Final Group Project Report

Srijon Mukhopadhyay and Zhuoya Li (Group 5)

Instructor: Dr. Amir Jafari
Class: Machine Learning II
Date: 4/03/2021

Contents

1. Introduction

Sentiment analysis has long been a problem for business, marketing and management areas for more value earned in the decision process. Previous work about sentiment analysis has been focusing on document-level, sentence-level and word-level sentiment extraction, with both supervised and unsupervised approaches. In this project, we are introducing a classification model for sentiment analysis with context information participated in the feature space.

2. Description of the dataset

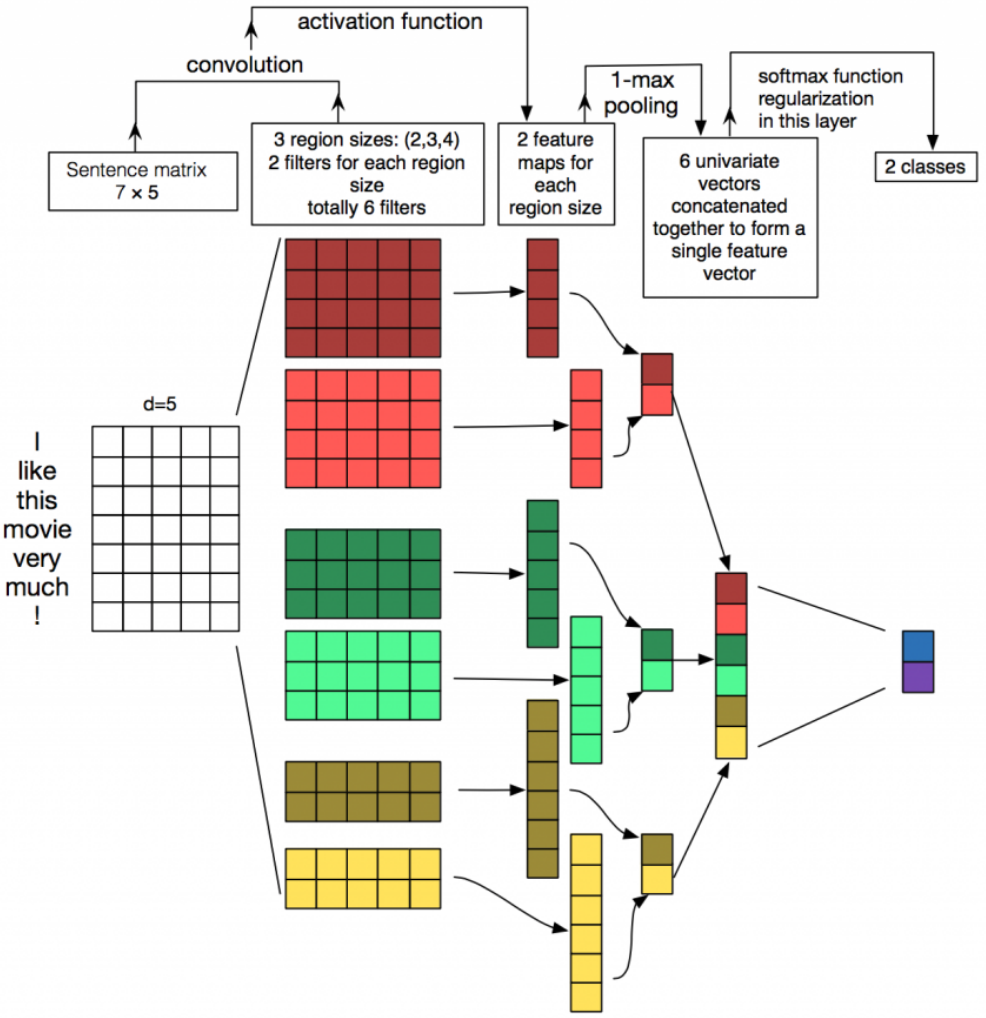
The labelled data set consists of 50,000 IMDB movie reviews, specially selected for sentiment analysis. The sentiment of reviews is binary, meaning the IMDB rating < 5 results in a sentiment score of 0, and rating >=7 have a sentiment score of 1. The 50,000 reviews are split into 25,000 for training and an additional 25,000 unlabeled data points for testing. In addition, there are another 50,000 IMDB reviews provided without any rating labels.

3. Background of the deep learning network and training algorithm

We will be building a Convolutional Neural Network (CNN) classifier and Long Short Term Memory (LSTM) to predict whether a sentiment is positive or negative.

