

Text Classification of Movie Reviews

SRI VISHNU S

16EE146

V SEMESTER

ELECTRICAL AND ELECTRONICS DEPARTMENT

NITK SURATHKAL

**Abstract**

In this project, artificial neural network is used to tackle the problem of sentiment classification of large movie reviews, each consisting of multiple sentences. Artificial neural networks is used to classify the movie reviews as positive and negative based on the sentiment output given by the model. The IMDB movie review dataset is used for training and validating the model. Two models, MLP(multi-layer perceptron) and LSTM(Long Short Term Memory) models are trained and tested to find the better model.

Movie reviews are converted in to a series of integers. A Embedding layer, a class of approaches for representing words and documents using a dense vector representation is used as the first layer for both the models. Keras API from TensorFlow library is used in this project to build, test, validate and predicting model outputs.

**Introduction**

Text categorization refers to the process of assigning a category or some categories among predefined ones to each document, automatically. Text categorization is a pattern classification task for text mining and necessary for efficient management of textual information systems.

The problem of sentiment classification has received a lot of attention of late. It finds applications in the fields of business intelligence, recommender systems (so as to be able to understand the sentiment in a user’s feedback), in online surveys that have natural language sentences as responses, message filtering, assessing movie review polarity, among others.

There have also been a spate of developments in the field of natural language processing using methods from deep neural networks, for various tasks including simpler ones like POS tagging and Named Entity Recognition.

There are two types of approaches to text categorization: rule based and machine learning based approaches. Rule based approaches mean ones where classification rules are defined manually in form of if-then-else, and documents are classified based on the rules. For example, classification rules are defined as, “business and company then company” meaning that if a document includes the two words ‘business’ and ‘company’, it is classified into the category, ‘business’. This class of approaches has high precision but poor recall, because of its poor flexibility. Machine learning based approaches mean ones where classification rules or equations are defined automatically using sample labeled documents. This class of approaches has a much higher recall but a slightly lower precision than rule based approaches. In addition to their poor flexibility, rule based approaches require time consuming manual jobs for building classification rules. Therefore, machine learning based approaches are replacing rule based ones for text categorization.

Like any other pattern classification problem, in text categorization, it is true that documents given as raw data should be encoded into numerical vectors. This strategy of encoding documents leads to two main problems: huge dimensionality and sparse distribution. In spite of using feature selection methods, a reduced dimension of numerical vectors representing documents still remains large. Excessive reduction of the dimension of numerical vectors using a feature selection method degrades the robustness of text categorization. The second problem, sparse distribution, leads to poor discrimination among numerical vectors for categorizing them.

