Fake News Identification through Shallow and Deep Learning Techniques

Level 3 Big Data Summative Assignment

Z0954757

Abstract

Efficient and accurate tools to discern fake from genuine news have become sought-after as mis-informative articles continue to spread though social media. This report outlines approaches to build classifiers through both shallow and deep learning approaches. A shallow learning TF-IDF based Multinomial Naïve Bayes classifier optimised through a grid search achieved an accuracy of 0.9100. Implementing an LSTM based deep learning classifier with self-defined word-embeddings yielded a classifier achieving an accuracy of 0.8279. Even the most rudimentary shallow learning approaches outperformed the LSTM and were around 5 times more time-efficient in model fitting stages, making them a more suitable choice for application to the task.

1 Introduction

Rising to prominence during the 2016 US Presidential Election as social media sites were plagued by false and inflammatory articles, the emergence and influence of 'fake news' has drawn significant attention over the past two years. The number of fake articles being produced makes the problem insurmountable for human editors, leading those who wish to halt the spread of fake news to turn to computational methods.

This report outlines the efficacy of both shallow and deep learning techniques to identify fake news articles. Firstly, the report evaluates several shallow learning approaches by implementing a Multinomial Naïve Bayes model, using a variety of Natural Language Processing methods to extract numerical features from a dataset of 6336 labelled articles, and assessing the efficacy of each approach.

Secondly, the report explores the use of a deep learning approach. A long short-term memory model (LSTM) is implemented as a classifier, using a custom word-embedding process to feed the model. It also investigates the usage of a pre-trained word2vec embedding.

Preface

The computational analysis for this project was limited by availability of powerful hardware. All computations were performed on an Intel 7200U processor with 8Gb of RAM no GPU acceleration available. As such, it does not cover part D.II of the assignment as progress through part D.I was slow and required optimisation to obtain a workable pipeline.

2 Shallow Learning Approaches

Methodology

This report makes use of a dataset of 6336 news articles labelled as either fake or genuine news. A number of pre-processing techniques were explored to build and optimise a Multinomial Naïve Bayes classifier. In all approaches, a 'Bag of Words' model was implemented.

First approach - Term Frequency
The first approach implemented a classifier based on term frequency (TF), a simple measure of word occurrence in a document. To provide a baseline measure, the dataset underwent only tokenization.

Scikit's 'CountVectorizer' class was used to tokenize and vectorize the dataset, converting the corpus into a matrix of token counts to be used for TF-based analysis [1]. The Multinomial Naïve Bayes algorithm was also provided by Scikit [2].

Using these tools, a classifier was built based upon the term frequency of unigrams in the non-pre-processed dataset.

In all initial investigations, the fake news dataset was divided into training and test sets in a 60:40 ratio and no limit was placed on the maximum number of features to include in the classifier.

Second approach - Adding a Pre-Processor
In theory, the first approach is not ideal as the classifier will encode similar words such as 'church' and 'churches' as different vectors, losing the relationship between the two. To prevent the loss of this information, a second approach implemented a lemmatization tool to reduce similar tokens to a unified form. In the given example, both tokens would be reduced to 'church'. In addition to lemmatization, a list of stop words was used to remove common words that add little meaning to a document.

The Natural Language Toolkit (NLTK) offers tools for tokenizing and lemmatizing words in a corpus [3] [4]. NLTK also offers pre-prepared stop-word lists for implementation in a text classification pipeline. Both tools were used to pre-process the data-set before building a second TF based classifier.

Third approach - TF-IDF
Term frequency-inverse document frequency, or TF-IDF, takes the ideas of term frequency a step further. IDF is a measure of how rare a token is in the corpus and is calculated as

$$idf = \log \frac{N}{\mathrm{df}_t},$$

where N is the number of documents in a collection, and df_t is the document frequency of the term t [5].

TF-IDF is the scalar product of the TF vector and the IDF vector for a specific token [5]. As such TF-IDF gives a measure of a word’s significance to the corpus by scoring rare words that have high TF values higher than common words with comparable values.

Scikit’s ‘TfidfVectorizer’ class offers an analogous tool to the ‘CountVectorizer’ tool used in the first two approaches. Using this vectorization tool, a third classifier was built. Before vectorization, the corpus was lemmatized and stopped using the same tools as in the second approach.