Though the convolutional neural network is typically used for tasks such as image classification, its principle may also be applied for text classification purposes. A CNN can be used to convert text into stacked vector representations which may then be treated as an ‘image’. The CNN may then use filters that are of the same length as word embeddings which allows it to make shapes corresponding to ngrams. Ngrams allow the model to understand spatial representations of text which is important for any text classification task. The preprocessing required in a CNN is much lower as compared to other classification algorithms. While in primitive methods filters are

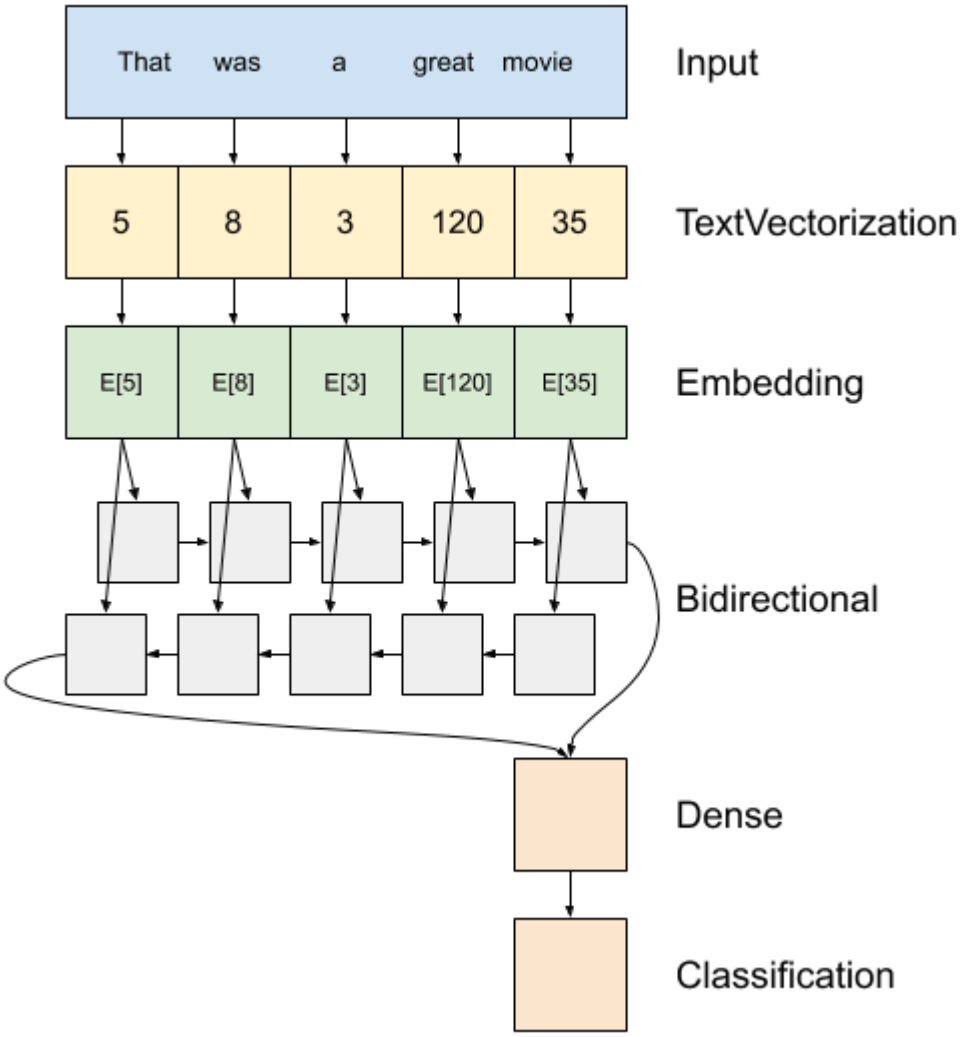
hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.



sourced from: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

A Long Short Term Memory (LSTM) model is a model that is typically used for text classification. It is a special type of an RNN model and is also able to save information about past inputs when processing the current input. Its advantage over the RNN model is that it is also able to retain information about long term dependencies. This solves the vanishing gradient problem that RNNs usually face. Since text data usually incorporates long term dependencies, this makes

LSTMs very useful for solving problems such as text classification, sentiment analysis, information retrieval, etc.



sourced from: https://www.tensorflow.org/tutorials/text/text_classification_rnn

4. Experimental setup

The data is divided into training and test sets with an 80% and 20% split. This implies that 20000 samples are kept for training the model while a dataset of 5000 samples is used to evaluate our models. When training the model, 33% of the training set will also be used as a validation set for the model to make predictions on and update weights accordingly.

Since this is a binary classification task, the models will have a final sigmoid activation layer which will output a value between 0 and 1. A value greater than 0.5 will be considered to be a positive sentiment while a value less than 0.5 is considered to be a negative sentiment.

The parameters that we will mostly be updating will be minibatch size and learning rate. Our primary readings online suggested that higher batch sizes generally tend to have lower accuracy when the model is evaluated on the test set. We have also read that this lost test accuracy may be recovered by increasing the learning rate.

We will thus try for two different cases in our models:

- Fits with higher batch sizes and learning rates
- Fits with lower batch sizes and learning rates

Once the data is fit we will evaluate the model by making predictions on the test set and evaluating the accuracy. We will also use our own metrics to assess the performance of our models.

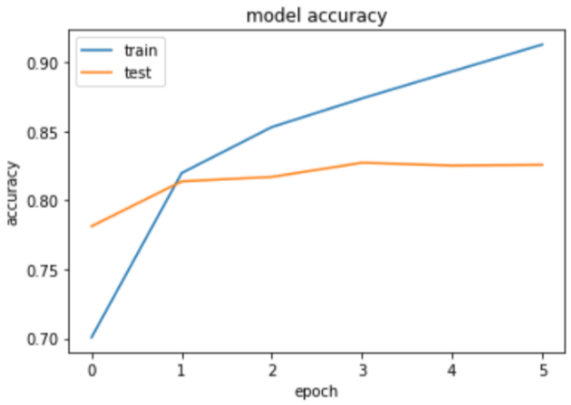
We will try different learning rates and plot the results. We will thus configure the batch size and learning rate so that they make up the optimal fit for our CNN and LSTM models.