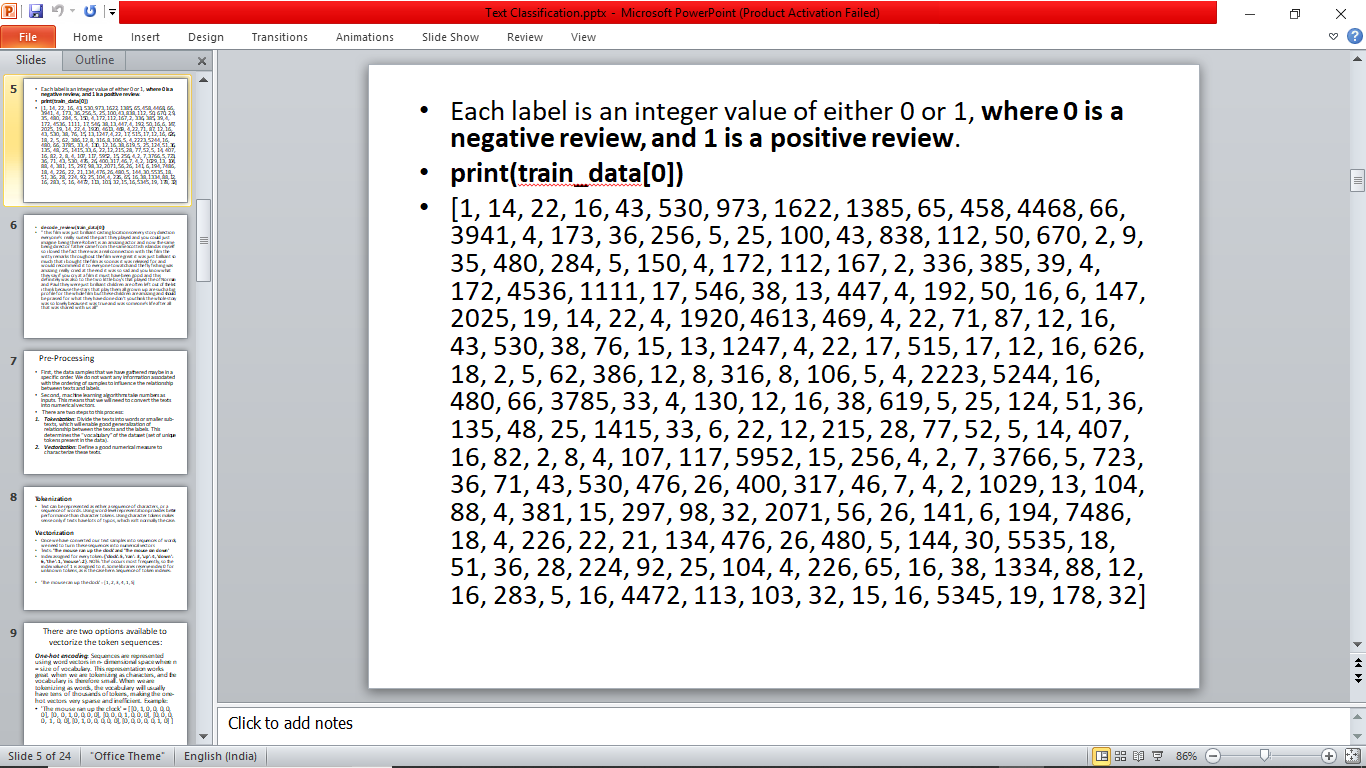
In this project, sentiment classification for paragraphs of text (i.e., with multiple sentences, as is the case typically in movie reviews) is done using Embedded Layers, MLP and LSTM models and based on the sentiment output , reviews are classified as positive or negative.

**Methodology**

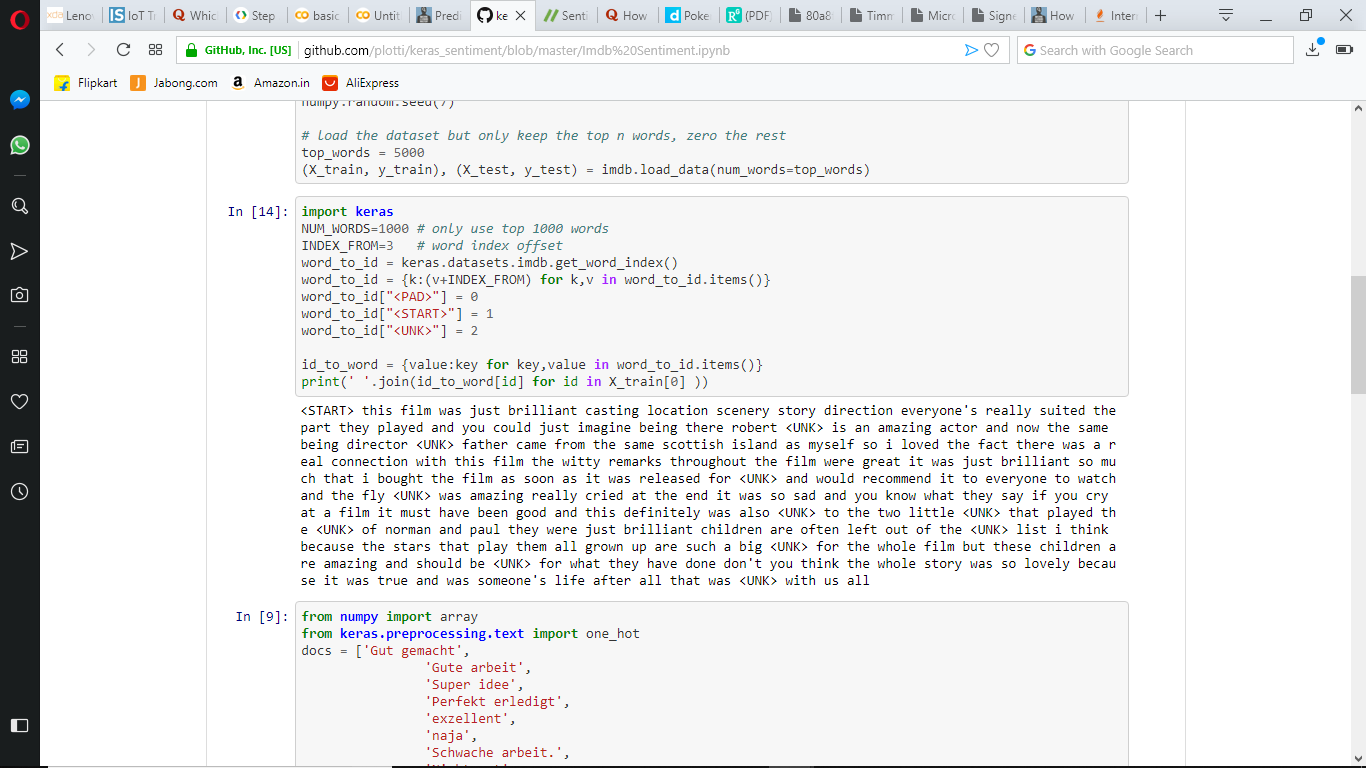
The dataset used is the IMDB movie review dataset which comes packaged with TensorFlow. It has already been preprocessed such that the reviews (sequences of words) have been converted to sequences of integers, where each integer represents a specific word in a dictionary. The whole dataset containing 50000 reviews is split in to 25000 reviews for training and 25000 reviews for testing. Only the first top 10000 words based on the frequency of occurrence are taken for each review. The rare words are discarded to keep the size of the data manageable.