Fourth Approach - N-Grams

The first three approaches all shared the characteristic that they considered each word in a document in isolation as a unigram. N-grams encode tokens as groups of neighbouring words, which may help to retain more context. As a basic example if we took the bigram “not good” and split it into the constituent unigrams “not” and “good” a unigram classifier could extract a positive sentiment. However, a bigram classifier may be able to extract that the original sentiment was negative.

In this fourth approach, the document was considered as a collection of bigrams using NLTK’s lemmatizer and English stop list and Scikit’s CountVectorizer. The n-gram format was varied to attempt to identify an optimal configuration.

Results

Initial Investigation

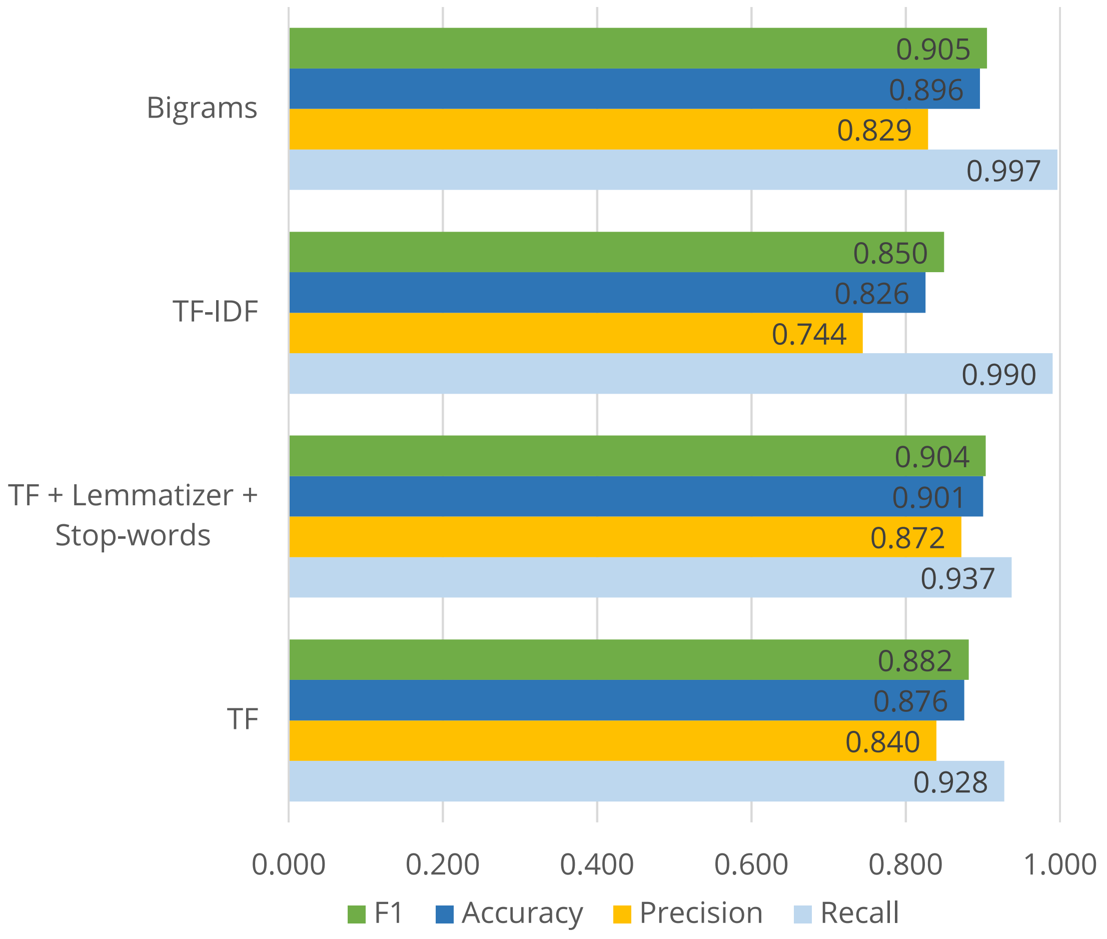

Figure 1 - Classifier metrics from the initial investigations in shallow learning approaches to fake-news classification.

