Sentiment Analysis with Neural Networks

Alexander Johns

April 14, 2020

1 Introduction

1.1 Sentiment Analysis

Sentiment Analysis is the process of using a computer to identify and categorize opinions in a piece of text in order to determine the writer's attitude toward a particular product, topic, service and so on.

1.2 Data Pre-processing

1.2.1 Tokenization and Lemmatization

When given a document or character sequence, tokenization is the process of chopping it up into smaller pieces, known as tokens. There are many different ways one can go about tokenization, breaking up a document by paragraph, by sentence or most commonly by word. In the context of data pre processing for sentiment analysis documents are often tokenized into words. Lemmatization is a textual data preprocessing tool that aims to remove inflectional endings on words in an attempt to return them to their dictionary form, known as a lemma. In sentiment analysis it is important to lemmatize words as the input data passed into neural networks or machine learning algorithm often consists of matrices which aim to represent the frequency of different words within the text. If you consider different tenses of the same word as different features then it may be difficult for models to learn underlying patterns within the natural language.

1.2.2 Text Vectorization

Textual data itself cannot be fed as input to neural networks which rely on numerical features. The textual data must be vectorized into numerical data, there are four main methods of vectorization for textual data.

Binary vectorization will create a sparse input matrix with each column corresponding to each word in the entire corpus of all documents. A value of 1 will mean that the word is present in the document and a value of 0 means that the word in which the feature represents is not present.Count vectorization will create a sparse input matrix similar to binary vectorization however the value

is the frequency in which that particular word is found in the document e.g. if a word is found five times in a document, the resulting value in the matrix will be 5.

Tf-idf vectorization is similar to the previous two forms of vectorization in the structure of the vectorized matrix. The defining feature of Tf-idf vectorization is that the values of the matrix contain tf-idf scores. Tf-idf stands for term frequency-inverse document frequency, it is a statistical measure used to evaluate how important a word is a document in a corpus. The importance increases proportionally to the number of times a word appears in a document, however is offset by how many times that particular word appears in the corpus. Tf-idf if calculated by the formula (1), where tf denotes term frequency, N is the total number of documents in the corpus and df denotes document frequency. Term frequency is the number of occurrence of a particular term in a document, document frequency is the number of documents which contain that term.

$$Tf - Idf = tf * \log(\frac{N}{df}) \tag{1}$$

The word2vec text vectorization model differs in structure from the previous methods listed earlier. The columns of the matrix represent the sequential ordering of the words, with the first word of the document corresponding to the first column of the matrix and so on. The values of the matrix are encoded dictionary mappings for words in the corpus. Consider the following example sentences "The cat is good", "I love my cat" and "My cat is vicious". Each sentence is a document and represents a row in the matrix. A corresponding word2vec matrix would be:

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 1 \\ 6 & 1 & 2 & 7 \end{bmatrix}$$

1.3 N-grams

N-grams are very important in the context of natural language processing. An N-gram is best defined as an N-length sequence of words (order dependent). Single words are unigrams, two word sequences are known as bigrams, three word sequences are known as trigrams and so onwards. They are important in natural language processing tasks such as sentiment analysis because the combination of certain word pairings in context are more indicative of meaning than just the frequency counts of the word themselves.

The concept of N-grams can be easily incorporated into word vectorization. For example count vectorization matrices can be calculated with the only different between regular unigram count and greater N-grams is that each column of the matrix corresponds to an N word sequence.

1.4 Multi-layer Perceptron

A multilayer perceptron (MLP) model is a deep artificial neural network. It is composed of more than a single perceptron. There is an input layer to receive signals, an output layer that makes a decision or prediction and an arbitrary number of hidden layers. MLP models are often applied to supervised learning problems. Training involves adjusting weight and bias parameters which link neurons between one layer to the next. Training involves adjusting these parameters in order to minimize error.

In a multilayer perceptron model there are two components of training. The forward pass which involves a feed through the network to make a prediction.Using the backwards propagation algorithm, partial derivatives of the error function with respect to the weights and biases are fed backward through the network. This differentiating gives a gradient along which the parameters may be adjusted to get closer to the error minimum.

2 Methods

2.1 Text Processing

The dataset that was used was a csv file consisting of 30,000 movie reviews that were classified as positive or negative. The first step in the data pre processing was encoding the labels as 0 for a negative review and 1 for a positive review. The reviews were then tokenized by word, removed of all punctuation, numeric digits and lemmatized. Finally a set of english stopwords from nltk were removed from the tokenized collection. Stop-words are common words in the English language like "a", "is", "the" that convey little meaning to the sentiment of a document.

2.2 MLP Implementation

The next step after basic text pre processing was to vectorize the processed text. For the MLP model the text was then vectorized by both sklearn CountVectorizer and TfidfVectorizer on N-gram ranges 1 through 3. These vectorizers were passed 100000 as the max-features argument, meaning they would only transform the data into a sparse matrix of 100000 columns (features), representing the most frequently occurring 100000 N-grams.

