Fake News Assignment

cmkv68

March 4, 2018

Program Prerequisites

The python program utilises numpy, keras and nltk. Keras by default installs tensorflow, but it is recommended to install tensorflow manually to optimise its performance (as it's own installion compiles the program based on the user's machine, whereas installing via pip3 does not.) The program will attempt to download the GloVe dataset itself. (It cannot however, show a loading bar but it will verify integrity!)

```
pip3 install numpy nltk tensorflow keras
python3 main.py
```

Testing Configuration

For the sake of setup, both learning techniques are subject to a machine utilising an `intel i7-6600u` with 12 Gigabytes of DD4 memory, alongside NVM-E storage. Due to the relatively large nature of the word2vec dataset which will be used, running times may vary dependent the hardware utilised such as the storage, memory and the processor itself.

Data Sanitisation

Each document's article is converted into lowercase and is then subject to the ordered removal of:

- Strong quotation marks
- Commas between numbers (and joining numbers together)
- Apostrophes between letters, alongside the suffix (which indicate plurality, posessive cases or contrations)
- URLs
- Line breaks
- Unicode strings
- Duplicate quotes
- Punctuation
- Stopwords

The removal is ordered as removing punction entirely in one farce may produce unintended effects, i.e `20,000,000 => 20 000 000` or `John Lennon's => john lennon s`.

The reasoning behind the removal of stopwords is related to the fact that the words themselves are used for grammatical understanding, which are not considered in our learning methods, especially in the case for shallow learning.

Once computed, a dataset object is created, containing each document in the dataset. Each document is an object itself, containing the raw article, and the cleaned data, (now a list of words in python). While this is

not necessarily the most efficent method, it was chosen to help understand the learning mechanisms used in the assignment.

The documents are split such that there is 300 documents which are used as test data and the rest is used as training data.

Shallow Learning

Whilst there exist a variety of packages that may facilitate the shallow learning process, I have chosen to implement them manually to help understand the inner workings of how shallow learning techniques work. There is naturally a tradeoff however. For instance, the implementation I may have used might be not as efficent as the libraries.

Term Frequency Inverse Document Frequency (TF-IDF) is weighting scheme that is commonly used in information retrieval tasks. The idea is to treat each document as a bag of words, ignoring the exact ordering of the words in the document while retaining information about the occurrences of each word.

Term Frequency

Each document is iterated, with each word being counted relative to the rest of the other words in the document. This is then stored in the document's object in a dictionary, where each word is a key, and their word count is the value.

Term-Frequency/Inverse Document Frequency

A global document frequency μ for words is introduced, using the training documents's words. For each document, we iterate through its term frequency and accumulate them to μ. This ensures that each unique word is counted per document as opposed to their word-count/term-frequency. Once this is accumulated, Each document is iterated through and a TF-IDF value is calculated for each word w using the formula $tfidf(w) = tf(w) * df(w)^{-1}$. TF-IDF values are stored in the same style as how term-frequency is stored in a document's object.

N-Grams

There is the option to use either bigrams or trigrams (unigrams can be considered equal as term-frequency.), when using the n-grams option. Using the `NLTK` library, N-Grams are produced, using the document's cleaned word-list and are stored in the document's object.

```
trigrams([the quick brown fox jumped over the lazy dog])

  (the quick brown)
  (quick brown fox)
  (brown fox jumped)
  ...
  (the lazy dog)
```

Naive Bayes Classifier

Words that are found in the test document but not in the training document are **ignored.** This is chosen to reinforce the value of the training data.

With α as the conditional probability of a word or a wordgroup for a given class, ϵ being the class itself, the naive bayes classifier is composed of the formula:

$$\beta = (count(\alpha, \epsilon) + 1)/(count(\epsilon) + vocabulary)$$
$$P(X|Y) = \beta \cdot \gamma$$

Where γ represents the probability of a class. For the n-grams model, they are treated in the same vein as term-frequency; for instance we collate the frequency of a particular n-gram to calculate their conditional probability. For TF-IDF, we substitute the term frequency with their tf-idf value.

This formula is used on each of the classes (fake and real), and the class is chosen based on whether its probability is higher than the other.

Deep Learning

For deep learning, we subject each document using a public dataset, the glove word2vec model (which can be found at https://nlp.stanford.edu/projects/glove/) as a premise to compare the documents against.

Four different activation functions are tested to maximise the performance of both the LSTM and the regular recurrent neural network, Linear, ReLu, Sigmoid, and tanh. Whilst Linear and ReLu are quite similar, they all provide different ranges. Training was performed with one epoch, but was repeated 10 times to deduce an average.

