NLP on IMDB Reviews

En-Yu Huang
enyuh@uci.edu

William Richard Schallock
wschallo@uci.edu

Hamza Errahmouni Barkam
herrahmo@uci.edu

Abstract

In this project, we will be performing sentiment analysis on the IMDB Movie Reviews dataset. This dataset is composed of text based reviews of popular movies that were originally published on the IMDB website. While there is no ‘one size fits all‘ approach when it comes to text analysis, machine learning can be utilized to process text content in order to classify and understand it.

Specifically, we will utilize machine learning to classify these movie reviews by sorting them into two groups, the ‘positive‘ and the ‘negative‘ reviews of movies. One advantage of using a machine learning approach for this specific task is that it is more efficient than manual classification. For this project, we will present the two machine learning classification methods, compare and contrast their approaches, and discuss the benefits and drawbacks of using each method.

The first method we will discuss is called Random Forest. A Random Forest is a classifier that is built by constructing an ensemble of decision trees, where each tress estimates the classification label of a given piece of text, and then the forest aggregates these individual estimations to form a final classification. Effectively, it fits a number of decision tree classifiers on various sub-samples of the data set. The second method is using a Deep Learning Neural Network model such as a Convolutional Neural Network (CNN) and Long Short Term Memory Network (LSTM). A Deep Learning Neural network model works by training a series of interconnected nodes called neurons. Each neuron is assigned a weight which is adjusted so the model can estimate a classification.

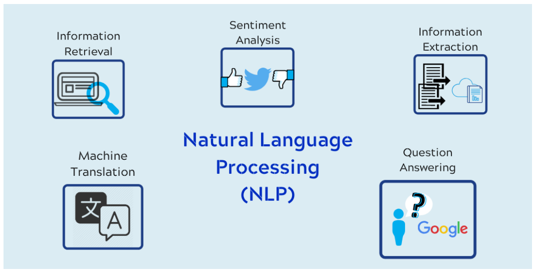
One challenge our group will face is figuring out how to implement a robust system that is able to handle the noise contained in text based reviews. A potential solution is to pre-process and sanitize the text before training. We expect that by comparing the performance of a Random Forest model with a Deep Learning Neural Network model, our group will be able to evaluate the performance of both approaches individually, and by doing so figure out a way to combine them both to create an ensemble model with better performance results.

1 Data Exploration

For our project, we will generate different Machine Leaning models to classify movie reviews. Before we discuss any specific implementation details, we will begin with a brief introduction and answer the question: ""Where does this stand in the big scheme of Machine Learning?"

1.1 NLP

Natural Language Processing, or **NLP** for short, is an emerging branch of Machine Learning that models human communication, in order to better understand the semantics and replicate humans' communicative abilities. Some example applications that utilize NLP techniques are Google Translate and Amazon's voice assistant Alexa. NLP is not straightforward and easy since human language and our ability to learn it are holistic. In addition, language depends on context (what gestures or voice we are putting in addition to our words) so trying to extract sentiments from them are hard (that is why lately we have multimodal assessment on language).



One example of where contextual clues are used in human speech to convey a different underlying meaning than the words by themselves would is irony. We need to understand human language comprehensively in order to correctly interpret the correct message.

1.2 NLP Understanding

NLP has many branches, such as speech generation but our project will focus on NLP understanding which has as main techniques syntactic and semantic analysis. Syntax refers to the arrangement of words and phrases to create well-formed sentences in a language so they make sense. Its application will be defined by the data preprocessing that we will need, and these are some of these techniques:

- Lemmatization, which consists of reducing the different forms of a word to its most basic one (ex.: transform housing -> house)
- Stemming means cutting the inflected words to their root form (ex.: transform went -> go)
- Morphological segmentation implies dividing words into individual units (morphemes).
- Word segmentation involves separating a large text into distinct units (also called tokenization, which we will explain a bit later).
- Part-of-speech tagging means identifying the part of speech for every word (ex.: finding the object of the sentence)

Semantics the branch of linguistics and logic concerned with meaning, it refers to what meaning the text is trying to express (irony, happiness, rage). It is one of the most difficult aspects of NLP as we said previously and it has not been fully solved yet. We will not tackle it in this project.

1.3 Our Data

From the README document included in the dataset, it is described as:

The dataset contains movie reviews along with their associated binary sentiment polarity labels. It is intended to serve as a benchmark for sentiment classification. This document outlines how the dataset was gathered, and how to use the files provided. The core dataset contains 50,000 reviews split evenly into 25k train and 25k test sets. The overall distribution of labels is balanced (25k pos and 25k neg). We also include an additional 50,000 unlabeled documents for unsupervised learning.
In the entire collection, no more than 30 reviews are allowed for any given movie because reviews for the same movie tend to have correlated ratings. Further, the train and test sets contain a disjoint set of movies, so no significant performance is obtained by memorizing movie-unique terms and their associated with observed labels. In the labeled train/test sets, a negative review has a score <= 4 out of 10, and a positive review has a score >= 7 out of 10. Thus reviews with more neutral ratings are not included in the train/test sets. In the unsupervised set, reviews of any rating are included and there are an even number of reviews > 5 and <= 5.

The reviews in this dataset vary in length, from a single sentence up to several pages of text and the main reason behind it is that they are from different IMDB critics. This is the distribution of the training and validation set (80% training, 20% validation):

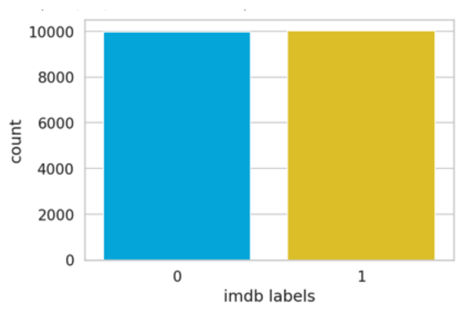


Figure 1: Distribution of positive and negative reviews on the training set

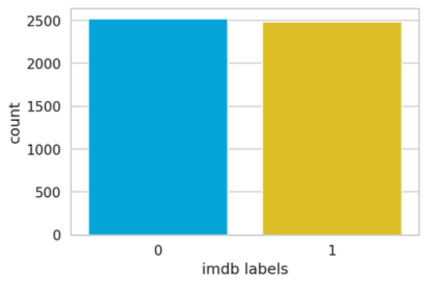


Figure 2: Distribution of positive and negative reviews on the validation set

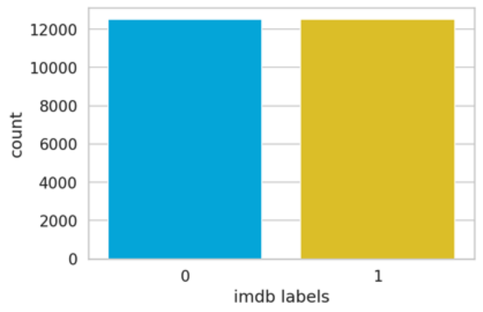


Figure 3: Distribution of positive and negative reviews on the test set

Some difficulties added to this project are the different vocabulary that each one of the reviewers might be using so we have to build a model that is able to find the difference between a positive or negative review (sentiment on the text) with different sizes of text and vocabulary (we need a rich training dataset and avoid overfitting).

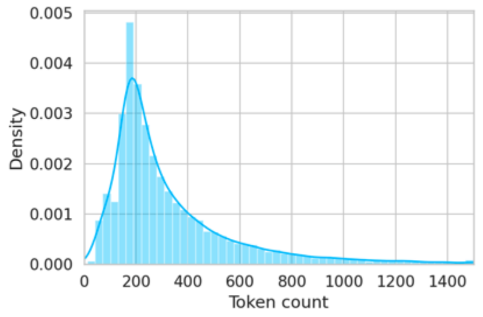


Figure 4: Distribution of the size of the reviews for the training and validation set (in number of tokens)

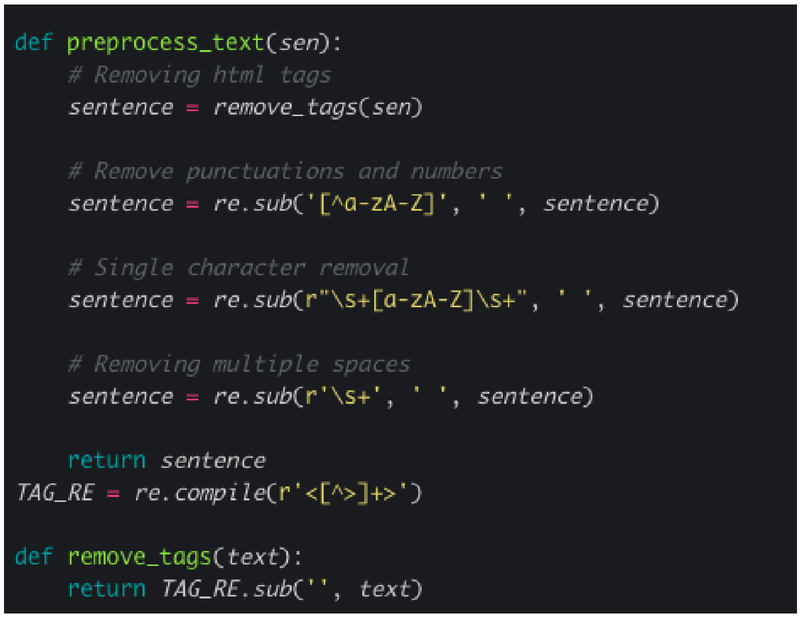
2 Data Preprocessing

In this section, we are going to explain the different preprocessing techniques needed for each one of the algorithms being used.

