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.

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

Testing Configuration

Both learning methods 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.

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

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 worth 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 adapte 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.)

Preliminary Activation Function Results

model	loss	accuracy	precision	recall	f1measure	activation
cnn	56.329	68.9	58.024	54.824	51.758	linear
cnn	56.164	71.167	58.173	59.009	54.166	sigmoid
cnn	55.59	71.2	59.501	58.83	54.626	tanh
lstm	52.839	74.3	62.782	59.082	56.864	ReLu
lstm	52.938	73.967	61.221	62.482	57.105	sigmoid
lstm	50.77	75.833	64.396	59.558	57.74	tanh
cnn	53.409	72.8	57.906	66.348	57.772	ReLu
lstm	51.656	75.133	61.157	66.752	59.706	linear

Going with the F1 Measure (which is the harmonic mean of precision and recall), the linear activation function performs the best with our LSTM, and ReLu with our RNN. Oddly enough,

However, tanh performs second best for both our LSTM and RNN, and is therefore chosen as our activation function as choice.

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).

Results

Shallow

Method	Precision	Accuracy	Recall	F1 Measure	Time Taken
TF	49.81	87.68	87.92	63.59	Negligible
TF-IDF	94.19	51.67	50.87	66.06	Negligible

Method	Precision	Accuracy	Recall	F1 Measure	Time Taken
Trigrams	50.34	99.33	98.68	66.67	Negligible
Bigrams	50.34	99.33	98.68	66.67	Negligible

Out of our shallow learning methods, we can see

TF-IDF performs the worst in terms of accuracy, which makes sense. Words that are necessarily unique to a given test document may not be considered on training data, which makes it difficult to precisely quantify the significance of those particular words. It also happens to be the case that those words would hold more weight than other words that are considered otherwise in other documents.

In other terms, TF-IDF doesn't differentiate between the terms and the perspective of conveyed sentiments (as is the case with all shallow learning methods). For instance, we assume the training set has the words "toaster", "bad", "good" and the test data contains "magnet", "good". Since we have a weighting for "good" as it is in the training set, we do not have any information on the term "magnet", even though the term itself holds more semantic information than "good".

This may also help show why the precision score of TF-IDF is especially high compared to the other methods.

Deep

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

Running more epochs can naturally improve the quality of the classifier with the tradeoff of introducing possible overfitting of the data.

An interesting fact is that the inclusion of weights reduces the overall 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.

Due to the inherent differences between the two approaches, it makes it very difficu

Conclusion

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.

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).