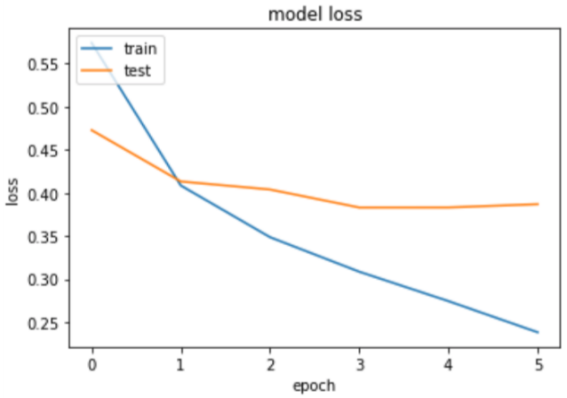
We first trained a convolutional neural network. Though in some cases our model fit the training data after 6 epochs, there were instances where we were required to change the epoch size to 15 to avoid overfitting.

Because we wanted to find the optimal relation between batch size and learning rate, four cases arose from the different model fits that we tested.

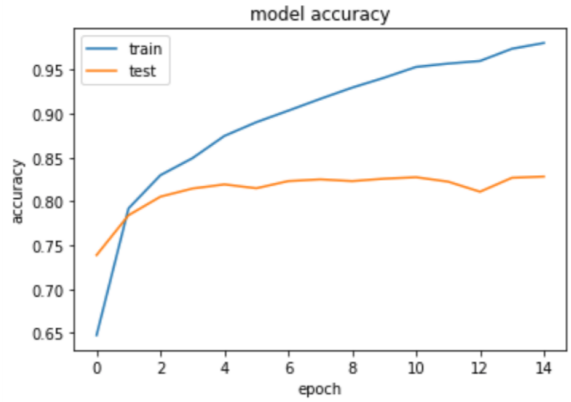
In order to keep the test set separate so as to not overfit the data and cause a generalizing gap, we also used 33% of the training data as validation data and evaluated our model's performance using the test set.

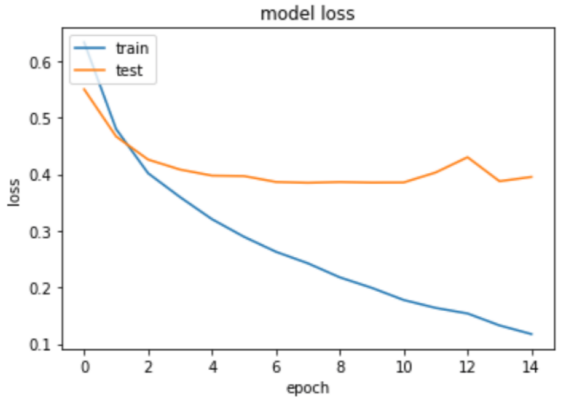
Firstly, we chose epochs as 6, batch size as 128, and the test accuracy gives us around 0.84, and model accuracy and model loss look like the following plots.





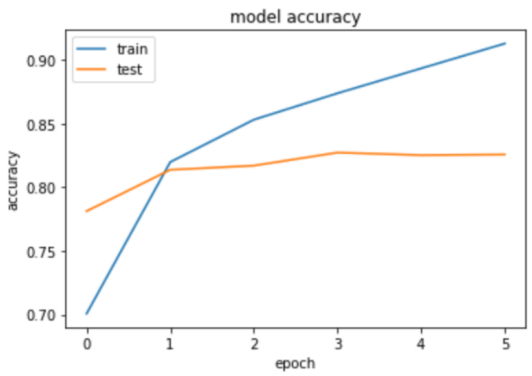
Next we changed the batch size as 32, and we get the test accuracy that is almost 0.84, we can see the plots:

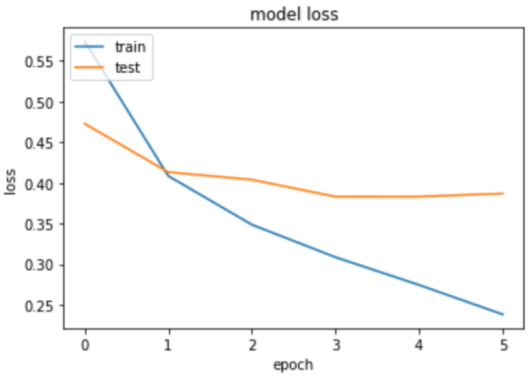




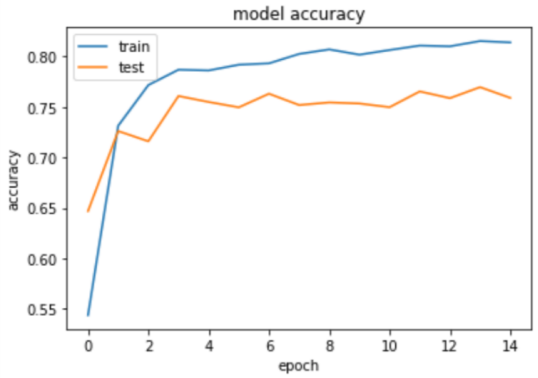
Then we are considering configuring the learning rate and batch size. As mentioned above, we read that lower batch sizes are able to capture gradients better and the test accuracy lost by higher batch sizes can be made up by increasing the learning rate. As we trained the model as batch size is 128, so we set 32 and 256 as the low batch size and high batch size separately, then with 0.01 and 0.1 of learning rate relevantly. So we get the related accuracy scores