We saw that our dataset contained punctuation and HTML tags. In this section we will define a function that takes a text string as a parameter and then performs preprocessing on the string to remove special characters and HTML tags from the string, before finally returning the raw text.

One of the example of Raw Data:

> Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the time.<br /><br />This movie is slower than a soap opera... and suddenly, Jake decides to become Rambo and kill the zombie.<br /><br />OK, first of all when you're going to make a film you must Decide if its a thriller or a drama! As a drama the movie is watchable. Parents are divorcing & arguing like in real life. And then we have Jake with his closet which totally ruins all the film! I expected to see a BOOGEYMAN similar movie, and instead i watched a drama with some meaningless thriller spots.<br /><br />3 out of 10 just for the well playing parents & descent dialogues. As for the shots with Jake: just ignore them.



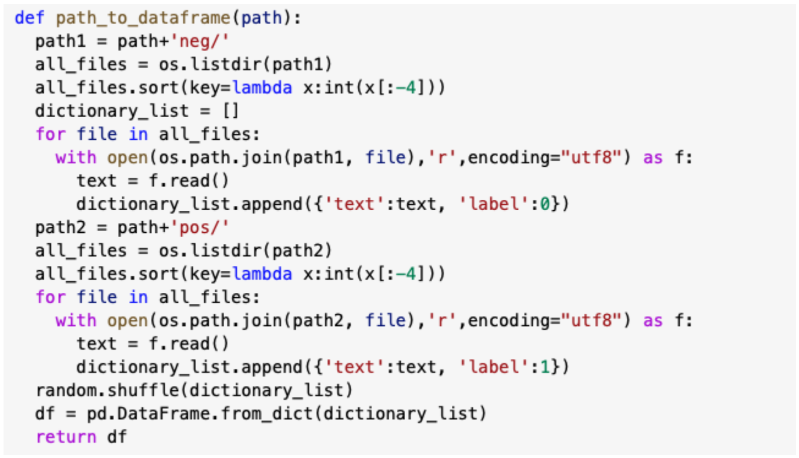
```python
def preprocess_text(sen):
    # Removing html tags
    sentence = remove_tags(sen)

    # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sentence)

    # Single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)

    # Removing multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)

    return sentence
TAG_RE = re.compile(r'<[^>]+>')

def remove_tags(text):
    return TAG_RE.sub('', text)
```

Figure 5: Functions that processes the text removing certain characters and tags

In the preprocess_text() method, the first step is to remove the HTML tags. To remove the HTML tags, remove_tags() function has been defined. The remove_tags() function simply replaces anything between opening and closing <> with an empty space.

Next, in the preprocess_text() function, everything is removed except English letters. This can result in single characters tokens, that make no sense. For instance, when you remove apostrophe from the word "Mark's", the apostrophe is replaced by an empty space. Hence, we are left with single character "s".

Consequently, we remove all the single characters and replace it by a space which creates multiple spaces in our text. Finally, we remove the multiple spaces from our text as well. Next, we will preprocess our reviews and will store them in a new list as shown below:



```python
def path_to_dataframe(path):
  path1 = path+'neg/'
  all_files = os.listdir(path1)
  all_files.sort(key=lambda x:int(x[:-4]))
  dictionary_list = []
  for file in all_files:
    with open(os.path.join(path1, file),'r',encoding="utf8") as f:
      text = f.read()
      dictionary_list.append({'text':text, 'label':0})
  path2 = path+'pos/'
  all_files = os.listdir(path2)
  all_files.sort(key=lambda x:int(x[:-4]))
  for file in all_files:
    with open(os.path.join(path2, file),'r',encoding="utf8") as f:
      text = f.read()
      dictionary_list.append({'text':text, 'label':1})
  random.shuffle(dictionary_list)
  df = pd.DataFrame.from_dict(dictionary_list)
  return df
```

Figure 6: Function that transforms all the positive reviews and negative reviews to a Data Frame Object

Example of Raw Data after preprocessing:

```
Basically there a family where little boy Jake thinks there a zombie in his closet his parents are fighting all the time This movie is slower than soap opera and suddenly Jake decides to become Rambo and kill the zombie OK first of all when you re going to make film you must Decide if its thriller or drama As drama the movie is watchable Parents are divorcing arguing like in real life And then we have Jake with his closet which totally ruins all the film expected to see BOOGEYMAN similar movie and instead watched drama with some meaningless thriller spots out of just for the well playing parents descent dialogues As for the shots with Jake just ignore them.
```

From the output, you can see that the HTML tags, punctuation, and numbers have all been removed. We are only left with the alphabetic characters.

Tokenization:
In order to input our text to the any NLP model, we need not only to preprocess the data as specified previously but we also need to transform it so our machines understand them. Tokenization is the first step of separating the sentences or paragraphs into the smallest units called tokens (either simplified words through lemmatization, subwords like prefixes or sufixes and characters). There are many techniques to have tokenization but we won't go through them since this would take a lot from our report.

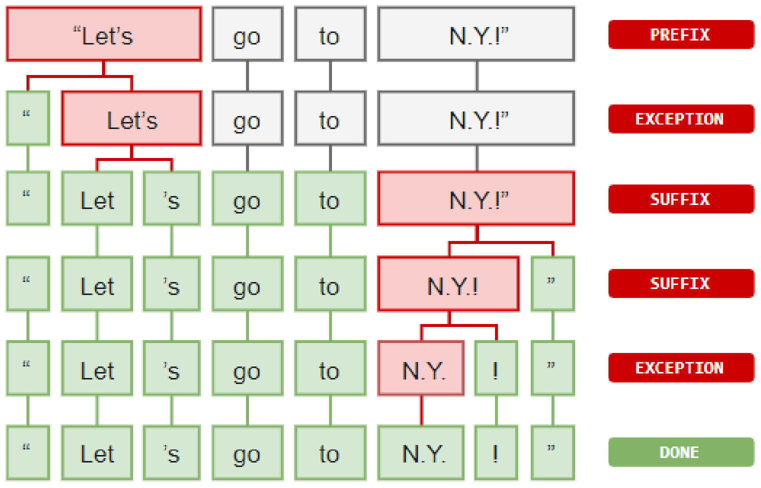


Figure 7: Example of tokenization steps

Tokenization is a fundamental step and then we need to implement the Count Vectorizer which uses the tokens to then be encoded as integers, or floating-point values, for use as inputs in machine learning algorithms.

The CountVectorizer module is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.

3 Model Exploration

As we described in the abstract, our goal is to compare the Random Forests algorithm with two different Neural Networks architectures (CNNs and LTSM). In this section we are going to describe briefly the theory behind the three algorithms so our application of them makes more sense.

3.1 Random Forests

Random Forests has been gaining lots of traction the last years for classification or regression on the Machine Learning world. It consists of an ensemble of decision trees (supervised learning) and each one of the trees has a subset of features and they are trained using what has been called the bagging method. Bagging has been confused with breaking the training set on pieces and using each piece for a different tree but it does not do that. Each tree has as dataset the size of the initial training set, the only difference is that each point is chosen randomly from the training set (many data points can be repeated).

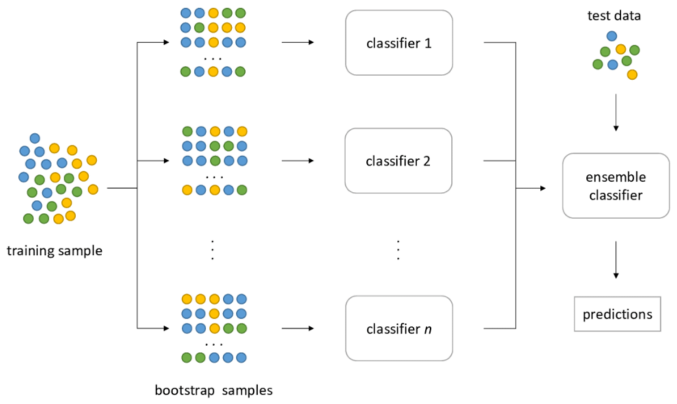


Figure 8: Example of the bagging method

Just by what we present, we assume that our results will show that this algorithm should be more stable than using individual decision trees for example, because by having many decision trees, local individual errors (probably made because that tree didn't have a specific feature that is key) are taken care of. Finally, on this model generated, the subset of features for each tree is generated randomly so all the combinations give robustness to the algorithm results.

