Big Data Summative

fdqd43 - James Paterson

March 12, 2018

1 Running the program

To run the program, invoke `index.py` from the command line. There are many different options you can pass to the program to enable the different modes.

```
Usage: python index.py
        -f <filename> Open filename as for the data
        -v <number> Designate a vocab size as the argument
        -n Enable noisy (debug) mode
        Options for shallow:
                -a Use all terms as vocab (no limit)
                -i Turn off TF_IDF
                -g Turn off Ngrams
        Options for deep:
                -d Use deep learning rather than shallow learning
                -o Turn off the dropout layer
                -e <number> Set the number of epochs to run for
                -b <number> Set the batch size
                -r Use RNN rather than LSTM
```

Below are some example commands that I have used to obtain the data in this report.

Run a shallow learning with N-grams and TF-IDF –	`python index.py`
Run a shallow learning with just TF –	`python index.py -i -g`
Run a shallow learning with TF-IDF –	`python index.py -g`
Run a shallow learning with TF and N-Grams –	`python index.py -i`
Run a shallow learning with just TF and custom vocabulary size –	`python index.py -i -g -v 5000`
Run a deep learning with the LSTM –	`python index.py -d`
Run a deep learning with the RNN –	`python index.py -d -r`
Run a deep learning with the LSTM and a custom number of epochs –	`python index.py -d -e 2`

You will also need to have the following libraries installed:

```
        numpy
        keras
        pandas
        collections
        nltk & nltk "en stopwords"
        spacy & spacy "en_vectors_web_lg" dataset
        sklearn
        unidecode
```

2 Shallow Processing

Implemented in the file `shallow.py` are three different classifiers based on shallow learning, TF, TF-IDF, and N-Grams.

2.1 Data Cleaning

The data from the csv is cleaned in several ways, in the function `cleanText()`. First, the text is encoded to unicode, and then stripped of all non ASCII characters. This is to reduce the number of characters we have to process, and to fix terms where unicode entities are used to represent spaces or odd punctuation such as speech marks or long dashes. The text is then stripped of all punctuation and converted to lowercase using `text_to_word_sequence()`, which also breaks the string into a list of words. Stop words are then removed (The list from NLTK is used). The list of words is then converted back to a string.

2.2 Algorithms

2.2.1 TF

The term frequency algorithm that is the base of all three methods is implemented using `calculateTermFrequency`. It uses the `CountVectorizer` from SciKitLearn to retrieve the frequency matrix and list of terms.

2.2.2 TF-IDF

If TF-IDF is turned on (It is by default), then the term frequency matrix is then converted into the TF-IDF using the `TfidfTransformer` also from SciKitLearn.

2.2.3 Ngrams

If N-Grams are turned on (Also by default), then `calculateTermFrequency` is called twice more to get the 2-grams and 3-grams from the corpus. There are processed much the same way as the unigram frequency terms. If TF-IDF mode is also enabled then these additional frequency matrices are also converted into the TF-IDF format.

2.3 Bayesian Classifier

The classifier is self written and is in the file `bayes.py`. The functions are called by the `documentClass` function, which calculates some constants beforehand to speed up the execution, such as the sum of the frequencies for each class. The classifier, in TF and TF-IDF mode, works simply by returning the log probability of the word given the class. Laplacian smoothing is used to avoid zero probabilities, and additionally because it is a cheap and effective method for doing this in the scenario of text classification. In N-Grams mode, instead of simply returning the probability, the N-Grams formula for probability (Shown below) is used instead, with hard coded weighting for lambdas.

$$\begin{aligned} P(w_n|w_{n-1}w_{n-2}) = {} & 0.125P(w_n) \\ & + 0.375P(w_n|w_{n-1}) \\ & + 0.5P(w_n|w_{n-1}w_{n-2}) \end{aligned}$$

The weightings assume that if a 3-Gram is found, it's presence is more significant than a 2-gram, which is more significant than a unigram.

2.4 Results

Results generally scale with the amount of terms in the vocabulary. Accuracy results for a test size of 500 are shown below when a variety of vocabulary sizes are tried as well as the elapsed time to train and test. Results are given with the following measures:

C –	Classification Accuracy %
R –	Recall Measure
P –	Precision Measure
F –	F Measure
Time –	Time in seconds to train and test 500 articles

	TF Accuracy Rate				
Vocabulary Size	**C**	**R**	**P**	**F**	***Time***
7500	86.0	0.784	0.941	0.855	58.8
15000	88.2	0.852	0.918	0.884	91.8
30000	90.0	0.894	0.915	0.904	191.1
Unbounded (65063)	90.8	0.917	0.910	0.913	707.3

Table 1: Term Frequency Results

Term frequency produces an impressive 90.8% classification accuracy when using an unbounded vocabulary, and close to that at around half the vocabulary. It is also very quick to execute and increasing the vocabulary results in steady increases in all metrics except precision, which drops very slightly.