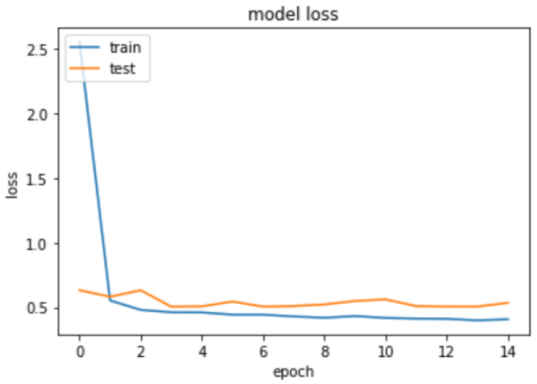
for each of them. One is about 0.78 with lower parameters, and the plots are shown below.





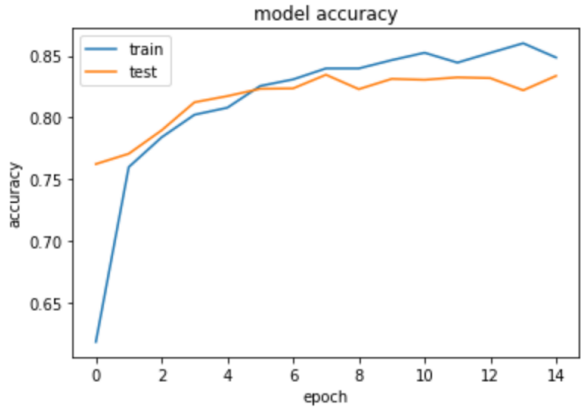
Another result we get with high parameters is around 0.76, and we can see the plots.

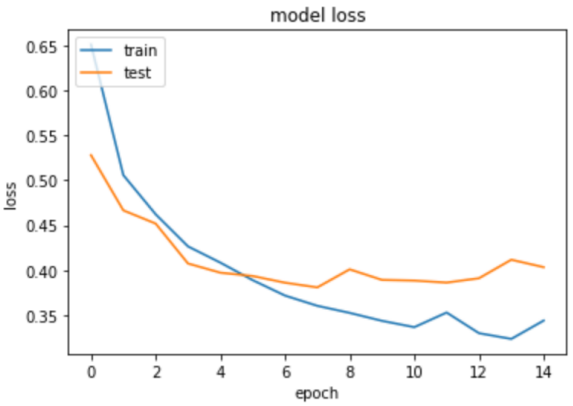




Overall, by comparing two scores and viewing the trend of two lines in plots, we can know that the accuracy is higher when learning rate and batch size are lower. Therefore, we could set those settings as our training parameters in this way.

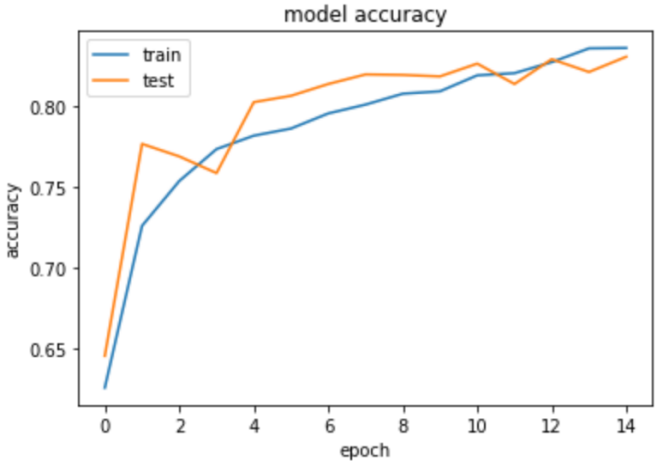
We also followed the same process for the LSTM model. The main goal was to find an optimal ratio between batch size and learning rate. We tried a higher batch size coupled with high learning rate:

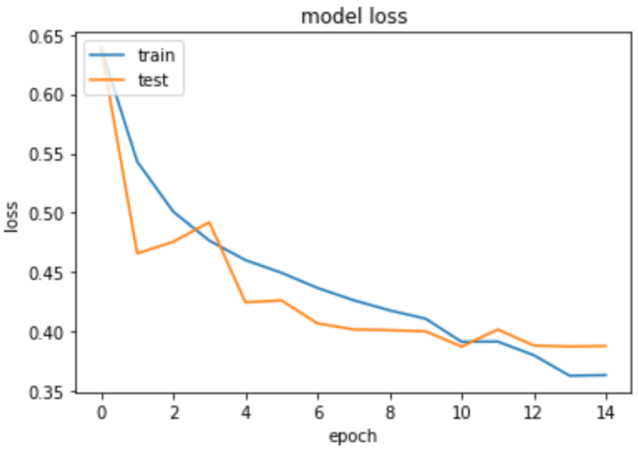




This gave us an accuracy of 0.83.