Since my computer does not have enough memory to hold the entire resulting sparse matrices for training, a batch generator was used. The batch generator allowed for only using a small number of records at a time for training. The models were compiled with the Adam optimizer and trained against binary-crossentropy loss function, as standard for binary classification problems in MLP models. Various permutations of different hidden layers and neurons were experimented with. The hidden layer neurons use the ReLU activation function. The output layer of the model consists of one output neuron activated by a sigmoid function to output the probability that the instance belongs

to the postive class (positive sentiment). The model was trained for only two epochs (runs through the full training set) to prevent overfitting.

2.3 LSTM Implementation

For the LSTM model, a word2vec vectorization was used with an optimized number of words used, set as the max-features parameter. An issue with the word2vec vectorizer is within a corpus there are varying different lengths of documents, in this case movie reviews. One requirement for neural network models is that all input dimensions must be the same size. To adhere to this property, the vectorized data was zero padded to specific length. Documents which were longer than the length were cut off and those shorter were padded with zeroes.

The first layer in the LSTM model is an embedding layer, which is a dense vector representation of the textual data. This then feeds into a bidirectional LSTM with an optimized number of units. Following the LSTM layer a single fully connected hidden layer with a ReLU activation function, followed by an output layer with one output activated by a sigmoid function.

3 Results

The MLP model with optimized hyper-parameters yielded an accuracy of 88.53% on the test data, with an F1 score of 0.88699. The optimized LSTM model recorded an accuracy of 88.27% on the test data, with an F1 score of 0.88729. The results between the two very different models are extremely similar. The MLP model in design is much more simple and only has 4,000,081 parameters to train compared to the LSTM model which has 14,073,381.

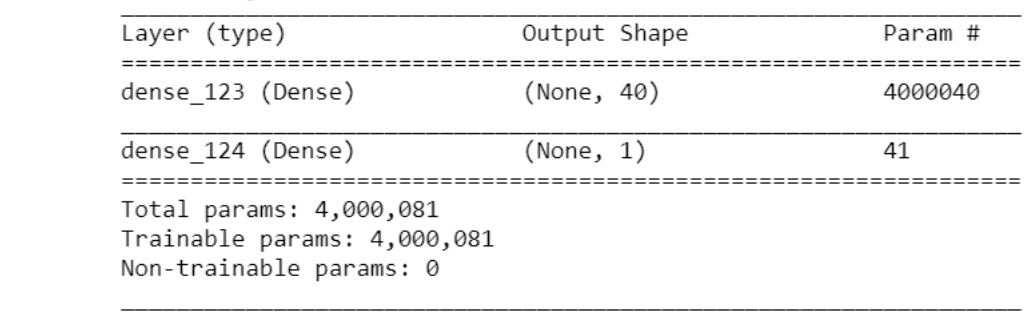

Figure 1: Optimized architecture of MLP

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 120, 150)          13962000
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               110080
_________________________________________________________________
dense (Dense)                (None, 10)                1290
_________________________________________________________________
dropout (Dropout)            (None, 10)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 11
=================================================================
Total params: 14,073,381
Trainable params: 14,073,381
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Optimized architecture of LSTM

