Sentiment Analysis on the IMDB Movies Database

Contents

Introduction

This capstone project is to understand the various paradigm involved in Sentiment Analysis of texts. In a largely online world Sentiment analysis has become an important tool to understand the user/ buyers sentiment towards a movie /video / product.

Sentiment Analysis

The process of computationally identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writer's attitude towards a particular topic, product, etc. is positive, negative, or neutral.

Generally speaking, sentiment analysis aims to determine the attitude of a speaker, writer, or other subject with respect to some topic or the overall contextual polarity or emotional reaction to a document, interaction, or event. The attitude may be a judgment , happiness, review , comment , feedback etc

In this project I am using the IMDB Movies review data. This project demonstrates about how to

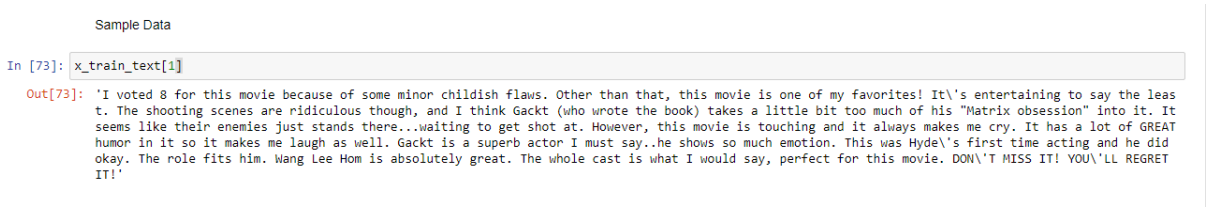
- Prepare Data for Sentiment Analysis
- Use LSTM techniques to build a sentiment analyser
- Predict sample 5000 rows from the test data and check the output of prediction

Getting The Data

The dataset is compiled from a collection of 50,000 reviews from IMDB on the condition there are no more than 30 reviews per movie. The numbers of positive and negative reviews are equal. Negative reviews have scores less or equal than 4 out of 10 while a positive review have score greater or equal than 7 out of 10. Neutral reviews are not included. The 50,000 reviews are divided evenly into the training and test set.

This can be downloaded from
: http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz.

Sample Data



Pre Processing the Data

As we can see the data contains a lot of generally used words like “a”, “for”, “the” – i.e commonly used stop words. As a first step we clean up the stop words using NLTK library

In [153]:
```
# As we can see the data has lot of stop words removing stop words before processing using keras
def filter_stop_words(train_sentences, stop_words):
    for i, sentence in enumerate(train_sentences):
        new_sent = [word for word in sentence.split() if word not in stop_words]
        train_sentences[i] = ' '.join(new_sent)
    return train_sentences
```

In [77]:
```
stop_words = set(stopwords.words("english"))
x_train_text = filter_stop_words(x_train_text, stop_words)
x_test_text = filter_stop_words(x_test_text, stop_words)
data_all = x_train_text + x_test_text
```

In [78]:
```
num_words = 1000
```

Tokenising the Data

Create Vocab to Int mapping dictionary

In most of the NLP tasks, we need to create an index mapping dictionary in such a way that the frequently occurring words are assigned lower indexes. We use the Tokeniser library from KERAS to achieve this. In the word-to-index dictionary, each word in the corpus is used as a key, while a corresponding unique index is used as the value for the key

.

As a first step, we will use the Tokenizer class from the keras.preprocessing.text module to create a word-to-index dictionary. In the word-to-index dictionary, each word in the corpus is used as a key, while a corresponding unique index is used as the value for the key.

```
#Create Vocab to Int mapping dictionary using Toeknizer from keras

tokenizer = Tokenizer(num_words=num_words)
```

```
## Fit the tokenizer on all the  data

tokenizer.fit_on_texts(data_all)
```

```
# Get the word index from the train data
tokenizer.word_index
```

```
{'br': 1,
 'i': 2,
 'the': 3,
 'movie': 4,
 'film': 5,
 'one': 6,
 'it': 7,
 'like': 8,
 'this': 9,
 'good': 10
```

Truncate and pad the review sequences

We set the maximum size of each list to 250. The lists with size greater than 250 will be truncated to 250.This process is called padding We need to have text samples of the same length in order to feed them into our neural network. If reviews are shorter than 250 words we will pad them with zeros.

Introduction to the Model

With this the pre processing steps of the data have been completed. Nowe we proceed towards the model set up. Our model has 2 parts / concepts associated with it

1) Word embedding
2) LSTM

Word Embedding

A word embedding is a learned representation for text where words that have the same meaning have a similar representation. The embedding layer will learn a word embedding for all the words in the dataset.