	TF-IDF Accuracy Rate				
Vocabulary Size	**C**	**R**	**P**	**F**	***Time***
7500	89.8	0.879	0.924	0.900	58.6
15000	89.6	0.939	0.873	0.905	104.3
30000	87.4	0.970	0.823	0.890	296.8
Unbounded (65063)	81.1	0.989	0.740	0.846	598.6

Table 2: Term Frequency Inverse Document Frequency Results

TF-IDF produces immediate gains for the same amount of processing time for lower vocabulary sizes. All metrics are improved upon except for precision for both the 7500 and 15000 vocabulary sizes. However, paradoxically, the larger the vocabulary, the worse classification accuracy we obtain. This is likely to be because of the lower frequency terms obtained when using a larger vocabulary polluting the probabilities and throwing off the balance, as these terms are given higher weight with the inverse document frequency measure. However, recall still increases, meaning most real articles are classified as such (But many fake articles are also classified as real).

	N-Grams TF Accuracy Rate				
Vocabulary Size	**C**	**R**	**P**	**F**	***Time***
7500	91.0	0.943	0.892	0.917	405.94
15000	91.4	0.951	0.893	0.921	1098.8

Table 3: N Grams TF Results

	N-Grams IDF Accuracy Rate				
Vocabulary Size	**C**	**R**	**P**	**F**	***Time***
7500	91.0	0.962	0.879	0.919	511.3
15000	89.0	0.977	0.840	0.904	1028.0

Table 4: TF-IDF and N-Grams Results

The results for N-Grams are better than that of any other method, however the execution time leaves much to be desired, taking almost ten times the amount of time as the TF or TF-IDF methods. The same trends are present as in TF, with general increases in all statistics as vocabulary increases, and with TF-IDF a general decrease in most statistics as vocabulary increases. Additionally, I have not run the N-Grams algorithm for larger vocabulary sizes as it often results in out of memory errors due to to much space being taken up by storing the N-Grams.

2.5 Comparision

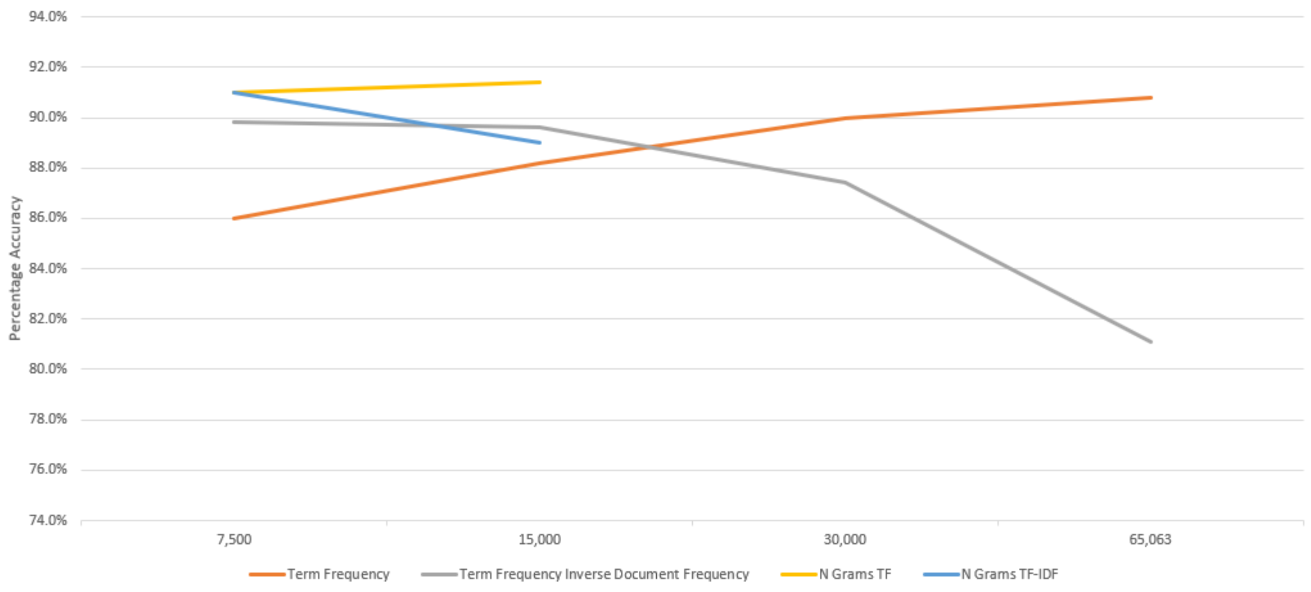

Figure 1: Graph of classification accuracy vs vocabulary size for all four variants