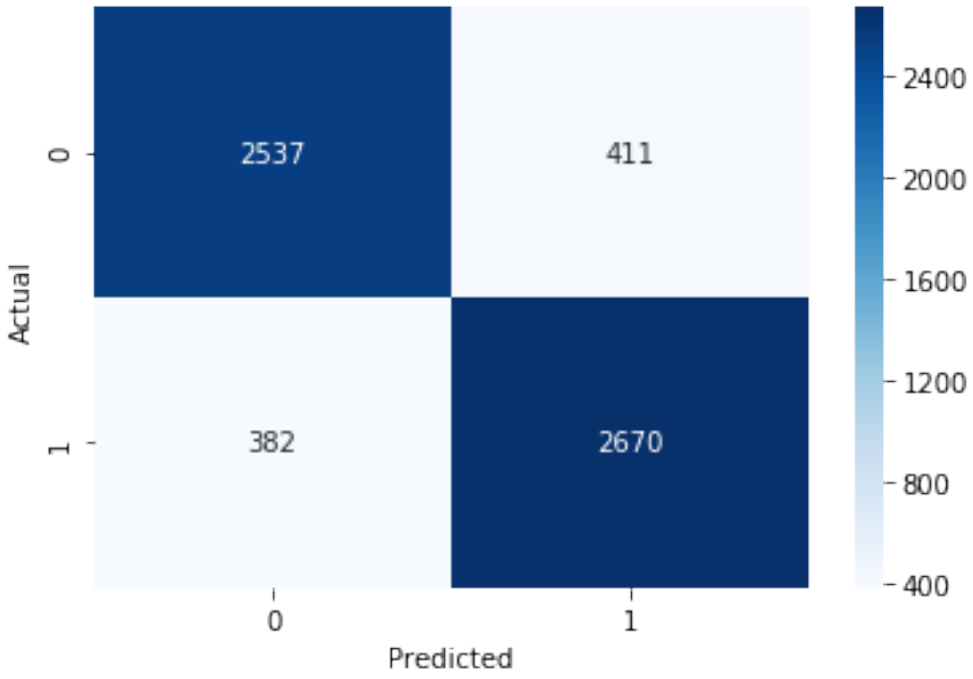

Figure 3: Confusion Matrix for MLP

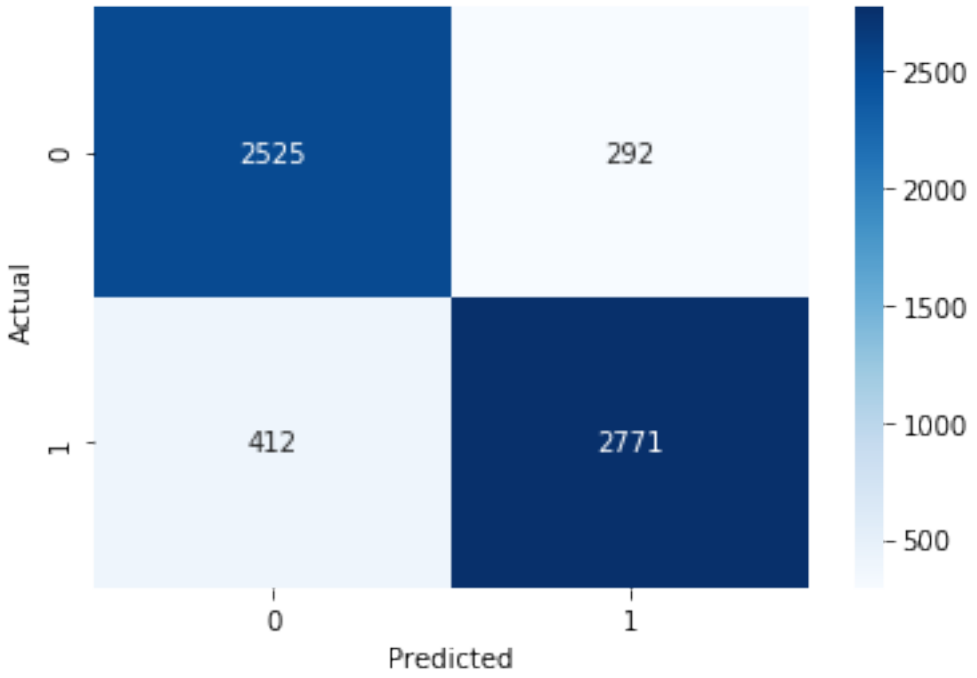

Figure 4: Confusion Matrix for LSTM

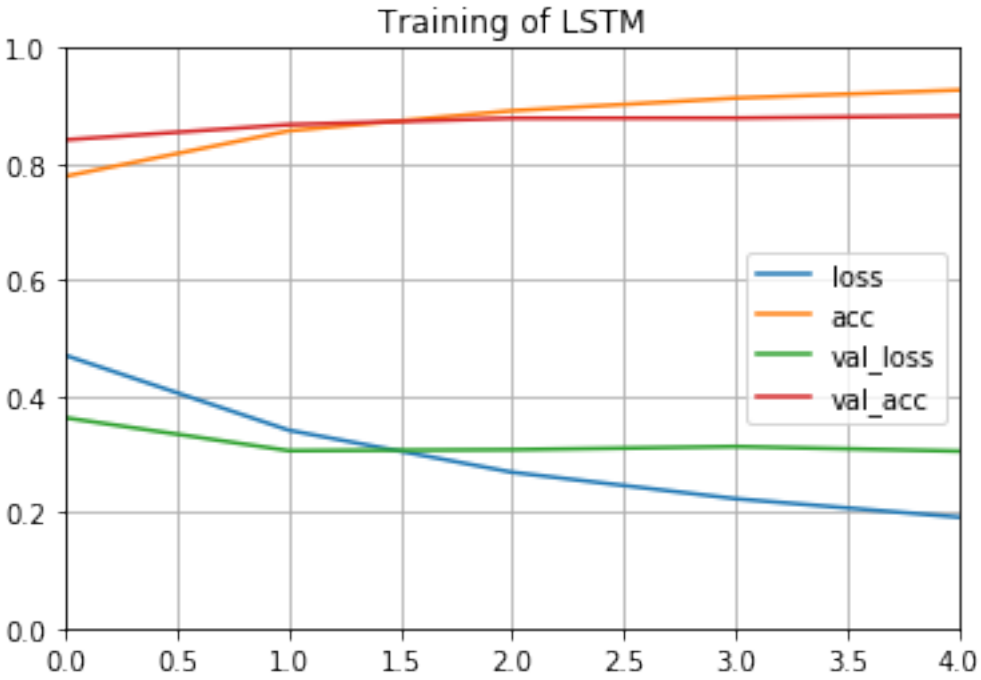

Figure 5: Training of the LSTM per epoch

References

[1] Geron Aurelien. (2019) . *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems.* Sebastopol, CA: OReilly Media, Inc.

[2] Loy, J. (2019). *Neural network projects with Python: the ultimate guide to using Python to explore the true power of neural networks through six projects.* Birmingham: Pakt Publishing.