Big Data Summative Assignment

hzwr87

1 Dependencies

For this assignment, the main libraries used are Numpy, SkLearn and Keras (using the Tensorflow GPU backend). All of the Deep Learning experiments were run on a NVIDIA GeForce GTX 1050 graphics card with CUDA8 and CudNN6.

The GloVe embeddings in the Deep Learning component require the following text file - glove.6B.300d.txt - this can be downloaded from http://nlp.stanford.edu/data/glove.6B.zip.

The code is written to run the main function automatically when called from command line using "python3 main.py". No additional parameters are needed, it will run the best shallow and deep parameter sets and display the results. The provided dataset and GloVe file must be in the same directory as main.py.

The CudNNLSTM layer was used for this study for the runtime measurements, and is commented out in the implementation as it causes errors when no GPU is present. If testing on a GPU, superior runtime can be obtained by swapping the LSTM and CudNNLSTM layers on lines 236 and 237 of the code.

2 Preprocessing

The first preprocessing step was to convert all of the text to lowercase, remove newlines, special characters and multiple spaces. The purpose of this was to try and ensure that the text was in a uniform format, and try and differentiate between real and fake new based on text content alone. On examination of the dataset, an early observation was that many of the articles were web based, and contained many web URLS, twitter handles and hashtags. These were also removed, again for the purpose of differentiating based on text. There were also several examples which were either blank documents or individual words. The decision was made to remove any document containing less than 10 characters from the dataset, to ensure that each document contained sufficient text to learn from. This preprocessing was all done using python regular expressions.

3 Shallow Learning

3.1 Feature Extraction

Having cleaned the data, we now come to extract features. The two features used in this study are term frequency (tf) and term frequency-inverse document frequency (tf-idf). Both have been used here in order to determine the effect that taking into account the "importance" of each word to a document. For both of these features, we have also considered a range of n-grams. The range chosen here begin with unigrams alone and end with the combination of all n-grams up to 8. The inclusion of the combination of all these n-grams was due to the strength of the combination of the long distance dependencies of the larger n-grams and the local features of the smaller n-grams. We have not gone higher than 8-grams as up to this point the precision, recall, F1 score and accuracy had peaked and started to decrease, whereas the runtime is increasing.

3.2 Classification

After these features were extracted, they were passed into a Multinomial Naïve Bayes classifier. The main parameter for this classifier is alpha, which represents the level of additive smoothing. Through experimentation, it was established that the optimal value for this parameter was 0, i.e. no smoothing, and the results presented below were collected using this value. The data was split 80:20 between train and test, and the classification report of SciKitLearn was used to obtain the precision, recall and F1 score. The accuracy and runtime are also recorded. The optimal parameters were the combination of 1-4 grams using tf-idf, although the results were very close to the 1-3 grams, which had a shorter runtime. Both have been highlighted in bold in figure 1 below, as which is preferable would depend on whether runtime is being valued by the user or not.

Feature	n-gram range	Precision	Recall	F1-Score	Accuracy	Runtime
tf	1-1	0.8860	0.8849	0.8848	0.8849	4.98
	1-2	0.9054	0.9040	0.9039	0.9040	15.50
	1-3	**0.9129**	**0.9111**	**0.9110**	**0.9111**	**34.30**
	1-4	0.9126	0.9103	0.9102	0.9103	60.26
	1-5	0.9112	0.9087	0.9086	0.9087	84.27
	1-6	0.9121	0.9095	0.9094	0.9095	126.93
	1-7	0.9084	0.9056	0.9054	0.9056	153.57
	1-8	0.9082	0.9048	0.9046	0.9047	180.59
tf-idf	1-1	0.9016	0.9016	0.9016	0.9016	4.73
	1-2	0.9219	0.9206	0.9206	0.9206	16.28
	1-3	**0.9287**	**0.9270**	**0.9269**	**0.9270**	**33.20**
	1-4	**0.9289**	**0.9270**	**0.9269**	**0.9270**	**56.52**
	1-5	0.9282	0.9262	0.9261	0.9262	82.48
	1-6	0.9272	0.9246	0.9245	0.9246	133.54
	1-7	0.9258	0.9230	0.9229	0.9230	148.90
	1-8	0.9251	0.9222	0.9221	0.9222	183.23

Table 1: Results of Shallow Learning - the best results for each feature are highlighted in **bold**

4 Deep Learning

4.1 Feature Extraction

For the extraction of features for deep learning, the decision was made to use GloVe: Global Vectors for Word Representation. This algorithm is trained on "aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space"[3]. This was chosen instead of word2vec due to its superior runtime, which is achieved by being much more memory intensive. Given the computational resources available for this study, this was deemed to be an advantageous trade-off, as the results have been shown to be comparable and often superior to word2vec[2].

