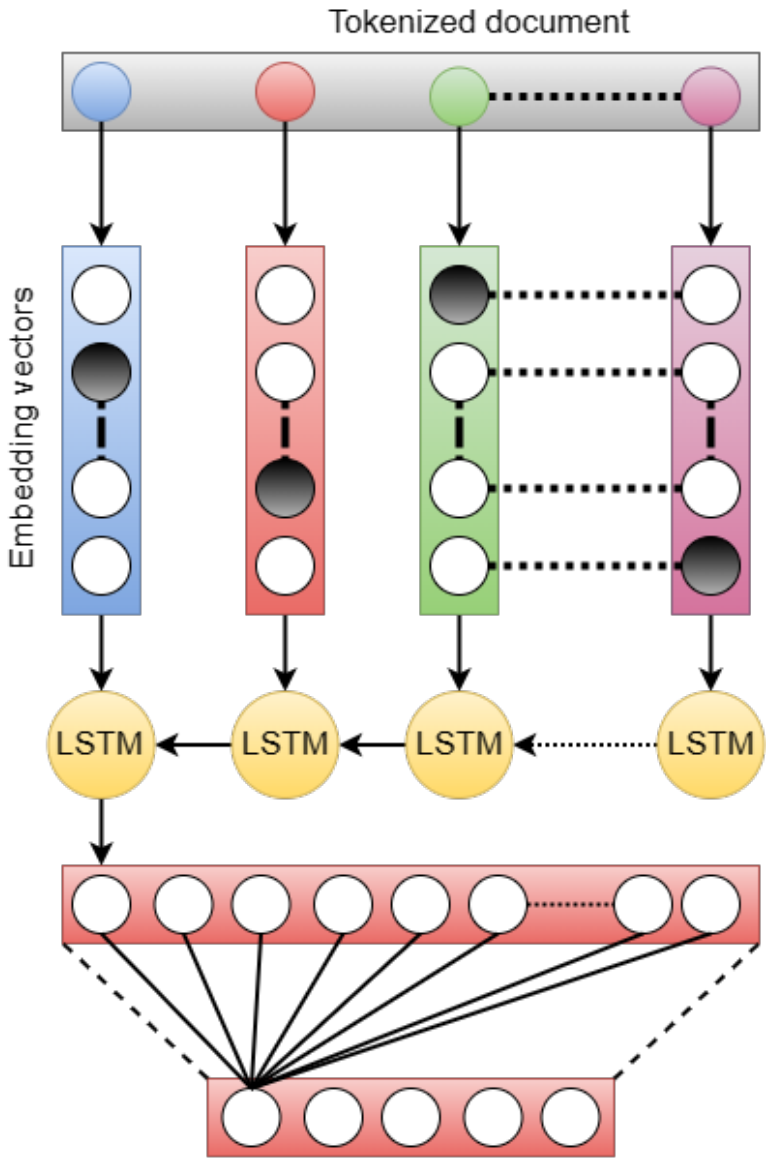

Figure 11: Model architecture

Since the classes are not highly unbalanced, I have used an accuracy metric to measure each model and find the best one. In other cases, a different metric such as F1-score should be considered.

The model is trained with 10 epochs and a batch size of 32 with the tunning parameters as follows:

- lstm_activation: ["relu", "tanh"]
- spartial_dropout: [0.2, 0.3, 0.5, 0.7]
- lstm_dropout: [0.2, 0.3, 0.5, 0.7]
- recurren_dropout: [0.2, 0.3, 0.5, 0.7]

5.4 Model evaluation

After training, I got the best parameters:

- lstm_activation: "tanh"
- spartial_dropout: 0.3
- lstm_dropout: 0.2
- recurrent_dropout: 0.5

The accuracy and loss of the training and validation set are shown in Figure 12.

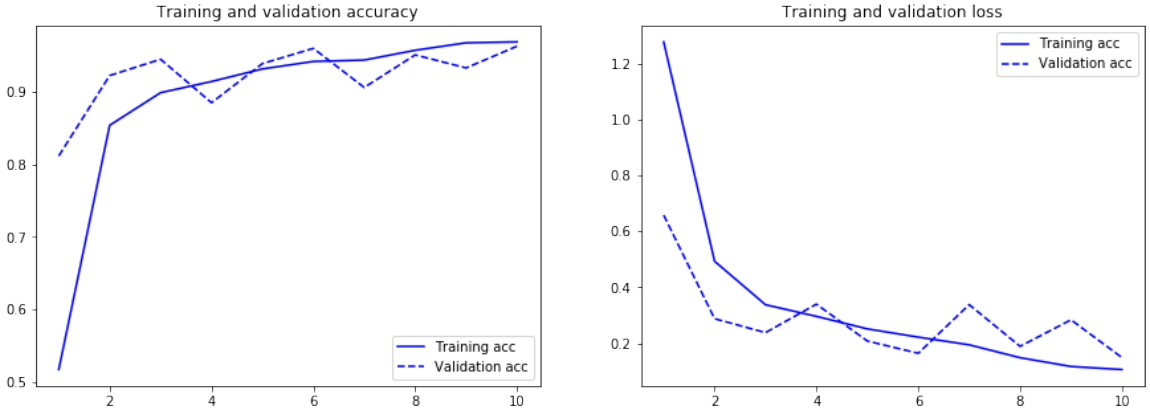

Figure 12: The accuracy and loss of the training and validation set

The global accuracy (i.e. taking into account all categories) looks pretty good for this selection of model and parameters: 96.26%. I also give a confusion matrix for each class in Figure 13