The dataset comes preprocessed: each example is an array of integers representing the words of the movie review. Each label is an integer value of either 0 or 1, where 0 is a negative review, and 1 is a positive review. The text of reviews have been converted to integers, where each integer represents a specific word in a dictionary.

An example is given below:



Every word in the vocabulary is linked to an integer value and the dictionary get\_word\_index() can be used to get integer value for any word. It may be useful to know how to convert integers back to text. Here, a helper function to query a dictionary object that contains the integer to string mapping is created. The example shown above when decoded becomes as shown below.



The reviews—the arrays of integers—must be converted to tensors before fed into the neural network. This conversion can be done in a couple of ways:

* One-hot-encode the arrays to convert them into vectors of 0s and 1s. For example, the sequence [3, 5] would become a 10,000-dimensional vector that is all zeros except for indices 3 and 5, which are ones. Then, make this the first layer in our network—a Dense layer—that can handle floating point vector data. This approach is memory intensive, though, requiring a num\_words \* num\_reviews size matrix.
* Alternatively, we can pad the arrays so they all have the same length, then create an integer tensor of shape max\_length \* num\_reviews. We can use an embedding layer capable of handling this shape as the first layer in our network

In this project the latter approach is used. Since the movie reviews must be the same length, pad\_sequences function is used to standardize the lengths. Embedding Layer is given as the first layer of the model.

Word embeddings provide a dense representation of words and their relative meanings. They are an improvement over sparse representations used in simpler bag of word model representations in which words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space. Word embeddings can be learned from text data and reused among projects. They can also be learned as part of fitting a neural network on text data. one could save the learned weights from the Embedding layer to file for later use in other models.

Only the first 10,000 most used words in the dataset are used. Therefore the vocabulary size will be 10,000. We choose to use a 32-dimension vector to represent each word. Finally, we choose to cap the maximum review length at 500 words, truncating reviews longer than that and padding reviews shorter than that with 0 values.

Building machine learning models with Keras is all about assembling together layers, data-processing building blocks, much like we would assemble Lego bricks. These layers allow us to specify the sequence of transformations we want to perform on our input. As our learning algorithm takes in a single text input and outputs a single classification, we can create a linear stack of layers using the Sequential model API.