The results from the classifiers produced by each approach are shown in Figure 1. Using a TF approach with the pre-processor gave the highest accuracy, 0.901. However, the bigram approach gave the highest F1 measure, 0.905.

The TF-IDF classifier performed relatively poorly with an F1 measure of 0.850. It received the lowest scores in all metrics but recall. This was unexpected as in theory TF-IDF should offer improvements over a classifier built on TF alone. Its high recall value may indicate that the conditions it was tested under resulted in overfitting.

Further Investigation

Given the unexpectedly poor performance of the TF-IDF based classifier, an additional investigation into its optimisation was carried out using a grid search method. The fraction of samples used to train the classifier was varied to investigate the effect on the effectiveness of the classifier of the ratio between the number of training and test samples. The maximum number of features included in the classifier was also varied to investigate its effect on the effectiveness of the classifier. The results of this experiment are shown in Figure 2.

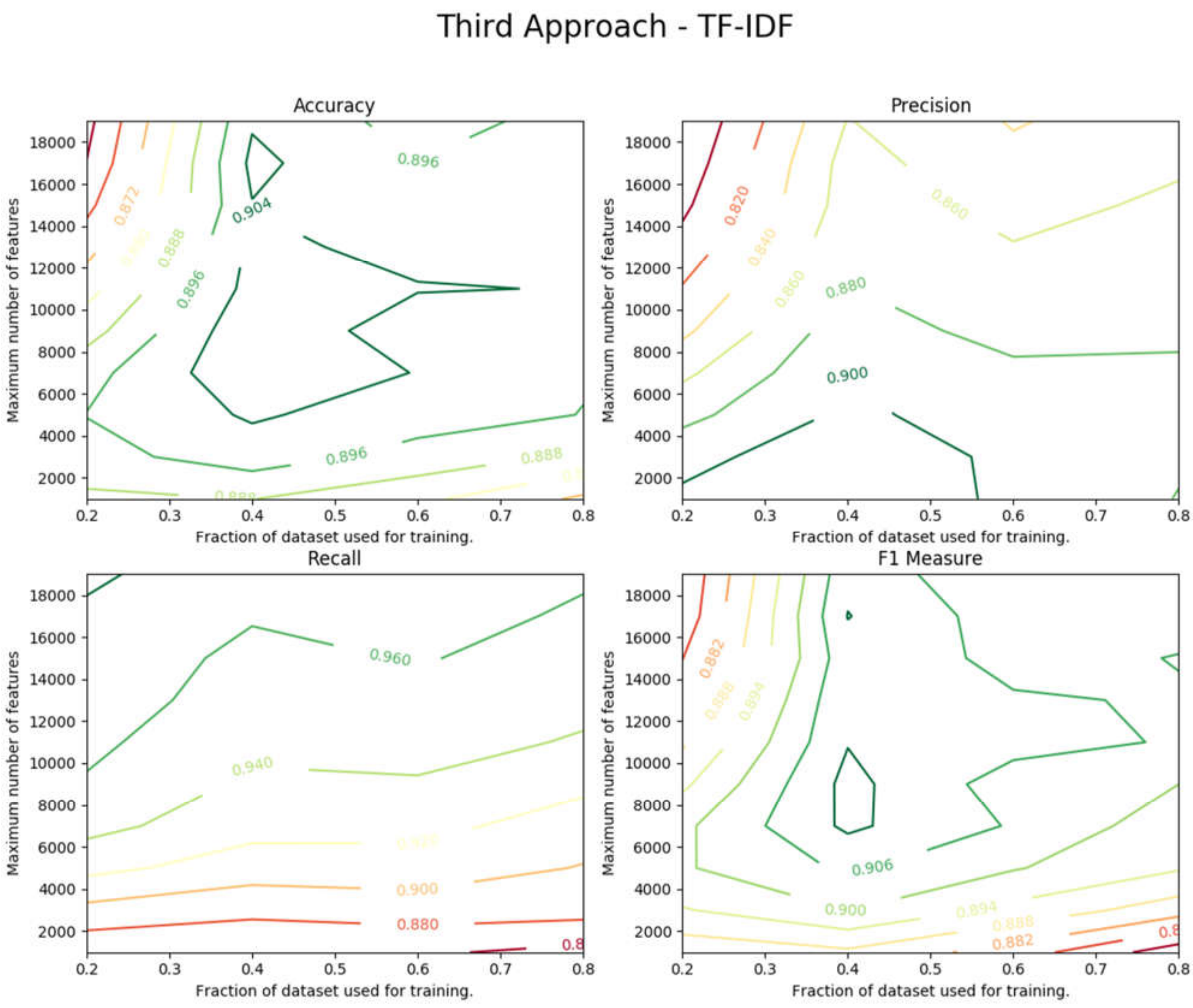