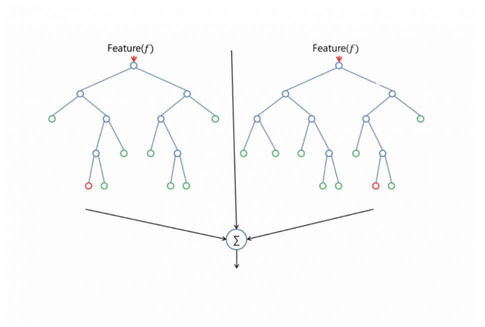


Figure 9: How RF works

This is an example of what the decision would be from an ensemble of random forest:

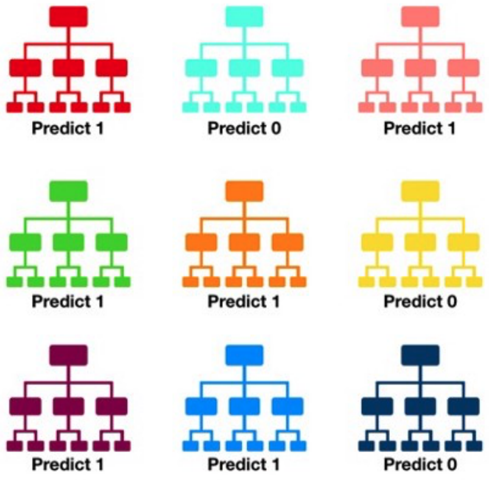
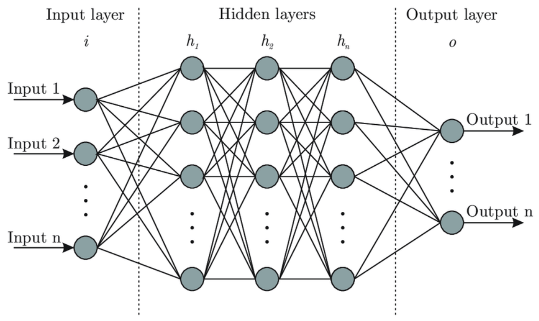


Figure 10: Example of RF computation, we have 6 ones and 3 zeros, so we output 1

3.2 Neural Networks

A neural network is a series of tiny algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.

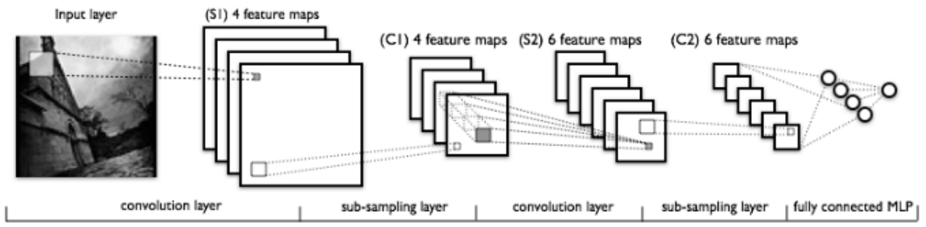


3.2.1 Convolutional Neural Networks

ConvNets were initially developed in the neural network image processing community where they achieved break-through results in recognising an object from a pre-defined category (e.g., cat, bicycle, etc.).

A Convolutional Neural Network typically involves two operations, which can be though of as feature extractors: convolution and pooling.

The output of this sequence of operations is then typically connected to a fully connected layer which is in principle the same as the traditional multi-layer perceptron neural network (MLP).



The Convolutional Neural Network architecture applied to image classification.
(Image adapted from http://deeplearning.net/)

In the case of NLP tasks, i.e., when applied to text instead of images, we have a 1 dimensional array representing the text. Here the architecture of the ConvNets is changed to 1D convolutional-and-pooling operations.

One of the most typically tasks in NLP where ConvNet are used is sentence classification, that is, classifying a sentence into a set of pre-determined categories by considering n-grams, i.e. it's words or sequence of words, or also characters or sequence of characters.

1-D Convolutions over text

Given a sequence of words $w_{1:n} = w_1, \ldots, w_n$, where each is associated with an embedding vector of dimension d. A 1D convolution of width-k is the result of moving a sliding-window of size k over the sentence, and applying the same convolution filter or kernel to each window in the sequence, i.e., a dot-product between the concatenation of the embedding vectors in a given window and a weight vector u, which is then often followed by a non-linear activation function g.

Considering a window of words $w_i, \ldots, w_{i+k}$ the concatenated vector of the ith window is then:

$$x_i = [w_i, w_{i+1}, \ldots, w_{i+k}] \in R^{kd}$$

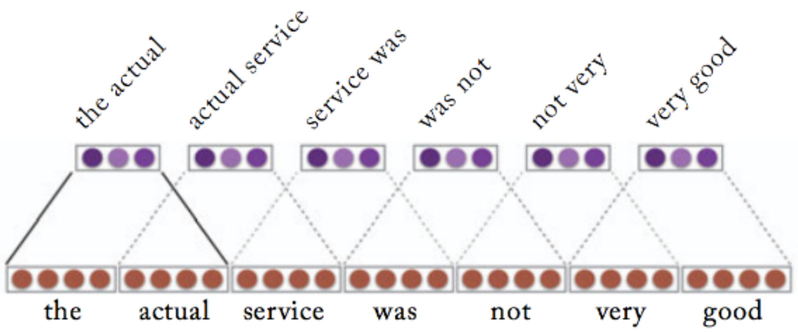
The convolution filter is applied to each window, resulting in scalar values r_i, each for the ith window:

$$r_i = g(xi \cdot u) \in R$$

In practice one typically applies more filters, $u_1, \ldots, u_l$, which can then be represented as a vector multiplied by a matrix U and with an addition of a bias term b:

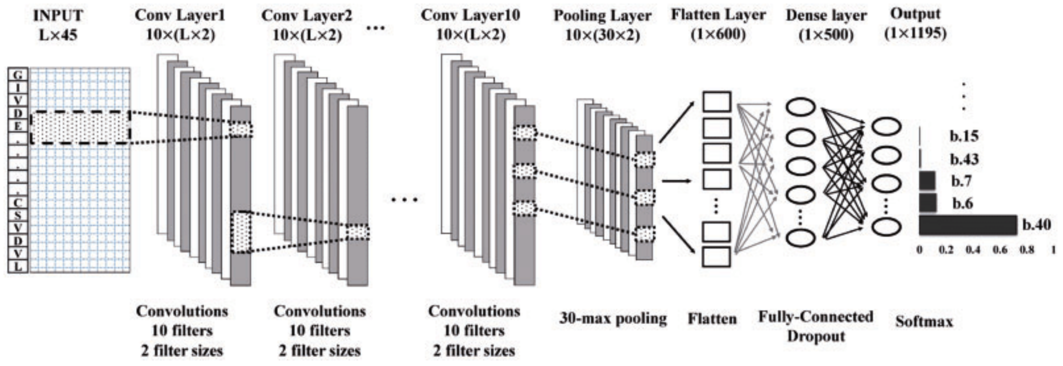
$$r_i = g(xi \cdot U + b)$$
with $r_i \in R^l, x_i \in R^{kd}, U \in R^{k \cdot d \times l}$ and $b \in R^l$

An example of a sentence convolution in a vector-concatenation notation:



Example of a sentence convolution with k=2 and dimensional output l=3.
(Image adapted from Yoav Goldberg book "Neural Network Methods for NLP")

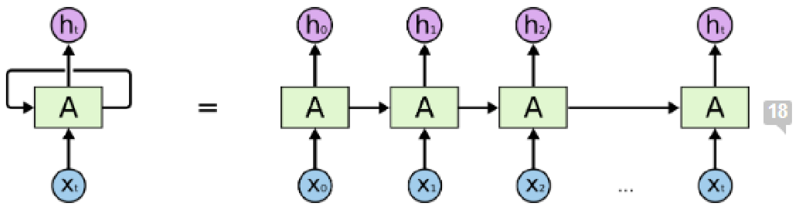
The architecture of 1D deep convolutional neural network for fold classification. The network accepts the features of proteins of variable sequence length (L) as input, which are transformed into hidden features by 10 hidden layers of convolutions. Each convolution layer applies 10 filters to the windows of previous layers to generate L hidden features.
Two window sizes (6 and 10) are used. The 30 maximum values of hidden values of each filter of the 10^{th} convolution layer are selected by max pooling, which are joined together into one vector by flattening. The hidden features in this vector are fully connected to a hidden layer of 500 nodes, which are fully connected to 1195 output nodes to predict the probability of each of 1195 folds. The output node uses softmax function as activation function, whereas all the nodes in the other layers use rectified linear function (ReLU) as activation function. The features in the convolution layers are normalized by batches.