It has three arguments the input_dimension in our case the 1000 words. The output dimension aka the vector space in which words will be embedded. In our case we have chosen 8 dimensions so a vector of the length of 32 to hold our word coordinates.

```
In [103]: model = Sequential()
          embedding_size = 8
          model.add(Embedding(input_dim=num_words,
                              output_dim=embedding_size,
                              input_length=max_len,
                              name='layer_embedding'))
```

So our output of the embedding layer is a 1000 times 8 matrix. Each word is represented through its position in those 8 dimensions. And the sequence is the 1000 words that we feed into the LSTM network.

Long Short Term Memory Model

Long short-term memory is an artificial recurrent neural network architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points, but also entire sequences of data.

Long Short-Term Memory networks (LSTM) are capable of learning the relationships between elements in an input sequence. In our case the elements are words.

We build a 3 layer LSTM model

Layer 1 has 16 units, Layer 2 has 8 units and Layer 3 has 4 units. Layer 1 and 2 return sequence as true. Finally at the end we have a dense layer with one node with a sigmoid activation as the output

```
n [104]: model.add(LSTM(units=16, return_sequences=True))
         model.add(LSTM(units=8, return_sequences=True))
         model.add(LSTM(units=4))
         model.add(Dense(1))
```

Since we are going to have only the decision when the review is positive or negative we will use binary_crossentropy for the loss function. The optimizer is the standard Adam optimiser and the metrics are also the standard accuracy metric.

```
In [105]: optimizer = Adam(lr=1e-3)
          model.compile(loss='binary_crossentropy',
                        optimizer=optimizer,
                        metrics=['accuracy'])
```

Model Summary

```
In [106]: model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer_embedding (Embedding)  (None, 250, 8)            8000
_________________________________________________________________
lstm_6 (LSTM)                (None, 250, 16)           1600
_________________________________________________________________
lstm_7 (LSTM)                (None, 250, 8)            800
_________________________________________________________________
lstm_8 (LSTM)                (None, 4)                 208
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 5
=================================================================
Total params: 10,613
Trainable params: 10,613
Non-trainable params: 0
_________________________________________________________________
```

The model is configured to train on 23750 samples and validate on 1250 samples

The model is trained and tested.

Results

We get a test accuracy of 82.652%. Our training accuracy was 83.29%. This means that our model is correctly fitting on the training set. The performance difference between training and test sets should be minimum. We have achieved a good accuracy of over 80 %

```
In [112]: history = model.fit(X_train, y_train,
                    validation_split=0.05, epochs=1, batch_size=64)
```

```
Train on 23750 samples, validate on 1250 samples
Epoch 1/1
23750/23750 [==============================] - 1124s 47ms/step - loss: 0.3889 - acc: 0.8329 - val_loss: 0.5020 - val_acc: 0.7376
```

To evaluate the performance of the model, we can simply pass the test set to the evaluate method of our model.

```
In [113]: result = model.evaluate(X_test, y_test)
          print("Accuracy: {0:.2%}".format(result[1]))
```

```
25000/25000 [==============================] - 44s 2ms/step
Accuracy: 82.65%
```

To check the test accuracy and loss, execute the following script:

```
In [114]: print("Test Score:", result[0])
          print("Test Accuracy:", result[1])
```

```
Test Score: 0.37567781929016114
Test Accuracy: 0.82652
```

Predictions using the model

We test the predictions against 5000 rows from test data. These predicted numbers would be between 0.0 and 1.0. So we set a threshold value of 0.5,and say that all values above 0.5 are taken to be 1.0 and all values below 0.5 are taken to be 0.0. This gives us predicted "class" of either 0.0 or 1.0. We observed that 334 rows out of 5000 have wrong or mis classified data

Sample mis-classified text is:

```
In [148]: text = x_test_text[idx]
          text
```

```
Out[148]: "seriously people need lighten accept funny funny, movie f**king hilarious. Better first Knoxville really grew pair film way crazier stunts first. If
Ebert Roper(not saying I'm huge fan theirs) look past pure idiocy film enough give 2 thumbs think people to. I sure expect floored rare sequel better
original. This new one I believe exceeds first big time. did,just relax kick back try barf points laugh ass off."
```

The predicted value of sentiment is

```
In [151]: y_pred[idx]
```

```
Out[151]: 0.28527364
```

The Actual Sentiment Value is

```
In [152]: cls_true[idx]
```

```
Out[152]: 1.0
```

Conclusion

In a new age digital world there is a lot of content generated. We also have a substantial number of people commenting on the content. Sentiment Analysis acts as a great tool for companies, business and individuals to assess the overall feeling of general public towards their work / products