Figure 2 - Variation of classifier metrics for the TF-IDF classifier, produced using a grid search method.

Considering the variation of the accuracy and F1 measure, the optimum conditions for the TF-IDF classifier were found whilst using a training set of 40% and setting a maximum number of features to around 8000. Values in this region also corresponded with the greatest values for accuracy. Using these optimum conditions, the TF-IDF classifier was able to reach the values shown in Table 1.

Table 1 - Classifier metrics for the TF-IDF classifier with 40% training set and a maximum of 9000 features.

Accuracy	Precision	Recall	F1 Measure
0.910024	0.892118	0.936401	0.913724

With bigrams initially offering the best F1 measure, the optimisation grid-search was extended to cover a combination of n-grams up to tri-grams. The results of the tests are shown in Figure 3. Using a combination of unigrams and bigrams offered the highest F1 measure and second highest accuracy (range = (1, 2)).

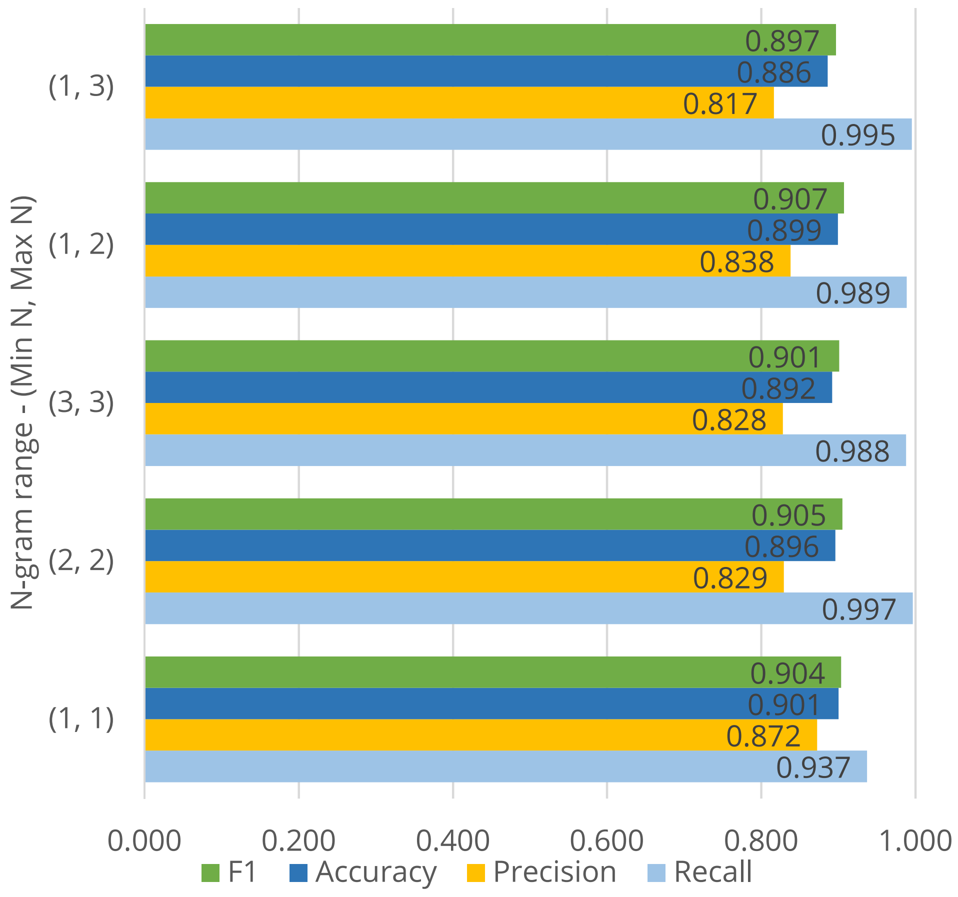

Figure 3 - Classifier metrics for the N-gram classifier with various n-gram range combinations from N = 1 to N =3.