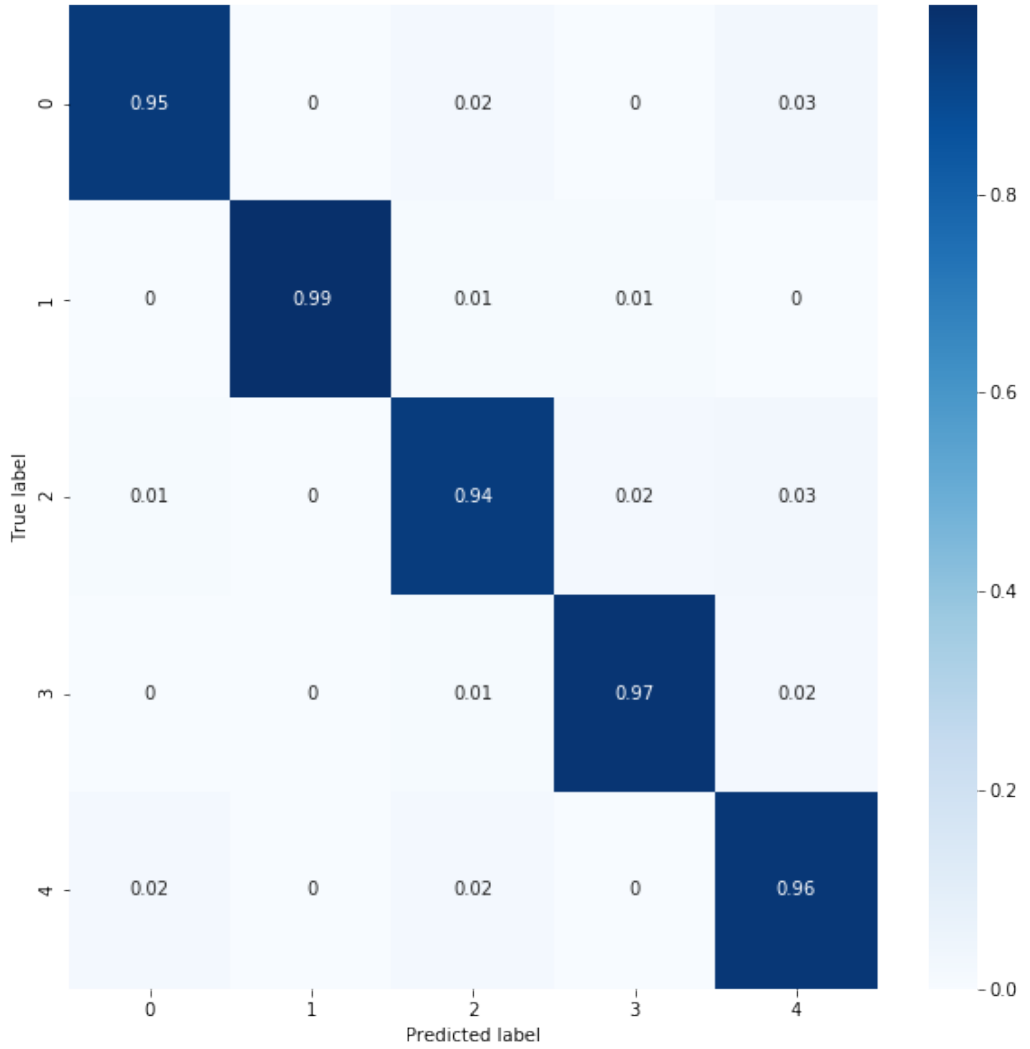

Figure 13: Confusion matrix of LSTM method.

If you take a closer look into the confusion matrix, you can see that most of the mismatches are in the categories 0 and 2 which, according to the dictionary of category-ID, are *entertainment* and *business*.

6 Conclusion

In general, we can see that both SVM and LSTM models give good performances for the task of text classification on a short text. Though it is a little bit imbalanced in five types of text, these methods also give good results in each category.

	SVM	LSTM
Accuracy	96.86%	96.26%

Table 5: The accuracy comparison between two methods.

References

[1] A. Pak and P. Paroubek, "Twitter as a corpus for sentiment analysis and opinion mining," in *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, 2010.

[2] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? sentiment classification using machine learning techniques," *arXiv preprint cs/0205070*, 2002.

[3] J. Han and M. Kamber, "Data mining: concepts and techniques, 2nd," *University of Illinois at Urbana Champaign: Morgan Kaufmann*, 2006.

[4] V. Jakkula, "Tutorial on support vector machine (svm)," *School of EECS, Washington State University*, vol. 37, no. 2.5, p. 3, 2006.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] D. Greene and P. Cunningham, "Practical solutions to the problem of diagonal dominance in kernel document clustering," in *Proc. 23rd International Conference on Machine learning (ICML'06)*, pp. 377–384, ACM Press, 2006.

[7] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct 1986.

[8] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1, no. 4, pp. 339–356, 1988.

[9] P. J. Elman, "Finding Structure in Time," *Cognitive Science*, vol. 14, pp. 179–211, Mar 1990.

[10] M. S. Akhtar, A. Kumar, D. Ghosal, A. Ekbal, and P. Bhattacharyya, "A Multilayer Perceptron based Ensemble Technique for Fine-grained Financial Sentiment Analysis," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 540–546, 2017.

[11] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, pp. 157–166, Mar 1994.

[12] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8624–8628, IEEE, 2013.

[13] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, Nov 1997.