Next we tried another fit with lower batch size (32) and learning rate (0.001):





This gave us an accuracy of 0.84.

To prevent overfitting for the LSTM, a dropout layer of 0.3 was implemented along with recurrent dropouts of 0.2 so they wouldn't affect the final dense layer. Though overfitting was still not controlled to the best possible rate in the LSTM model there was a constraint that we had to face. The LSTM model in general requires a lot of data points to train. Since we were limited with our

training samples, adding higher dropout layers to the model would affect the fit of the model as well.

5. Results

We evaluated the best fits of our CNN and LSTM model mainly using three metrics:

- F1 Scores, Precision and Recall
- Confusion Matrix
- AUC curve

For CNN:

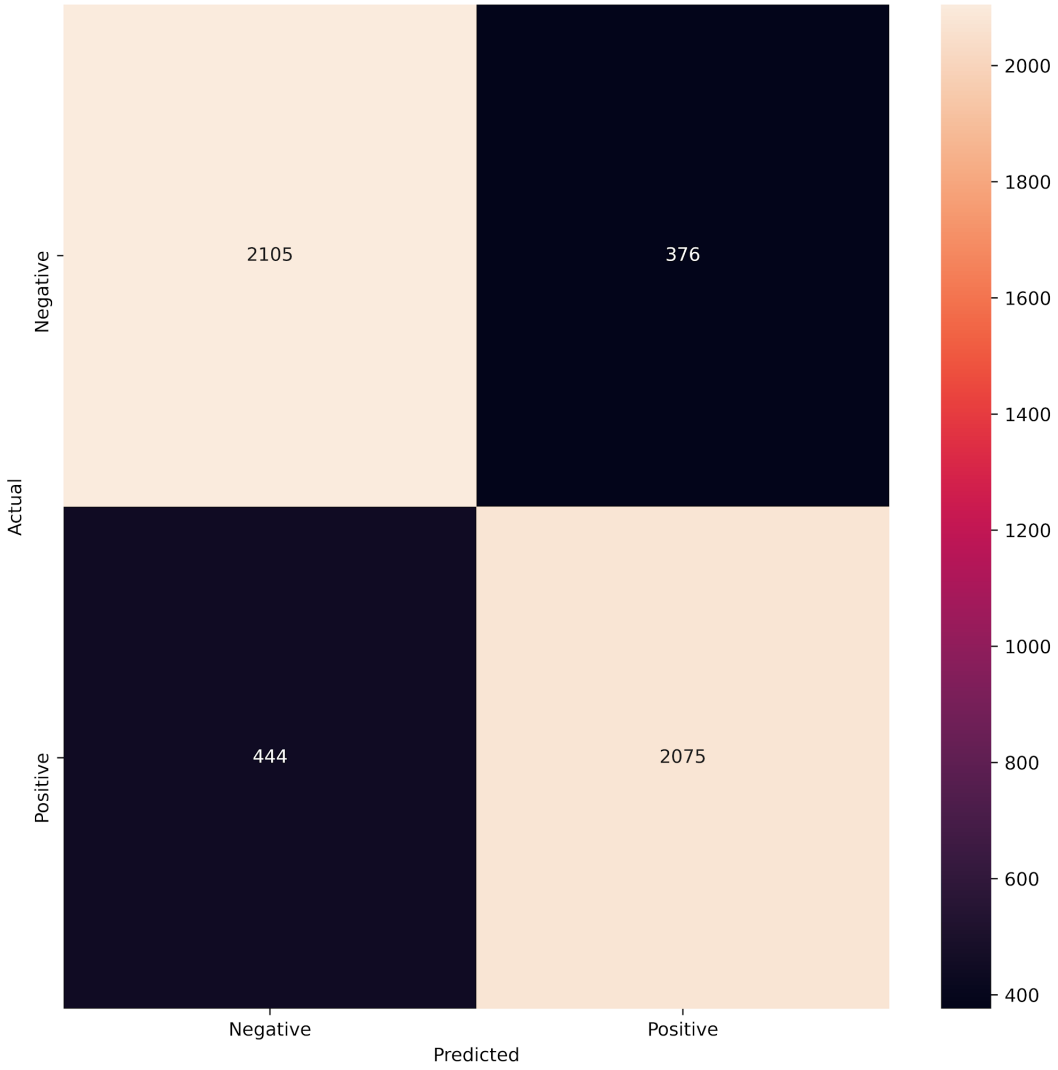
Sentiment(CNN)	Precision	Recall	F1 Score
Negative	0.83	0.85	0.84
Positive	0.85	0.82	0.84

Precision scores impies the proportion of the predicted instances that are actually relevant or predicted correctly, while Recall implies the proportion of relevant instances that the model is able to predict correctly. Since both precision cannot be improved without decreasing Recall, it is important to have another metric that provides a measure of how the model does in both cases. The F1 score is useful in this case and is the harmonic mean of the precision and recall scores of the model.

A high F1 score would imply that the model is able to control for a high a number of false negatives and false positives and can thus make reliable predictions.