3.2.2 Recurrent Neural Network (LSTM)

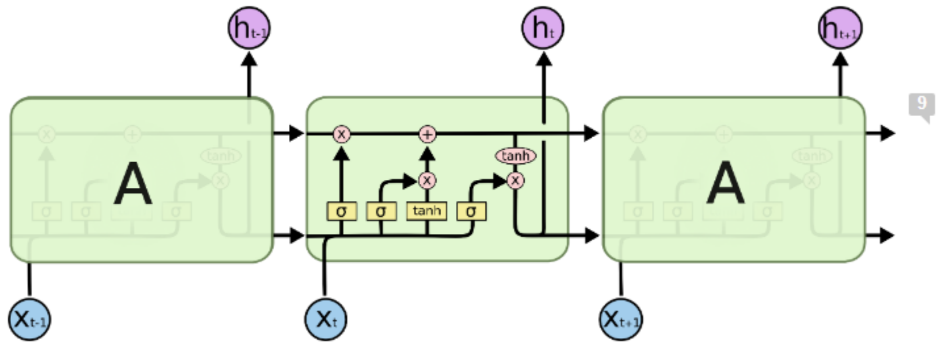
Recurrent neural network is a type of neural networks that is proven to work well with sequence data. Since text is actually a sequence of words, a recurrent neural network is an automatic choice to solve text-related problems.As the name suggests, LSTM networks have both long and short term memory. They're specifically designed to make long term memory possible, and they're able to do this because each LSTM cell is made up of several "gates".

In every single step (except the first), the output from the hidden layer (A) is being fed into itself along with the next time step in the input sequence (x). This means that at each timestep, the network is taking into account the previous parts of the sequence (or more precisely, the way it interpreted those parts) when producing the next output.



An unrolled recurrent neural network.

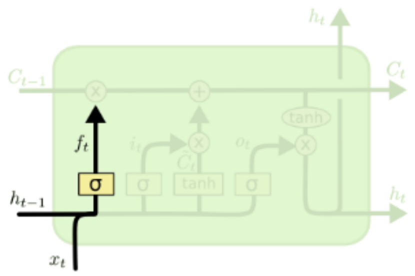
Long Short Term Memory (LSTM) Networks:
As the name suggests, LSTM networks have both long and short term memory. They're specifically designed to make long term memory possible, and they're able to do this because each LSTM cell is made up of several "gates".



The repeating module in an LSTM contains four interacting layers.

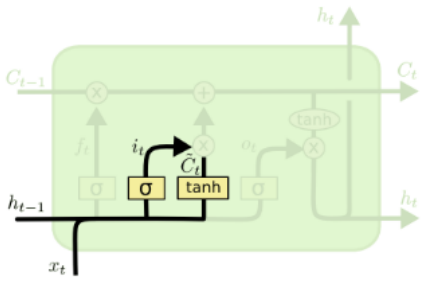
Below are three connected LSTM cells. While they might look complicated at first, once we break it down it's actually pretty simple.

The Forget Gate:
Since not every single part of a sentence is important for understanding it, the network doesn't need to remember every single part. This is why an LSTM has a forget gate, which determines which previous information is useful, and which information is removed from the cell.



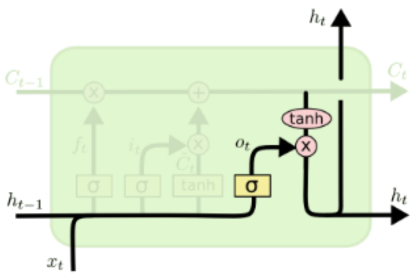
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

The Input Gate(s):
It first determines which new information to keep and which to forget. The tanh layer (which scales the output between -1 and 1) creates new "candidate" (potential) values for the current cell state, and then the output from these two gates is multiplied to determine which candidate values will be kept, and which will be forgotten.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

The Output Gate:
The final gate decides what will actually be outputted. Like the last two gates, it uses a sigmoid layer to determine which information to keep and multiplies it by the current cell-state (put through a tanh function first).



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

In the script below, we start by initializing a sequential model followed by the creation of the embedding layer. Next, we create an LSTM layer with 128 neurons. The rest of the code is same as it was for the CNN.

In [17]:
```
# RNN
model = Sequential()
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlen , trainable=False)
model.add(embedding_layer)
model.add(LSTM(128))

model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

In [18]:
```
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 100, 100)          6797900   
_________________________________________________________________
lstm (LSTM)                  (None, 128)               117248    
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129       
=================================================================
Total params: 6,915,277
Trainable params: 117,377
Non-trainable params: 6,797,900
_________________________________________________________________
None
```

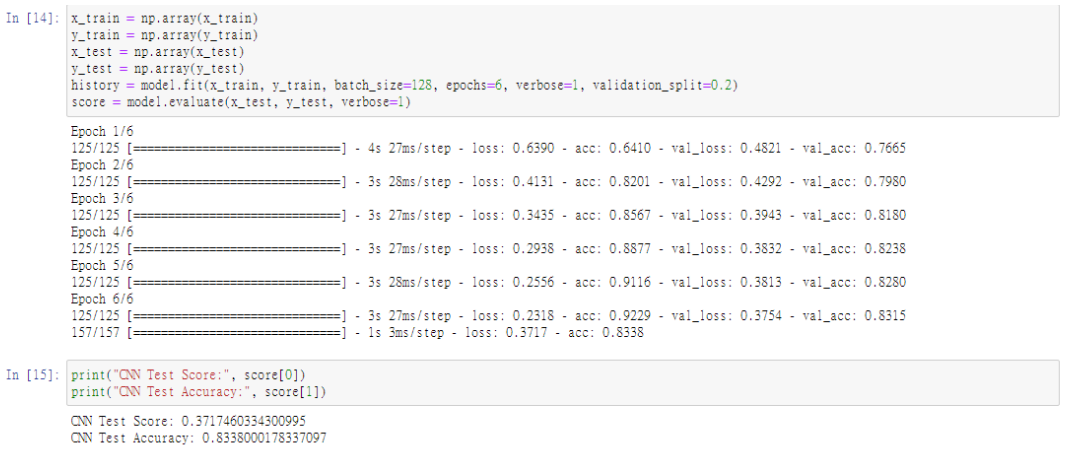
In the above script we create a sequential model, followed by an embedding layer. This step is similar to what we had done earlier. Next, we create a one-dimensional convolutional layer with 128 features, or kernels. The kernel size is 5 and the activation function used is sigmoid. Next, we add a global max pooling layer to reduce feature size. Finally we add a dense layer with sigmoid activation. The compilation process is the same as it was in the previous section.

Our next step is to train the model on the training set and evaluate its performance on the test set.

4 Performance Results

4.1 Results for Convolution Neural Network & Recurrent Neural Network(LSTM)

Results of Convolution Neural Network



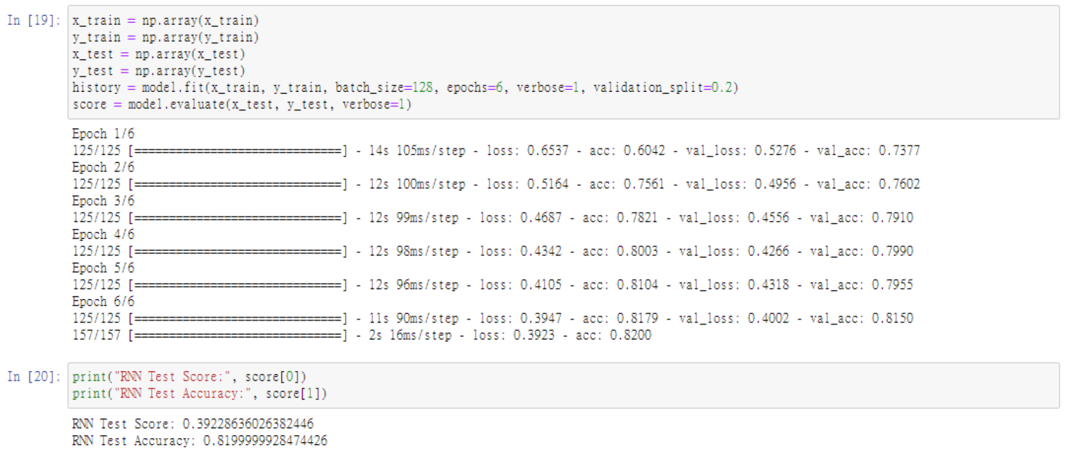
```
In [14]: x_train = np.array(x_train)
         y_train = np.array(y_train)
         x_test = np.array(x_test)
         y_test = np.array(y_test)
         history = model.fit(x_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)
         score = model.evaluate(x_test, y_test, verbose=1)