Using this combination, optimisation found that the classifier performed best when trained on a 40% split with 17000 max features as shown in Figure 4.

Performing grid search optimisation on the (1,2) case produced the optimal classifier metric results shown in Table 2.

Table 2 - Classifier metrics for the TF-IDF classifier with 60% training set and a maximum of 19000 features.

Accuracy	**Precision**	**Recall**	**F1 Measure**
0.9096	0.9102	0.9080	0.9091

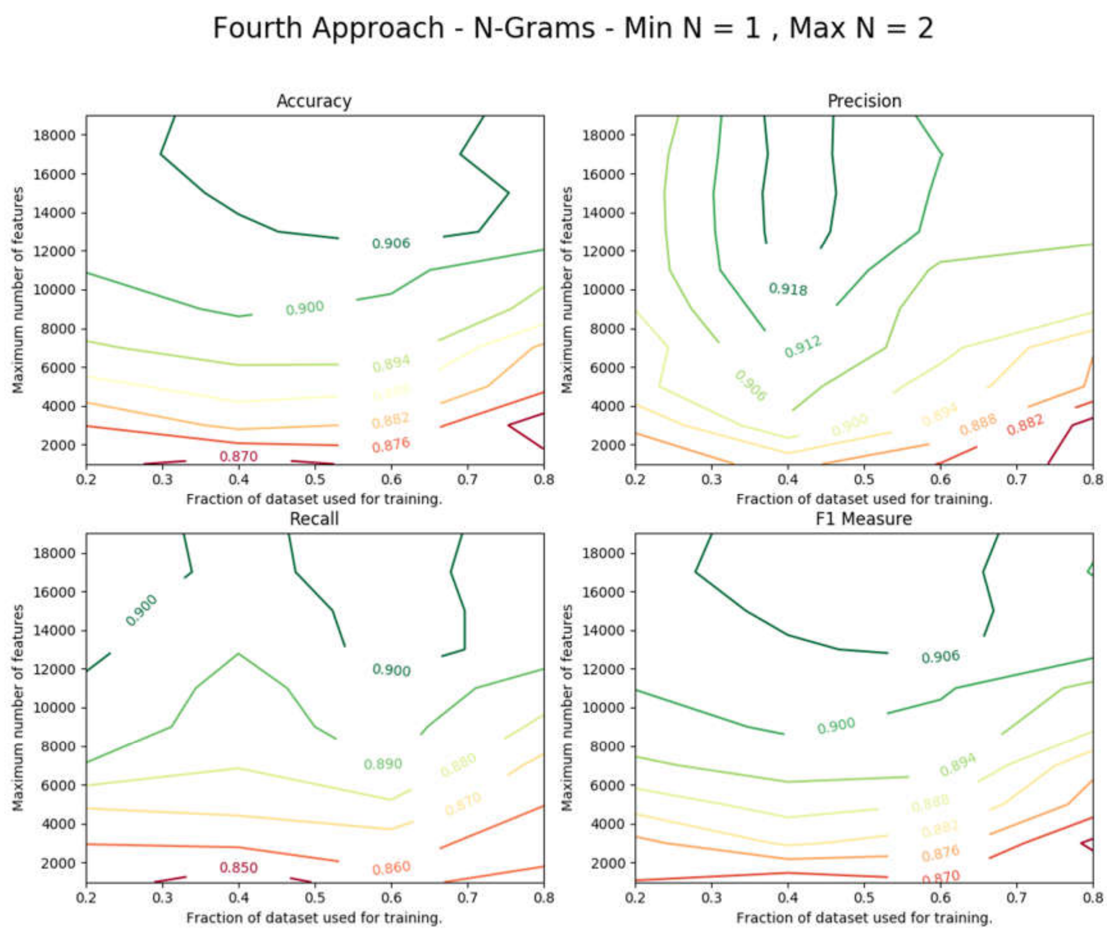

Figure 4 - Variation of classifier metrics for the combined Unigram and Bigram TF classifier, produced using a grid search method.