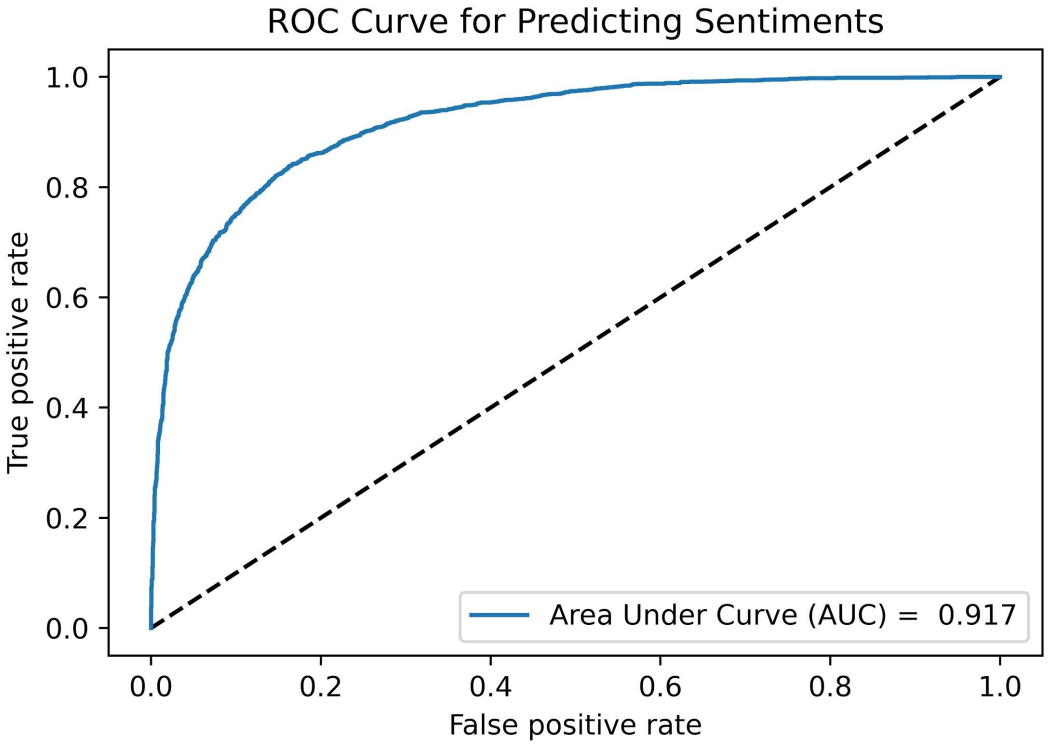
The CNN model had an F1 score of 0.84 for both the negative and positive sentiments which implies that there was no imbalance and one sentiment wasn't detected unfavorably over the other by the model.

For this type of problem, we would assume that there would be more emphasis on not mistakenly predicting negative sentiments as positive sentiments than the opposite. Especially in the case of business sectors, it is important to identify negative sentiments correctly so as to make better business decisions that will keep customers happy. We thus want to have fewer false positives for predicting positive sentiments than false negatives.



Looking at the confusion matrix, negative sentiments were predicted correctly 2105 times out of a total 2481 instances. Positive sentiments were predicted correctly 2075 times out of 2519 total instances.

The CNN model thus has 376 false positives when it attempts to predict positive sentiments.



Next, we plotted an AUC-ROC curve. The ROC curve, a performance measurement, is a probability curve between the True Positive Rate and False Positive Rate. We made the AUC-ROC curve based on how many times our model predicted positive sentiments.

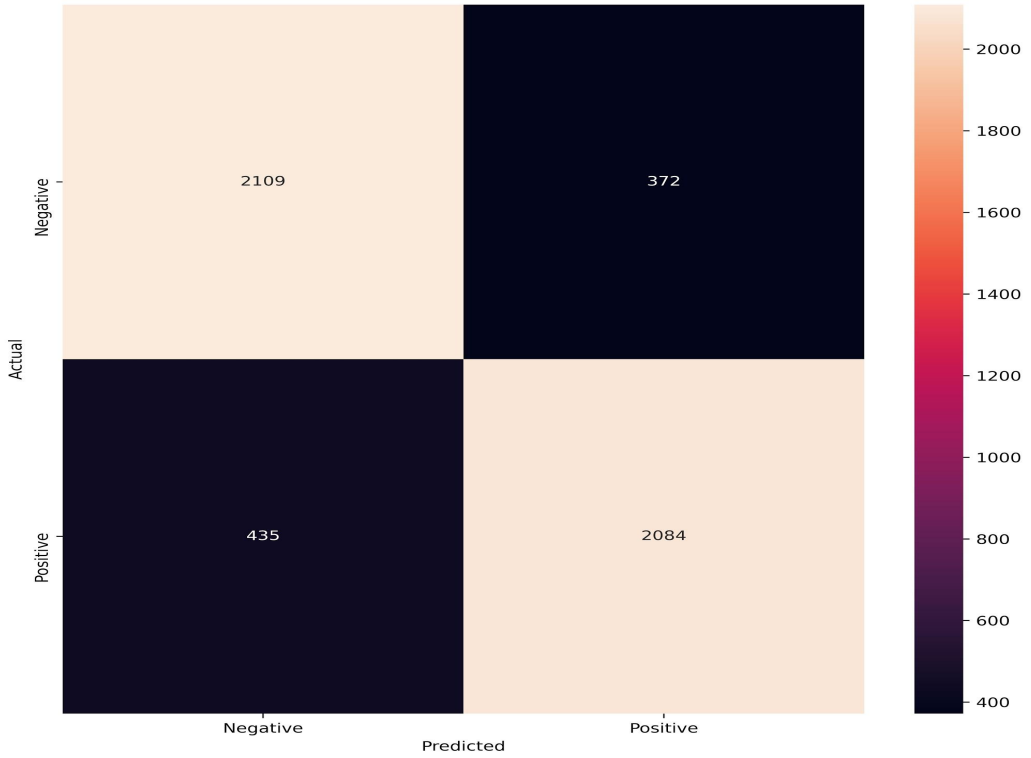
The AUC is the area under the curve of the ROC curve. Essentially, it is a degree of separability and its values can range between 0 and 1. A value of 0.5 would mean that the model is not differentiating at all between the two classes and 1 would mean that the model is perfectly distinguishing between positive and negative cases. When the value is 0, it means the model is predicting negative sentiments as positive sentiments and vice versa.