Otherwise, the structure for both LSTM and the RNN models remain the same. A binary entropy loss function is used to reduce the chances of overfitting the training data. Root Mean Square propagation is chosen to adapt the learning rate for each of the parameters. This goes in hand with the training data, as it has varying article sizes (which could be either very short or extensibly long.) Dropout layers are used to also help reduce the chances of overfitting of the data.

Sigmoid is used as our activation function. This is a natural decision, due to its range (from 0 to 1 inclusive), which fits well with the probability ranges needed to determine the two classes.

The vanishing gradient isn't necessarily a big issue due to the number of layers involved (which is not enough to justify the use of linear activation functions such as ReLu, as shown in our results).

During testing, there may be the case where words that are in the test set may not exist in the training data or in the glove dataset. In this situation, we create a new vector that utilises the mean of the vectors of the first 1000 glove words. (Note that the glove dataset is ordered in terms of word frequency, so the most popular words appear first, and so on.) This is to help create a fair comparison between the shallow and deep learning methods. Naturally, sentence padding utilises a zero vector.

Results

Shallow

Method	Precision	Accuracy	Recall	F1 Measure	Time Taken
TF	50.38	88.00	87.50	63.94	Negligible
TF-IDF	51.70	88.33	86.16	64.62	Negligible
Trigrams	50.34	99.33	98.68	66.67	Negligible
Bigrams	50.34	99.33	98.68	66.67	Negligible

Interestingly, the performance difference between TF and TF-IDF is within margin of error. This may be related to the fact that words that may be considered significant on the test set might not be in the training data. This can also explain why TF has a higher recall score i.e. it returns more relevant results than TF-IDF, and that the precision score of TF-IDF is especially high compared to the other methods.

Unsurprisingly, the n-grams method performs strongest, but it may be the case that the use of n-grams would be too sparse, given the size of the dataset.

Deep

Without Weights

Method	Precision	Accuracy	Recall	F1 Measure	Time Taken (Per Epoch)
LSTM (E1)	64.0	85.667	73.909	67.234	40s
RNN (E1)	55.376	66.0	54.152	50.386	23s
LSTM (E2)	74.667	90.333	61.606	67.331	41s
RNN (E2)	62.359	69.333	32.303	42.474	23s

With Weights

Method	Precision	Accuracy	Recall	F1 Measure	Time Taken (Per Epoch)
LSTM (E1)	66.21	71.12	63.70	63.771	62s
RNN (E1)	59.32	74.33	65.17	57.581	20s
LSTM (E10)	71.57	93.333	80.061	74.744	55s
RNN (E10)	64.74	78.670	59.42	60.56	23s

An interesting fact is that the inclusion of weights reduces the initial performance of the LSTM and CNN. This may be due to the fact that words that are not considered in the glove dataset have a weighting of 0 and may spoil the propagation of values on a recurrent neural network. It may also relate to the fact that popular words are indexed with a lower value, i.e. less common words will have an higher index. As we are feeding the indexes of the words into Keras, it would take that into account (notwithstanding the weights). This is is the case for 1 epoch however, running more epochs naturally improved the quality of the classifier, visibly showing a higher level of accuracy for the model with weights.

Conclusion

In terms of shallow feature extraction, it is quite difficult to justify using TF-IDF for a classifier task as its intended use is not necessarily relevant. Naive-Bayes is commonly defined using TF as the premise, and we can see that it is sufficent to provide a high degree of accuracy. That being said, the use of n-grams has shown to perform to a near-perfect accuracy.

Naive Bayes is a very strong generative classifier that requires very little data for it to work effectively, and the same can be said for LSTMs (or RNNs in general) as a discriminative model. They are however, very different methods of solving the same task. This disparity is represented through the inherent tradeoffs between the two. Naive-Bayes runs in a fraction of the time compared to Recurrent Neural Networks, and from our training data the immediate run shows naive bayes performing better overall on one epoch compared to our deep learning models.

RNN's are naturally sequential, and for words that do not exist in the training data, we can see the distruption in performance, as shown in our results.

I would argue that the dataset is not large enough to justify the use of a RNN. A strong performance can be seen for the shallow learning methods (tf, n-grams) in the fraction of the time of the deep learning methods. Within the immediate context, this would suffice.

One could further the performance of both classifiers by either feeding the test data into the training dataset, and, in the case for RNNs, we could increase the epochs (the runtime over the dataset).