         Epoch 1/6
         125/125 [==============================] - 4s 27ms/step - loss: 0.6390 - acc: 0.6410 - val_loss: 0.4821 - val_acc: 0.7665
         Epoch 2/6
         125/125 [==============================] - 3s 28ms/step - loss: 0.4131 - acc: 0.8201 - val_loss: 0.4292 - val_acc: 0.7980
         Epoch 3/6
         125/125 [==============================] - 3s 27ms/step - loss: 0.3435 - acc: 0.8567 - val_loss: 0.3943 - val_acc: 0.8180
         Epoch 4/6
         125/125 [==============================] - 3s 27ms/step - loss: 0.2938 - acc: 0.8877 - val_loss: 0.3832 - val_acc: 0.8238
         Epoch 5/6
         125/125 [==============================] - 3s 28ms/step - loss: 0.2556 - acc: 0.9116 - val_loss: 0.3813 - val_acc: 0.8280
         Epoch 6/6
         125/125 [==============================] - 3s 27ms/step - loss: 0.2318 - acc: 0.9229 - val_loss: 0.3754 - val_acc: 0.8315
         157/157 [==============================] - 1s 3ms/step - loss: 0.3717 - acc: 0.8338

In [15]: print("CNN Test Score:", score[0])
         print("CNN Test Accuracy:", score[1])

         CNN Test Score: 0.3717460334300995
         CNN Test Accuracy: 0.8338000178337097
```

If you compare the training and test accuracy, you will see that the training accuracy for CNN will be around 92%, which is greater than the training accuracy of the simple neural network. The test accuracy is around 82% for the CNN, which is also greater than the test accuracy for the simple neural network, which was around 74%.

However our CNN model is still overfitting as there is a vast difference between the training and test accuracy. Let's plot the loss and accuracy difference between the training and test set.

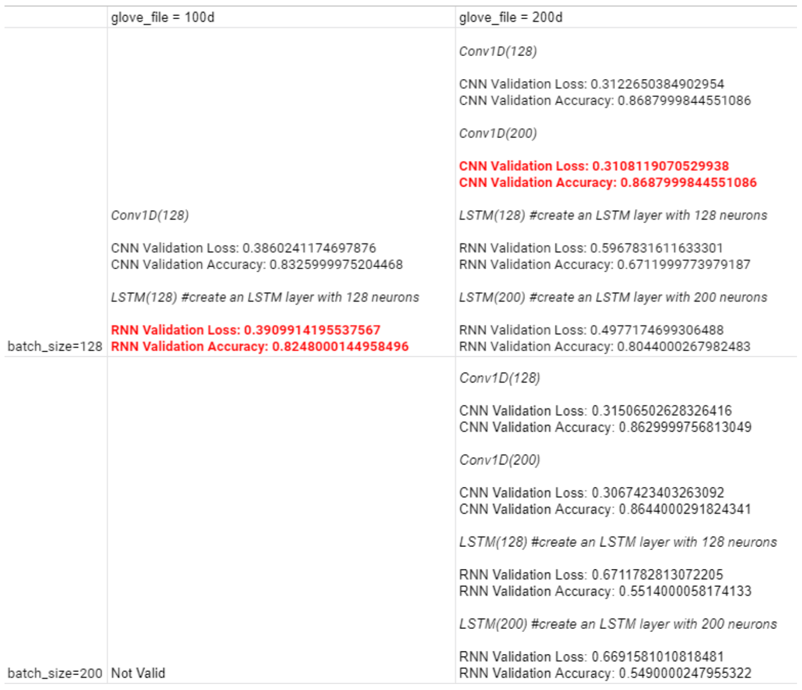
Results of Recurrent Neural Network(LSTM)

The script below trains the model on the test set. The batch size is 128, whereas the number of epochs is 6. At the end of the training, you will see that the training accuracy is around 85.40%.



```
In [19]: x_train = np.array(x_train)
         y_train = np.array(y_train)
         x_test = np.array(x_test)
         y_test = np.array(y_test)
         history = model.fit(x_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)
         score = model.evaluate(x_test, y_test, verbose=1)

         Epoch 1/6
         125/125 [==============================] - 14s 105ms/step - loss: 0.6537 - acc: 0.6042 - val_loss: 0.5276 - val_acc: 0.7377
         Epoch 2/6
         125/125 [==============================] - 12s 100ms/step - loss: 0.5164 - acc: 0.7561 - val_loss: 0.4956 - val_acc: 0.7602
         Epoch 3/6
         125/125 [==============================] - 12s 99ms/step - loss: 0.4687 - acc: 0.7821 - val_loss: 0.4556 - val_acc: 0.7910
         Epoch 4/6
         125/125 [==============================] - 12s 98ms/step - loss: 0.4342 - acc: 0.8003 - val_loss: 0.4266 - val_acc: 0.7990
         Epoch 5/6
         125/125 [==============================] - 12s 96ms/step - loss: 0.4105 - acc: 0.8104 - val_loss: 0.4318 - val_acc: 0.7955
         Epoch 6/6
         125/125 [==============================] - 11s 90ms/step - loss: 0.3947 - acc: 0.8179 - val_loss: 0.4002 - val_acc: 0.8150
         157/157 [==============================] - 2s 16ms/step - loss: 0.3923 - acc: 0.8200

In [20]: print("RNN Test Score:", score[0])
         print("RNN Test Accuracy:", score[1])