Our CNN model has a AUC of 0.917 which implies a 91% degree of separability between positive and negative sentiments. This is a good score.

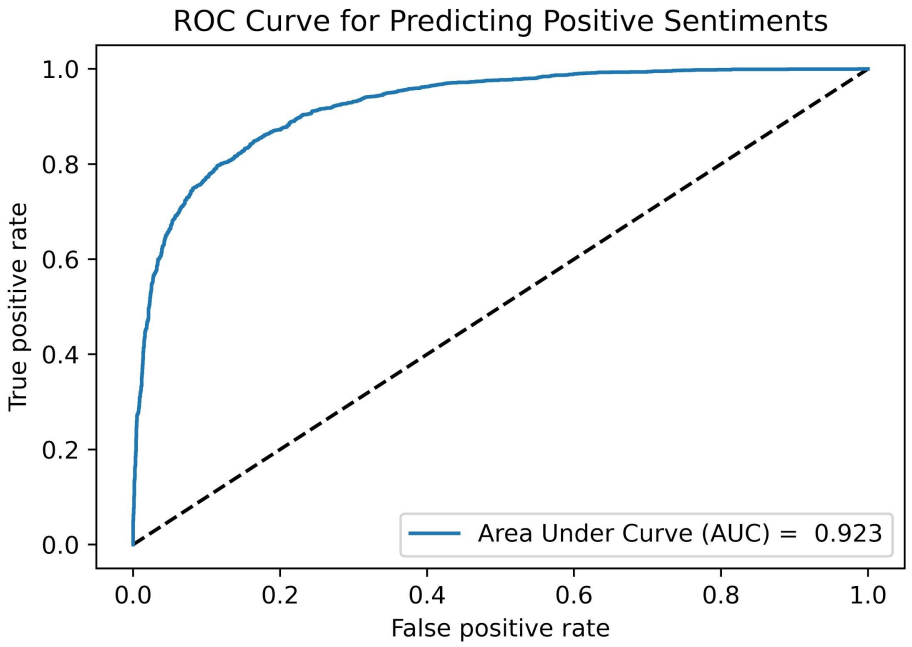
For LSTM:

Sentiment (LSTM)	Precision	Recall	F-1 Score
Negative	0.83	0.85	0.84
Postive	0.85	0.83	0.84

The LSTM model also displayed the same F1 scores as the CNN model.



Though the F1 scores for both the CNN and LSTM models are the same, the LSTM model has 372 false positive errors when it attempts to predict positive sentiments from the test set. It thus performs better than CNN in the proposed context of the problem.



The AUC of the LSTM model is 92%. This is slightly more than the AUC of the CNN model, implying a better performance in being able to distinguish between positive and negative sentiments from the test dataset.

6. Conclusions

The LSTM model slightly outperforms the CNN model with it having:

- Lesser number of false positive errors
- Higher AUC scores
- Lesser number of false negative errors (though this was not of high priority)

Both LSTM and CNN models performed better with lower batch sizes and learning rates. Even though we followed our previous readings and fixed for the higher batch sizes by increasing their corresponding learning rates, the models still fit the training data better when the batch sizes and learning rates were lower.

This is probably because the higher learning rates do not offset the problem of higher batch sizes well. Higher batch sizes imply random gradient updates which may be very small or very large, all depending on the sample. Furthermore, if the total loss is averaged over the batch, then larger batch sizes do not offer the model much flexibility in traveling from its initial weight updates.

In contrast, the smaller batch size almost acts as a regularizing feature, allowing for uniform gradient updates and allowing more flexibility for models in updating weights. The lower learning rate is also better for converging to an optimum and prevents the risk of overshooting the point, as higher learning rates sometimes do. This also helps prevent overfitting and thus leads to better accuracies on the test set.

This project can be improved. Once major issue that we ran into was our model overfitting on the training data. This was not something we were able to control for as there weren't enough training data samples for us to increase our dropout without affecting model performance. One way to improve our model performance would be to increase the number of training samples.