3 Deep Learning Approaches

Methodology

To implement a deep learning approach, a long short-term memory (LSTM) was built to model the articles in the dataset. The LSTM was implemented using Keras, a 'high-level neural networks API, written in Python' [6]. Keras was chosen due to its combination of power and simplicity, running on top of TensorFlow but providing a simple API.

Initially code for the implementation for the LSTM drew on an example implementation from the Keras documentation, where it was used to classify sentiment in an IMDB dataset [7]. The articles in the news dataset were tokenised and encoded into integers, with each integer value coding a unique token. The articles were then transformed using sequence padding provided by Keras, limiting the samples to the first 100 words of the article to reduce computational load and allowing for results to be obtained in reasonable times.

These samples were converted to a 128-dimension embedding using the 'Embedding' layer in Keras and passed to an LSTM with a dropout of 0.2 as specified in the IMDB example to prevent co-adaption and reduce overfitting [8]. The model was then compiled using an ADAM optimiser which was chosen for its computational efficiency [9]. The tests were then re-run using a dropout rate of 0.5, as suggested in [8].

The same process was repeated but with the word embeddings produced by a pretrained word2vec model provided by spaCy with 685k unique word vectors each encoded in 300 dimensions.

Results

Using the self-defined word-embeddings and a dropout rate of 0.2 allowed the production of a classifier with an accuracy of 0.8279. The process took around 15 minutes to run. Changing the dropout rate to 0.5 reduced the accuracy to 0.8185.

Due to hardware limitations, testing of the word2vec model was not possible.

4 Summary

Implementing a Multinomial Naïve Bayes model based on a selection of numerical features of the articles in the dataset yielded accurate and time efficient classifiers. The least effective of these shallow learning-based classifiers used only term frequencies of the tokenized dataset to achieve an accuracy of 0.876. Adding a lemmatizer and stopping words with low information content improved this accuracy to 0.901.

Implementing a classifier using TF-IDF numerical features in the model did not initially improve on this accuracy value, achieving an initial accuracy of 0.826. Optimisation through grid search improved the classifier’s accuracy to 0.910 when using a sample ratio of 40:60 (training samples: test samples).

Using bigrams as the numerical features gave an initial accuracy of 0.896 and the best initial F1 measure, 0.905. Optimisation through testing a number of N gram ranges and performing a grid search improved the accuracy to 0.9096 and the F1 measure to 0.9091.

A deep learning approach implementing a long short-term memory achieved an accuracy of 0.8279 and took significantly longer to execute that the shallow learning models. As a result of its long execution time, further investigation into deep learning approaches were abandoned.

The results from this report indicates that shallow learning models are a better choice for the task of fake news classification, both because of their increased accuracy and their increased time-efficiency.

Bibliography

[1] S. Learn, “sklearn.feature_extraction.text.CountVectorizer,” 2017. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html. [Accessed 11 03 2018].

[2] Scikit Learn, “sklearn.naive_bayes.MultinomialNB,” 2017. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html. [Accessed 11 03 2018].

[3] Natural Language Toolkit, “NLTK 3.2.5 Documentation - Tokenizer,” 2017. [Online]. Available: http://www.nltk.org/api/nltk.tokenize.html. [Accessed 12 03 2018].

[4] Natural Language Toolkit, “NLTK 3.2.5 Documentation - Source code for nltk.stem.worknet,” 2017. [Online]. Available: http://www.nltk.org/_modules/nltk/stem/wordnet.html. [Accessed 12 03 2018].

[5] C. D. Manning, P. Raghavan and H. Schutze, “Inverse Document Frequency - Introduction to Information Retrieval Book,” 2008. [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/inverse-document-frequency-1.html. [Accessed 12 03 2018].

[6] Keras, “Keras: The Python Deep Learning library,” 2018. [Online]. Available: https://keras.io/. [Accessed 12 03 2018].

[7] Keras, “Github - Keras - Examples - imdb-lstm.py,” 8 10 2017. [Online]. Available: https://github.com/keras-team/keras/blob/master/examples/imdb_lstm.py. [Accessed 12 03 2018].

[8] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *Department of Computer Science, University of Toronto,* 2012.

[9] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimisation,” *3rd International Conference for Learning Representations, San Diego,* 2015.