         RNN Test Score: 0.39228636026382446
         RNN Test Accuracy: 0.8199999928474426
```

Plots of CNN & RNN



	glove_file = 100d	glove_file = 200d
batch_size=128	*Conv1D(128)* CNN Validation Loss: 0.3860241174697876 CNN Validation Accuracy: 0.8325999975204468 *LSTM(128) #create an LSTM layer with 128 neurons* **RNN Validation Loss: 0.3909914195537567** **RNN Validation Accuracy: 0.8248000144958496**	*Conv1D(128)* CNN Validation Loss: 0.3122650384902954 CNN Validation Accuracy: 0.8687999844551086 *Conv1D(200)* **CNN Validation Loss: 0.3108119070529938** **CNN Validation Accuracy: 0.8687999844551086** *LSTM(128) #create an LSTM layer with 128 neurons* RNN Validation Loss: 0.5967831611633301 RNN Validation Accuracy: 0.6711999773979187 *LSTM(200) #create an LSTM layer with 200 neurons* RNN Validation Loss: 0.4977174699306488 RNN Validation Accuracy: 0.8044000267982483
batch_size=200	Not Valid	*Conv1D(128)* CNN Validation Loss: 0.31506502628326416 CNN Validation Accuracy: 0.8629999756813049 *Conv1D(200)* CNN Validation Loss: 0.3067423403263092 CNN Validation Accuracy: 0.8644000291824341 *LSTM(128) #create an LSTM layer with 128 neurons* RNN Validation Loss: 0.6711782813072205 RNN Validation Accuracy: 0.5514000058174133 *LSTM(200) #create an LSTM layer with 200 neurons* RNN Validation Loss: 0.6691581010818481 RNN Validation Accuracy: 0.5490000247955322

Figure 11: Modification of convolution matrix, LSTM layers for both CNN & RNN with different pre-trained data and batch_size

Figure 12 shows that the best performance for each neural network approach, glove_file is an open source pre-trained text data set with 100 and 200 1D matrix. batch_size is used to define the batch_size of model, the size is restricted by the 1D matrix of glove_file. Conv1D is 1D convolutional neural networks to extract features from our data, LSTM is layer with number of neurons.

From result above, CNN performs the best with glove_file.200d and Conv1D(200), because CNN is able to extract more features comparing to glove_file.100d, so the accuracy will increase. RNN performs the best with glove_file.100d and LSTM(128), due to the fact that RNN is hard to modify with the characteristic of processing time series data, even it can achieve better accuracy theoretically, however it is difficult to choose the right parameters for model, also it is hard to avoid overfitting when neurons number goes higher.

Plots of CNN

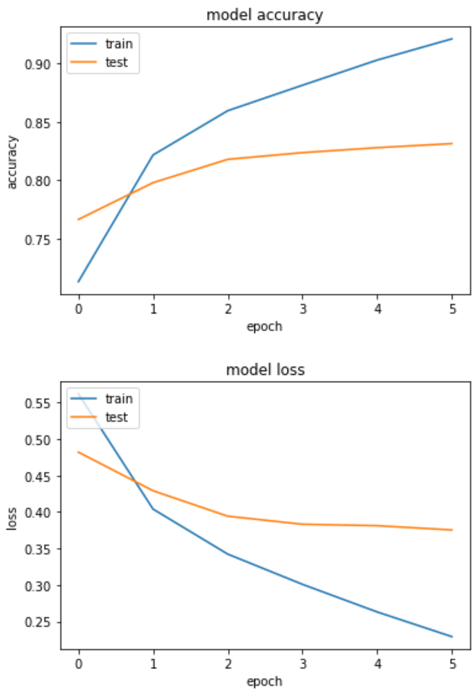


Figure 12: Result of CNN with glove_file.200d, Conv1D(200), batch_size = 200, epoch = 6

Plots of RNN(LSTM)

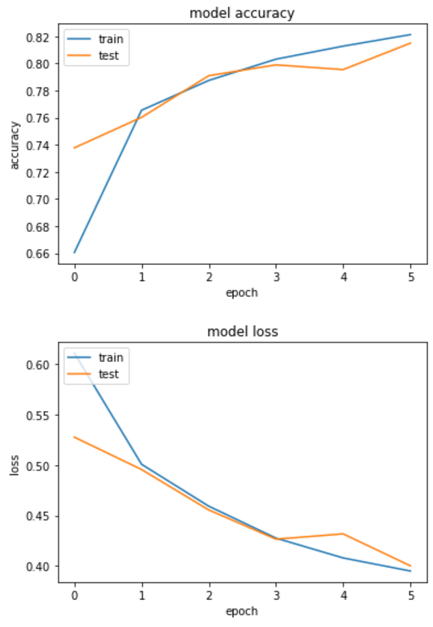


Figure 13: Result of RNN with glove_file.100d, LSTM(128), batch_size = 128, epoch = 6

4.2 Results for RF

For Random Forest, we do not have validation and test accuracy through epochs so we are not presenting results here (we will present the test results on the next section). The only thing we want to comment is that

5 Adaptability

Due to the difficulties associated with training an NLP model, it can be challenging to create a model that is adaptable, in the sense that can generalize it predictions to correctly classify data that differs from its training data. While this issue of adaptability extends past the bounds of NLP tasks and is prevalent in other common Machine Learning model, NLP models are often plagued by issues associated with essentially ’memorizing the training data’ and while can score well on training data, it will score poorly on new testing data classification. In this sections we will discuss the adaptability of our models.

5.1 Complexity / Error rate for Neural Networks

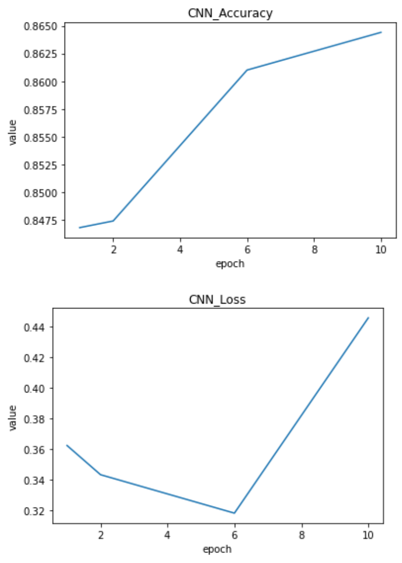
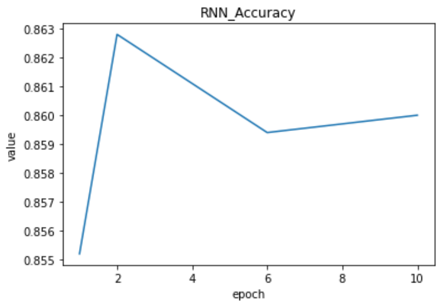


Figure 14: Modification of epoch numbers of CNN

From Figure 15, we can tell that the CNN model Loss achieve to the lowest value when epoch = 6, meanwhile the accuracy is good enough to be our ideal model, hence, we choose epoch = 6 to be our best approach.



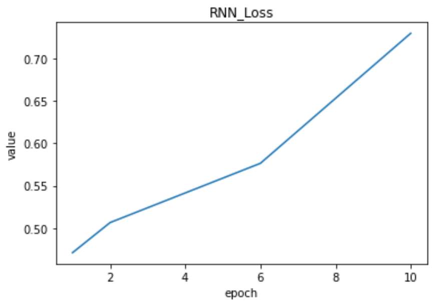


Figure 15: Modification of epoch numbers of RNN(LSTM)

From Figure 16, the loss of RNN model achieve to the lowest value when epoch = 2, yet, the model is not trained enough at the moment. To choose more ideal model, epoch = 6 will be an ideal option, meanwhile the model loss is still within the range we can afford, hence, we choose epoch = 6 to be our best approach.

5.2 Complexity / Error rate for RF

In order to create a model that is robust, we varied model complexity and evaluated the model's performance as a function of model complexity. For the Random Forest model, we played with the number of estimators (decision trees) and depth of the trees. The good thing is that the depth of the trees can be left as null and let them grow as much as they can and that is the one that we will use for the confusion matrix with the best result for number of trees.

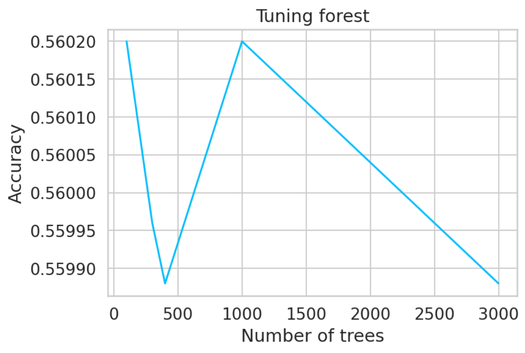


Figure 16: Modification of parameter number of trees of RF

As we can observe here, the performance of RF depends on the number of decision trees (adds robustness) but it fattens out, having more threes becomes not useful when the number is higher than

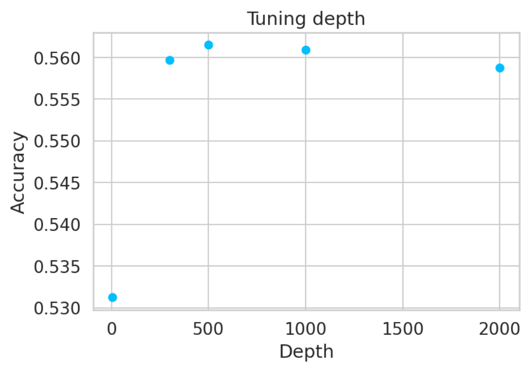
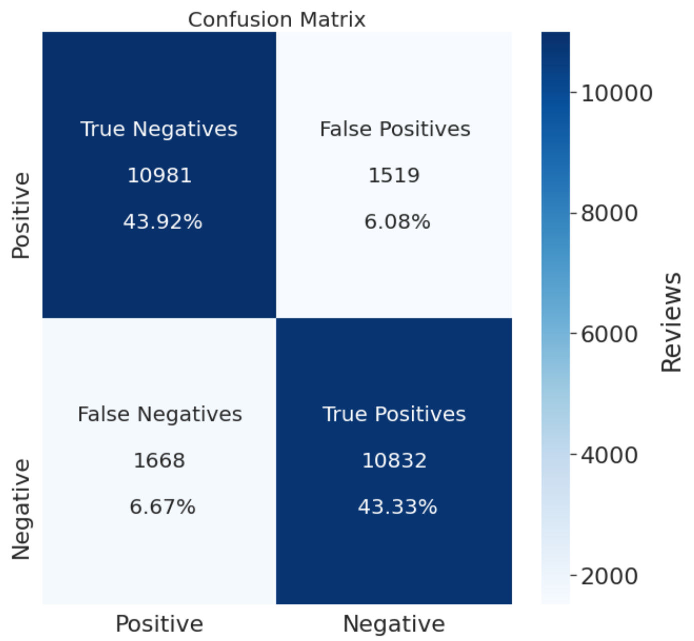


Figure 17: Modification of parameter max depth of trees of RF

As for the depth of the trees, the accuracy seems to

5.3 Confusion Matrix for test set

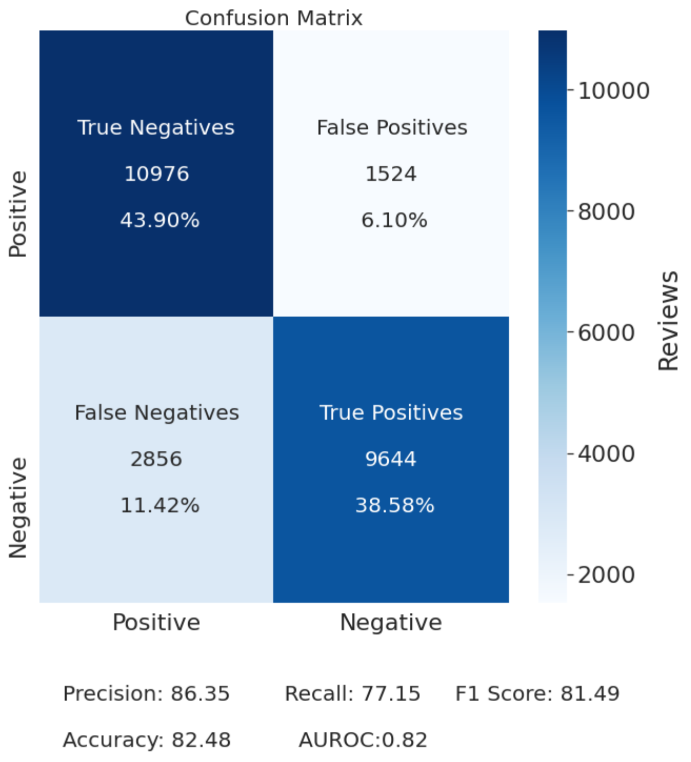


Precision: 87.7 Recall: 86.66 F1 Score: 87.18

Accuracy: 87.25 AUROC:0.87

Figure 18: Confusion Matrix for CNN