**Simple Multi-Layer Perceptron Model**

A simple MLP model with a single hidden layer and 250 hidden units is created. An Embedding layer is used as the input layer, setting the vocabulary to 10,000, the word vector size to 32 dimensions and the input\_length to 500. The output of this first layer will be a 32×500 sized matrix. Then flatten the Embedded layers output to one dimension, then use one dense hidden layer of 250 units with a rectifier activation function. The output layer has one neuron and will use a sigmoid activation to output values of 0 and 1 as predictions. The model uses logarithmic loss and is optimized using the efficient ADAM optimization procedure.We can fit the model and use the test set as validation while training. This model overfits very quickly so we will use very few training epochs, in this case just 2. There is a lot of data so we will use a batch size of 128. After the model is trained, we evaluate its accuracy on the test dataset.

**LSTM Model**

The first layer is an Embedding layer similar to that used in MLP model.

Recurrent neural networks are networks that are used for "things" that happen recurrently so one thing after the other (e.g. time series, but also words). Long Short-Term Memory networks (LSTM) are a specific type of Recurrent Neural Network (RNN) that are capable of learning the relationships between elements in an input sequence. In this case the elements are words. So the next layer is an LSTM layer with 100 memory units. LSTM networks maintain a state, and so overcome the problem of a vanishing gradient problem in recurrent neural networks.

So our output of the embedding layer is a 500 times 32 matrix. Each word is represented through its position in those 32 dimensions. And the sequence is the 500 words that we feed into the LSTM network.Finally at the end we have a dense layer with one node with a sigmoid activation as the output.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 one (adam) and the metrics are also the standard accuracy metric.

**Results**

After a model is fit with the training data, it is evaluated using the evaluate function. Two values will be returned. Loss (a number which represents our error, lower values are better), and accuracy.

The training loss *decreases* with each epoch and the training accuracy *increases* with each epoch. This is expected when using a gradient descent optimization, it should minimize the desired quantity on every iteration.This isn't the case for the validation loss and accuracy, they seem to peak after about twenty epochs. This is an example of overfitting, the model performs better on the training data than it does on data it has never seen before. After this point, the model over-optimizes and learns representations specific to the training data that do not generalize to test data.

To avoid overfitting and underfitting, the models are trained for only enough epochs.

Model-1 which is MLP, is trained for 2 epochs with a batch size of 128.

Model-2 which is LSTM, is trained for 3 epochs with a batch size of 64.

Model-1 took very less time to train and achieved an accuracy of 86.74%.

Model-2 took a much longer time to train and achieved an accuracy of 87.03%.

**Predicting Results**

As said earlier, the input to the model should be a set of integers with each number representing a particular word in the vocabulary. When giving an input review with multiple sentences to the model for prediction, the input must first be pre-processed. First all the letters in the string are converted to lower-case.Then This multi-line input or rather review is converted to series of integers using the dictionary get\_word\_index(). The words which are not in the vocabulary is assigned 0. Now the reviews are series of integers of different lengths which has to be padded to same size as training data and testing data, i.e 500.

The predict method returns the output in the range between 0 and 1, where (<0.5) means negative and (>=0.5) means positive.

Both the models were tested again with 4 types of review

1. Short negative review: “**The movie was a disaster. It was bad**”. Both the models were able to predict it as a negative review.
2. Short positive review: “**I really liked the movie and had fun**”. MLP model failed to predict it correctly.
3. Long positive review: Positive review of the movie “**Avengers Infinity war**” from [gamesradar.com](https://www.gamesradar.com/avengers-infinity-war-review).Both the models were able to predict correctly.
4. Long negative review: Negative review of the movie “**Nine Lives**” from [commonsensemedia.org](https://www.commonsensemedia.org/movie-reviews/nine-lives). Both the models predicted correctly.

**Conclusion**

Artificial Neural networks were used to build a system to classify movie reviews as positive and negative. Both the models that were built were evaluated. Both MLP and LSTM models achieved a testing accuracy nearly 87%, which is a very good score.

Therefore a proper preprocessing of input data based on Natural language processing approaches and a proper network when trained for the appropriate number of epochs gives a appreciable prediction. The LSTM model performed a little better than the MLP model.

**References**

[1]<https://developers.google.com/machine-learning/guides/text-classification/>

[2]<https://machinelearningmastery.com/predict-sentiment-movie-reviews-using-deep-learning/>

[3]<https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/>

[4]<https://www.gamesradar.com/avengers-infinity-war-review/>

[5]<https://www.commonsensemedia.org/movie-reviews/nine-lives>