TF and TF-IDF exhibit good performance at great speed when the vocabulary size is of a manageable amount, usually executing in less than two minutes. For TF, the performance follows the law of diminishing returns as the vocabulary size is increased, topping out at 90.8%. TF-IDF is predictably slightly better at classifying than TF, due to the additional complexity of the classifier. This results in gains of about $2-3\%$ in the percentage of correctly classified news articles for lower vocabulary test runs. TF-IDF does also not increase processing time by a large amount, and in extreme cases even decreased the processing time.
However, enabling N-Grams does. Almost 10 times as much time and memory are used, due to the vocabulary becoming more than three times larger as all the unigrams, 2-grams and 3-grams are stored. Additional time is also needed to calculate the more complicated probabilities. This does result in slight gains in accuracy of about $1-2\%$ over even the best TF results, however it is likely not worth the additional processing time required. Infact, for larger vocabulary sizes, the N-Grams procedure simply crashes due to not having enough memory to store all of the terms.

2.6 Shallow Learning Conclusion

Considering all factors, TF-IDF is the most balanced shallow learning algorithm due to it's simplicity, it's low execution time, and little need for large vocabulary sizes to get almost perfect results. However it is also worth noting that all measures produce far better classification than the baseline of 75%, even at low vocabulary counts.

3 Deep Learning

The deep learning section is contained within the file `lstm.py`. The LSTM is built with the `keras` library, and word2vec features are extracted from the `en_vectors_web_lg` spacy vector bank.

3.1 Cleaning Text

The cleaning text portion is slightly less thorough than in the shallow learning. Unicode characters are still replaced, punctuation is removed, and the text is converted to lowercase, but no stop words are deleted.

3.2 Converting and Encoding

Each article is cleaned one by one, and the resultant strings are concatenated. From this long string the 10,000 most common words are identified, and used as our vocabulary. A spacy vector representing a word2vec feature is then extracted for each of these words. The function that returns the word2vec features is named `textToVectors`. If the word is not in the dictionary, then a zero vector of the same size (300) is used instead. The words are then encoded using the `one_hot` function, and a mapping of words to integers is then built. Two extra words, the `<OTHER>` character, used to represent a word not in our most common words list, and the `<PAD>` character, used to pad shorter sequences to be of the correct length. Each article is then converted to a list of integers with this mapping, unknown words being replaced with the `<OTHER>` character. They are then either left padded with our pad character, or truncated so that each article is 1000 words long.

3.3 LSTM Structure

The neural network consists of a starting embedding layer with size `(vocab_size, 300)`, as the vectors obtained from spacy are of size 300. This is then fed to the LSTM layer, which outputs to 64, and this is followed by a Dropout layer that zeros 0.1 of the input. This layer was added because overfitting was happening at epoch 3. This is then fed into a Dense layer to output the probability. This layer uses a `sigmoid` activation function. The activation was chosen as it is often used for binary classification problems, owing to the fact that it tends to bring values to either side of the curve, as is required. The RNN model is the same, however of course instead of an LSTM layer there is instead a SimpleRNN layer.

3.4 Results

Results were run with a batch size of 32, and over 2 epochs, and with a vocabulary size of 10000. 500 articles were used to test the model after it had finished training.

	Deep Learning				
Type	**C**	**R**	**P**	**F**	***Time***
LSTM	82.6	0.872	0.799	0.834	725.2
RNN	79.2	0.904	0.739	0.813	184.8

Table 5: RNN and LSTM results

The RNN results were much faster than the LSTM, owing to the less complicated architecture, but the results were worse. This is likely because news has long term dependencies that are better expressed by an LSTM. Although an additional 3% of classification accuracy was generated by the LSTM, the training and testing procedure took more than three times as long to complete, which raises the question as to whether it would be better to train the RNN on a more complex network, or with a larger vocabulary, for the same amount of time. Both results are still above baseline, but not as impressive as those produced by a shallow method.

4 Conclusion

Overall, it is clear to see that that shallow learning methods are far superior in this particular problem domain. Not only do they achieve higher classification accuracy and F1 Measure than the deep methods, they also do it in far less time than either. In particular TF-IDF achieves an extremely high rate of correct classification in a relatively short time compared to other methods.
This superiority of the shallow methods is likely due to the problem domain. Viewing a document as a "bag of words" model is sufficient to classify it as real or fake news. This is due to the fact that different words are ostensibly used by the authors of real and fake news, making it easy to tell them apart. For example, a fake news author may use more slurs or write more sensationally to generate more interest. Additionally, using a shallow algorithm allows the whole article to be examined, instead of only the first 1000 words. Although this is not a problem for articles shorter than this limit, longer articles may benefit from the whole article being examined.