There were several different options of glove files to use, taken from Twitter, Wikipedia or Common Crawl, over various numbers of tokens, vocabulary sizes, case sensitivity and vector dimensions. As the preprocessing step made our data all lowercase, we do not require case sensitivity. Having experimented with all the possible GloVe files from [3], it was established that "glove.6B.300d.txt" was the best file to use, as it provides the best trade-off between performance and runtime. This file is taken from Wikipedia and Gigaword, which appears to be a more effective way of considering words used throughout the English language, as this dataset was taken from a variety of sources. However, the difference in classification performance between these files was very small.

4.2 Classification

From here, these features are passed to a Neural Network, written in Keras. The most important layers in this model are the LSTM and RNN layers, so those will be the main point of comparison addressed in this report. The remainder of the network is standardised to both approaches. The architecture that has been chosen is two convolutional layers with 128 filters each, and a kernel size of 5. Between each of these is a max pooling layer of pool size 5. This is then passed to the LSTM/RNN, followed by a dropout of 0.25, a dense layer of 128 units and finally a single unit dense layer with a sigmoid activation function. Sigmoid was chosen as the training labels only vary between 0 and 1, and the probabilities returned by the sigmoid activation function would match up easily with these.

The convolutional layers are used for the same reason as the n-grams from the shallow learning section, as they break the text into smaller regions to make use of intra-text dependencies. The pooling layers downsample the output of these convolutions to the correct number of dimensions for the LSTM or RNN layers to use. The final dense layer then collates the output of the LSTM/RNN before going to the sigmoid based output layer.

For the LSTM and RNN, the main parameter to be set is the units, or the dimensionality of the the output space. This was tested for both layer types for the range 8-1024. Powers of 2 were selected here due to them providing higher efficiency. The training was performed over 20 epochs, as it was clear that no great advantage could be gained by running for longer. Runtime is included here to illustrate the effect of increasing the number of units massively has, as the runtime is very similar for the majority of the parameters tested. It is also very similar between both the LSTM and RNN in the majority of cases, but the LSTM runtime increases substantially more than the RNN runtime as the number of units is increased.

Feature	Units	Precision	Recall	F1-Score	Accuracy	Runtime
LSTM	8	0.9219	0.9206	0.9206	0.9206	126.02
	16	0.9359	0.9357	0.9357	0.9357	122.57
	32	0.9248	0.9246	0.9246	0.9246	122.86
	64	0.9310	0.9310	0.9310	0.9310	124.23
	128	**0.9365**	**0.9365**	**0.9365**	**0.9365**	**127.85**
	256	0.9333	0.9333	0.9333	0.9333	129.78
	512	0.9278	0.9278	0.9278	0.9278	143.15
	1024	0.9262	0.9262	0.9262	0.9262	191.64
RNN	8	0.8223	0.8222	0.8222	0.8222	123.12
	16	0.8100	0.8087	0.8085	0.8087	124.47
	32	0.8025	0.8025	0.8025	0.8025	124.62
	64	**0.8897**	**0.8897**	**0.8897**	**0.8897**	**124.83**
	128	0.8140	0.8135	0.8134	0.8135	125.32
	256	0.7882	0.7873	0.7871	0.7873	125.95
	512	0.7291	0.7278	0.7274	0.7278	130.61
	1024	0.6077	0.5870	0.5668	0.5873	146.92

Table 2: Results of Deep Learning - the best results for each feature are highlighted in **bold**

5 Conclusion

The best results obtained on this dataset were using an LSTM with 128 units, giving us an accuracy of 93.65%. The runtime of the deep implementation is substantially longer than the shallow for a relatively small increase in performance. Therefore, whether shallow or deep would be preferred would depend on how much runtime is valued by the user.

Further work would examine deeper architectures with more layers, maybe even using multiple convolutional layers together with different kernel sizes to represent the combination of multiple n-grams. Useful data that could be used to solve this probelm that was not included in the corpus, such as title oir source, could also be experimented with to try to obtain better results.

References

[1] Francois Chollet. Using pre-trained word embeddings in a Keras model. `https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html`, 2016.

[2] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[3] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. `https://nlp.stanford.edu/projects/glove/`, 2014.