Figure 18 shows that the confusion matrix of Convolution Neural Network, it performs better in True Positives section comparing to RNN and achieve the similar accuracy to RNN in True Negatives.



Precision: 86.35 Recall: 77.15 F1 Score: 81.49

Accuracy: 82.48 AUROC:0.82

Figure 19: Confusion Matrix for RNN LSTM

Figure 19 shows that the confusion matrix of Recurrent Neural Network, in most of the case, it can not achieve the performance as good as CNN since the model structure of RNN. It is easy to cause overfitting if we don't modify the parameters well, and that is the reason what causing RNN accuracy to drop. However, RNN might perform better when processing a time series data which is the ability that CNN do not posses.

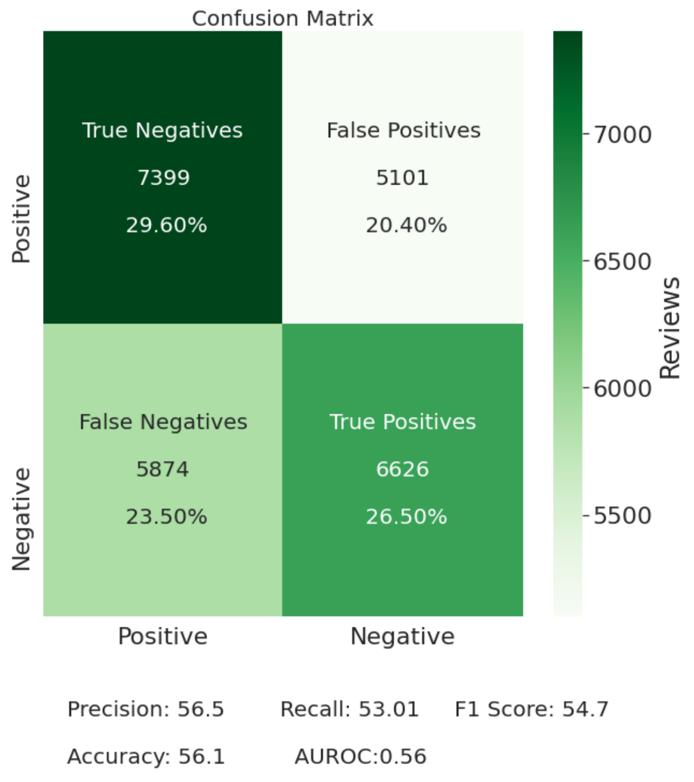


Figure 20: Confusion Matrix for Random Forest

5.4 BERT Transformer

We wanted to do a deep dive into NLP, so we decided to use an already implemented code (that one of us used for another project) to testing BERT on reviews. We used the preprocessing of the IMDB dataset that we implemented for our previous models (except the tokenization, which we used the one included in the Pytorch module that is used for BERT).

We wanted to learn about transformers, encoders and decoders but knowing the limit of pages on this report, we did not want to add the theoretical aspect of BERT (only made by encoders). It is a very strong language model that has the ability of Attention (that revolutionized the world of NLP) and we trained a simple logistic classifier with the outputs of BERT (it transforms text and its tokens to vectors of numbers where each number is the output of a specific word) if we use the option of Classify on the input of BERT.

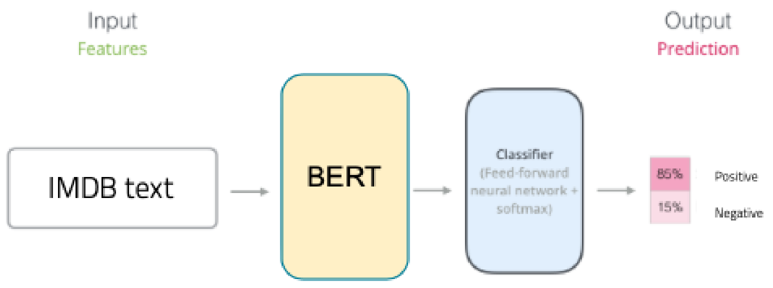


Figure 21: Structure of BERT and logistic classifier for IMDB

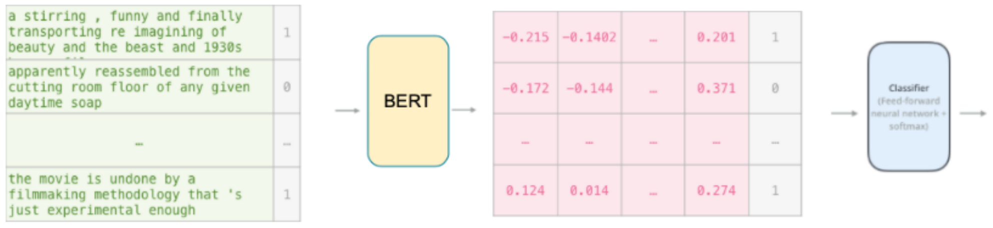

a stirring , funny and finally transporting re imagining of beauty and the beast and 1930s
1
apparently reassembled from the cutting room floor of any given daytime soap
0
the movie is undone by a filmmaking methodology that 's just experimental enough
1
BERT
-0.215
-0.1402
0.201
-0.172
-0.144
0.371
0.124
0.014
0.274
Classifier
(Feed-forward neural network + softmax)

Figure 22: How BERT and logistic classifier work together to classify IMDB reviews

Here are its results:

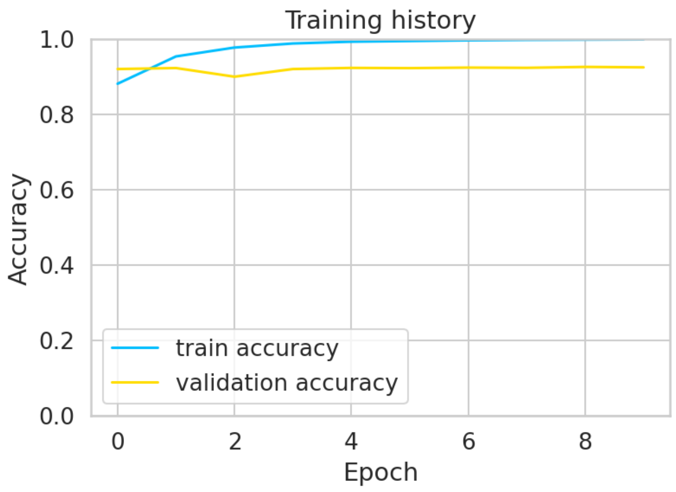


Figure 23: Accuracy results for BERT

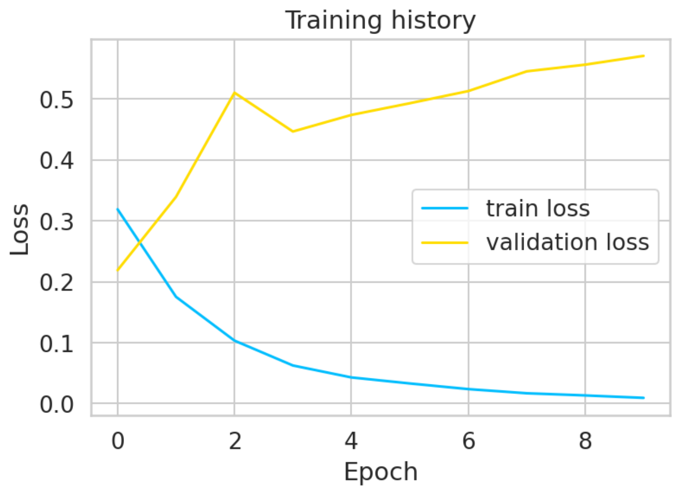


Figure 24: Loss results for BERT

Even though it might look like BERT is overfitting since the loss decreases to practically zero and the accuracy is almost 100% on the training set, the validation set results are generally impressive compared to the other 3 models used previously. We give this great performance to the capacity of BERT to be aware of the context of the words (Self Attention) using the entire surrounding context all-at-once.

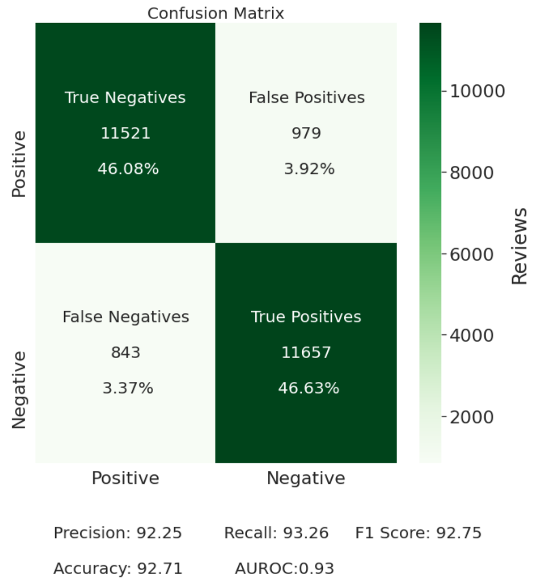


Figure 25: Confusion Matrix for BERT Uncased

And with this, the results on the test set, we observe an accuracy of 92% beating all the previous models. We want to show here the difference between the previous BERT that is trained for uncased text and what happens when the model is for Cased differentiation, which makes suffer the interpretation since writing something in caps might throw off the model.

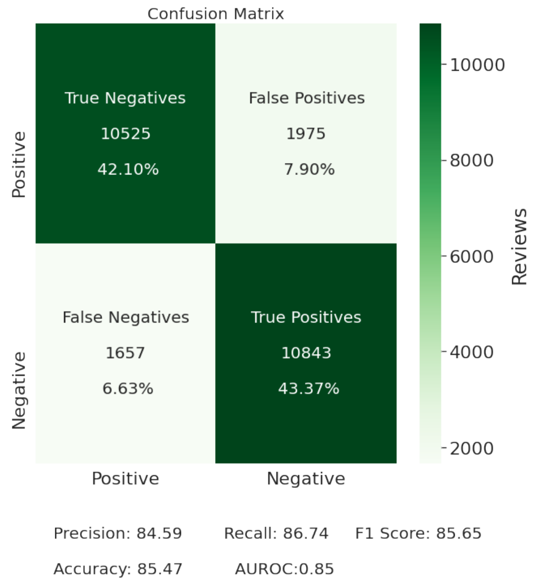


Figure 26: Confusion Matrix for BERT Cased

6 Conclusions

- **NLP**:

   1. We learned different techniques on how to handle NLP data such as getting rid of HTML tags, which made us realize that preprocessing is an arduous task and that is why there are many libraries and modules such as (tokenizer() from Keras) that handle those things so software engineers can focus on the model and the solution.
   2. Also, the results in general for our three implemented algorithms (RF, CNN and LSTM) were not near 100% which it may sound like it is not good enough but after getting in touch with this branch of AI and ML we can understand a little bit more why it is so hard. We definitely gained respect towards NLP.

- **Constraints for RF**:
   1. This algorithm has the problem of having computationally intensive tasks that we had to do to train the model (specially if we are executing this project on normal laptops).
   2. Another thing is that this algorithm works almost like a black box, we don't have much control of what is going on inside, we define very few factors (it has good and impressive results for some tasks but for others it might not work well), and those few parameters are the ones we played with.
   3. In addition, as we have seen on class with trees, we need that all our features to have information gain or entropy, else this algorithm will suffer a lot.
- **Pros for RF**:
   1. Commented on the constraints for RF, it can also be a positive view that RF acts as a black box since it does not the fine tuning that many Deep Learning models need on their hyperparamaters.
   2. Also, this algorithm is much more robust than a simple decision tree because if we make a wrong decision once or get an outlier data point, since it is a decision taken from many trees, it will probably not have much impact on the final prediction.
   3. In addition, we used it for classification but by reading papers we found that this algorithm can be used for regression tasks and it implicitly performs feature selection and generates uncorrelated decision trees (less work for the software engineer).
- **Constraints for CNN**:
