1. **Text Classification**

   It is highly recommended that you complete this project using Keras[1] and Python.

   (a) In this problem, we are trying to build a classifier to analyze the sentiment of reviews. You are provided with text data in two folders: one folder involves positive reviews, and one folder involves negative reviews.

   (b) **Data Exploration and Pre-processing**

   i. You can use binary encoding for the sentiments , i.e $y = 1$ for positive sentiments and $y = -1$ for negative sentiments.

   ii. The data are pretty clean. Remove the punctuation and numbers from the data.

   iii. The name of each text file starts with `cv_number`. Use text files 0-699 in each class for training and 700-999 for testing.

   iv. Count the number of unique words in the whole dataset (train + test) and print it out.

   v. Calculate the average review length and the standard deviation of review lengths. Report the results.

   vi. Plot the histogram of review lengths.

   vii. To represent each text (= data point), there are many ways. In NLP/Deep Learning terminology, this task is called tokenization. It is common to represent text using popularity/ rank of words in text. The most common word in the text will be represented as 1, the second most common word will be represented as 2, etc. Tokenize each text document using this method.[2]

   viii. Select a review length L that 70% of the reviews have a length below it. If you feel more adventurous, set the threshold to 90%.

   ix. Truncate reviews longer than L words and zero-pad reviews shorter than L so that all texts (= data points) are of length L.[3]

   (c) **Word Embeddings**

   i. One can use tokenized text as inputs to a deep neural network. However, a recent breakthrough in NLP suggests that more sophisticated representations of text yield better results. These sophisticated representations are called *word embeddings.* "Word embedding is a term used for representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning."[4]. Most deep learning modules (including Keras) provide a convenient way to convert positive integer representations of words into a word embedding by an "Embedding layer." The layer accepts arguments that define the mapping of words into embeddings,

[1] `https://keras.io`

[2] Keras has an API called Tokenizer. It can yield bag of words, one-hot encoded features, etc.

[3] Keras has pad_sequences for doing this.

[4] `https://en.wikipedia.org/wiki/Word_embedding`

including the maximum number of expected words also called the vocabulary size (e.g. the largest integer value). The layer also allows you to specify the dimension for each word vector, called the "output dimension." We would like to use a word embedding layer for this project. Assume that we are interested in the top 5,000 words. This means that in each integer sequence that represents each document, we set to zero those integers that represent words that are not among the top 5,000 words in the document.[5] If you feel more adventurous, use all the words that appear in this corpus. Choose the length of the embedding vector for each word to be 32. Hence, each document is represented as a 32×500 matrix.

ii. Flatten the matrix of each document to a vector.

(d) **Multi-Layer Perceptron**

i. Train a MLP with three (dense) hidden layers each of which has 50 ReLUs and one output layer with a single sigmoid neuron. Use a dropout rate of 20% for the first layer and 50% for the other layers. Use ADAM optimizer and binary cross entropy loss (which is equivalent to having a softmax in the output). To avoid overfitting, just set the number of epochs as 2. Use a batch size of 10.

ii. Report the train and test accuracies of this model.

(e) **One-Dimensional Convolutional Neural Network**:

Although CNNs are mainly used for image data, they can also be applied to text data, as text also has adjacency information. Keras supports one-dimensional convolutions and pooling by the Conv1D and MaxPooling1D classes respectively.

i. After the embedding layer, insert a Conv1D layer. This convolutional layer has 32 feature maps , and each of the 32 kernels has size 3, i.e. reads embedded word representations 3 vector elements of the word embedding at a time. The convolutional layer is followed by a 1D max pooling layer with a length and stride of 2 that halves the size of the feature maps from the convolutional layer. The rest of the network is the same as the neural network above.

ii. Report the train and test accuracies of this model.

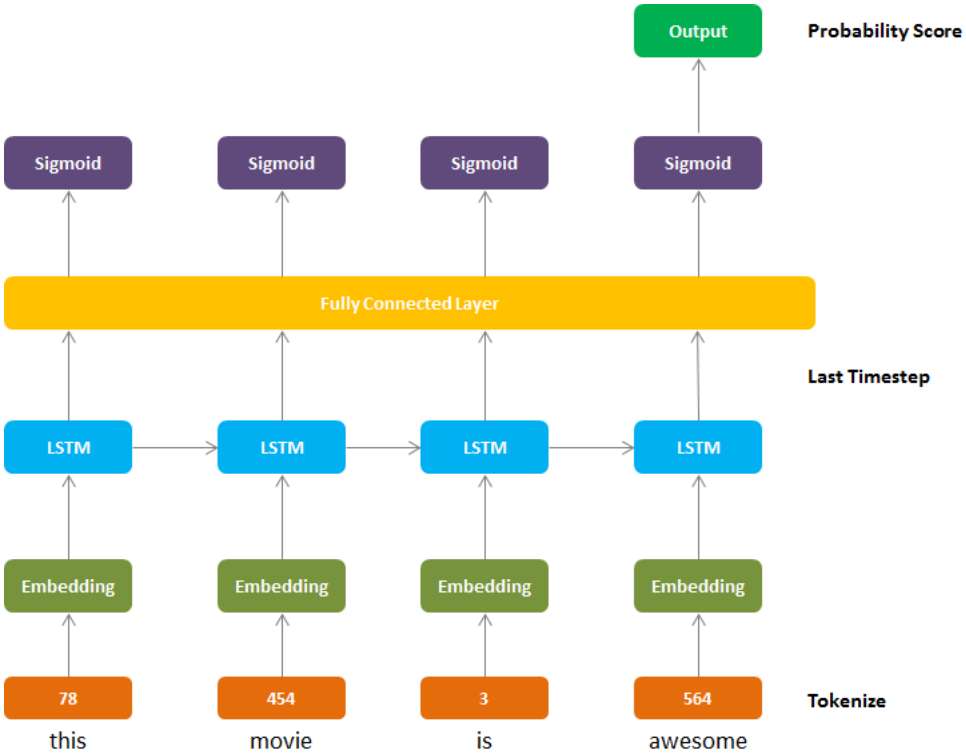
(f) **Long Short-Term Memory Recurrent Neural Network**:

The structure of the LSTM we are going to use is shown in the following figure.

i. Each word is represented to LSTM as a vector of 32 elements and the LSTM is followed by a dense layer of 256 ReLUs. Use a dropout rate of 0.2 for both LSTM and the dense layer. Train the model using 10-50 epochs and batch size of 10.

ii. Report the train and test accuracies of this model.

[5] This is done by setting an argument in the embedding layer provided by Keras. Example: model.add(Embedding(top_words, 32, input_length=max_words)), where top_words=5,000 and max_words=500.


Output
Probability Score
Sigmoid
Sigmoid
Sigmoid
Sigmoid
Fully Connected Layer
Last Timestep
LSTM
LSTM
LSTM
LSTM
Embedding
Embedding
Embedding
Embedding
78
454
3
564
Tokenize
this
movie
is
awesome