   1. The first is the difficulty of CNNs in modelling long word relationships. Since they depend on word embeddings, we'd have to concatenate the entire sentence, which will almost certainly result in data sparsity.
   2. The second is linked to word order; since they lack the "recurrence" of RNNs, they are unable to keep track of the location of words within a sentence, resulting in knowledge loss.Given these advantages and disadvantages, we can identify a simple set of criteria for making CNNs shine: short sentences as input, or when special features need to be identified. Listed below are a few examples: Machine Translation, Emotion Detection, Sentiment Analysis, Named Entity Recognition. RNNs are also preferable if the job is highly sequential or requires the handling of long-distance relationships (Dialog Management, Semantic Role Labeling, POS Tagging).
- **Pros for CNN**:
   1. Convolutional Neural Networks have a significant speed advantage over Recurrent Neural Networks. The explanation for this is that CNNs can be parallelized, while RNNs cannot.The RNN will compute a state at each timestep T that is conditioned on the previous state at timestep T – 1. CNNs, on the other hand, will only use the local context (within the convolution window) to build their state.We can easily parallelize those computations and make our network many times faster because there is no conditionality between states.
   2. Another benefit of CNNs is their ability to extract the most important features (read: key information) from the word embedding matrix they're fed (read: from a sentence) using max-pooling layers, which is typically the case.The use of a max-pooling layer after a convolution layer reduces the output's dimensionality by preserving only the most significant n-gram features in the entire sentence.
- **Constraints for RNN(LSTM)**:
   1. Gradient vanishing and exploding problems.

2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using tanh or relu as an activation function.

- **Pros for RNN(LSTM)**:
  1. The main advantage of RNN over ANN is that RNN can model sequence of data (i.e. time series) so that each sample can be assumed to be dependent on previous ones
  2. Recurrent neural network are even used with convolutional layers to extend the effective pixel neighborhood.
- **Which model is better?**: To conclude with this, we want to present which model we would use for this specific task:
  1. For this question, it is fairly difficult to answer since Random Forest gave us many problems with RAM limitations on Google Colab but we think that CNN would be the best option since it did not require as much fine tuning as RNN to generate good results and it did not provide the same problems of RAM that RF gave us early.
  2. Of course, BERT outperforms the three models and it shows how the community is using the same structure of pretrained models on NLP (as Computer Vision is doing with several models pretrained on different datasets to the final purpose) and how powerful they can be in different applications just by using them as vectorizers.

7 Tasks

The sections that are not described were done by all the participants of the group.

- **En-Yu**: Worked on the RNN(LSTM) algorithm, Data Preprocessing and Adaptability of RNN & CNN
- **William Schallock**: Worked on the CNN algorithm and did all the proof reading.
- **Hamza Errahmouni Barkam**: Worked on the RF algorithm, BERT, Data Exploration and Data Preprocessing

Webgraphy

- Text Classification with Python and Scikit-Learn
  - https://stackabuse.com/text-classification-with-python-and-scikit-learn/
- A complete guide for Random Forest Algorithm
  - https://builtin.com/data-science/random-forest-algorithm
- Learning to Classify Text using NTLK Library
  - https://www.nltk.org/book/ch06.html
- Understanding Random Forest
  - https://towardsdatascience.com/understanding-random-forest-58381e0602d2
- Random Forest: Pros and Cons
  - https://medium.com/datadriveninvestor/random-forest-pros-and-cons-c1c42fb64f04
- Natural Language Processing In 10 Minutes | NLP Tutorial For Beginners
  - https://www.youtube.com/watch?v=5ctbvkAMQO4
- Natural Language Processing (NLP) & Text Mining Tutorial Using NLTK
  - https://www.youtube.com/watch?v=05ONoGfmKvA
- NLP Basics: What You Need To Know About Neuro Linguistic Programming
  - https://www.youtube.com/watch?v=d1KcaycYsRs
- Natural Language Processing - Tokenization (NLP Zero to Hero - Part 1)
  - https://www.youtube.com/watch?v=fNxaJsNG3-s
- A Simple Introduction to Natural Language Processing
  - https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b
- What Is Natural Language Processing?
  - https://machinelearningmastery.com/natural-language-processing/
- Natural Language Processing (NLP) - What it is and why it matters
  - https://www.sas.com/en_us/insights/analytics/what-is-natural-language-processing-nlp.
html
- HOW TO BUILD A NEURAL NETWORK WITH KERAS USING THE IMDB DATASET
  - https://builtin.com/data-science/how-build-neural-network-keras
- Python for NLP: Movie Sentiment Analysis using Deep Learning in Keras
  - https://stackabuse.com/python-for-nlp-movie-sentiment-analysis-using-deep-learning-in-ke
- What are the pros and cons of convolutional neural networks?
  - https://www.quora.com/What-are-the-pros-and-cons-of-convolutional-neural-networks
- Writing Quotes like Aristotle with Recurrent Neural Networks
  - https://medium.com/@adar.kahiri4/writing-quotes-like-aristotle-with-recurrent-neural-net
- Explained: Neural networks
  - https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414