More experimentation with hyper parameter tuning is also required. To improve this, it may be worthwhile to consider different ways to change the learning rates while the epochs are running (decays such as time decay, exponential decay, etc). For this task we used Adam as the optimizer, which is an adaptive optimizer. Since Adam updates the learning rate by itself, we ultimately did not go ahead with any decaying but it would also be interesting to use another optimizer like Stochastic Gradient and try different methods to lower learning rates while calibrating momentum values.

7. References

https://wiki.pathmind.com/accuracy-precision-recall-f1

https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17

https://machinelearningmastery.com/predict-sentiment-movie-reviews-using-deep-learning/

https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

https://arxiv.org/pdf/1711.00489.pdf

https://www.tensorflow.org/tutorials/text/text_classification_rnn

http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/

8. Appendix

```
# Importing Required Libraries
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords

from numpy import array
import tensorflow
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers.core import Activation, Dropout, Dense
from keras.layers import Flatten
from keras.layers import GlobalMaxPooling1D
from keras.layers.embeddings import Embedding
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
import matplotlib.pyplot as plt
# Import the dataset
movie_reviews_labeled = pd.read_csv("labeledTrainData.tsv", header=0, delimiter="\t", quoting=3)
movie_reviews_unlabeled = pd.read_csv("unlabeledTrainData.tsv", header=0, delimiter="\t",
quoting=3)
test_data = pd.read_csv("testData.tsv", header=0, delimiter="\t", quoting=3)
# Convert .tsv file to .csv file and read the csv file
movie_reviews_labeled.to_csv('movie_reviews_labeled.csv')
movie_reviews = pd.read_csv('movie_reviews_labeled.csv')
# Remove quotations from string and print first 5 rows of the dataset
movie_reviews['review'] = movie_reviews['review'].str.strip('" "')
movie_reviews.head()
# Take a text string as a parameter
# Performs preprocessing on the string to remove special chracters from the string

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)

# Preprocess our reviews and will store them in a new list as shown below
X = []
sentences = list(movie_reviews['review'])
for sen in sentences:
  X.append(preprocess_text(sen))
y = movie_reviews['sentiment']
# Divide the dataset into 80% for training set and 20% for testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
# Prepare the embedding layer
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X_train)

X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)
# Find the vocabulary size and then perform padding on both train and test set
# Adding 1 because of reserved 0 index
vocab_size = len(tokenizer.word_index) + 1

maxlen = 100

X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
!wget nlp.stanford.edu/data/glove.6B.zip
#unzipping the zip files and deleting the zip files
!unzip \*.zip  && rm *.zip
# Load the GloVe word embeddings
# Create a dictionary that will contain words as keys and their corresponding embedding list as values.
from numpy import array
from numpy import asarray
from numpy import zeros

embeddings_dictionary = dict()
glove_file = open('glove.6B.100d.txt', encoding="utf8")

for line in glove_file:
  records = line.split()
  word = records[0]
  vector_dimensions = asarray(records[1:], dtype='float32')
  embeddings_dictionary [word] = vector_dimensions
glove_file.close()
# Create an embedding matrix where each row number will correspond to the index of the word in the
corpus
embedding_matrix = zeros((vocab_size, 100))
for word, index in tokenizer.word_index.items():
  embedding_vector = embeddings_dictionary.get(word)
  if embedding_vector is not None:
      embedding_matrix[index] = embedding_vector
```

```
# Create a simple convolutional neural network with 1 convolutional layer and 1 pooling layer
from keras.layers.convolutional import Conv1D
model = Sequential()

embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlen , 
trainable=False)
model.add(embedding_layer)

model.add(Conv1D(filters=128, kernel_size=5, padding='same', activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(1, activation='sigmoid'))
#opt = tensorflow.keras.optimizers.Adam(learning_rate = 0.01)
model.compile(loss='binary_crossentropy', metrics=['acc'], optimizer='adam')

print(model.summary())
# Train our model and evaluate it on the training set
history = model.fit(X_train, y_train, batch_size=32, epochs=15, verbose=1, validation_split=0.33)

score = model.evaluate(X_test, y_test, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()
pred_cnn = model.predict(X_test)
pred_cnn = (pred_cnn>0.5).astype(int)
import seaborn as sns
conf_mat = confusion_matrix(y_test, pred_cnn)
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(conf_mat, annot=True, fmt='d',
        xticklabels=['Negative','Positive'], yticklabels = ['Negative','Positive'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.savefig('cnn_cm', dpi = 720)
plt.show()
```

```
report = metrics.classification_report(y_test, pred_cnn)
from sklearn.metrics import roc_curve, roc_auc_score
pred_cnn = model.predict(X_test)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, pred_cnn)

from sklearn.metrics import auc

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='Area Under Curve (AUC) =  {:.3f}'.format(auc(fpr,tpr)))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC Curve for Predicting Sentiments')
plt.legend(loc='best')
plt.savefig('auc_cnn.jpg', dpi = 720)
plt.show()



# Create a Bidirectional LSTM with low learning rate

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, LSTM, Bidirectional, Dense, SpatialDropout1D
from tensorflow.keras.layers import Embedding, Flatten
from tensorflow.keras.layers import MaxPooling1D, Dropout, Activation, Conv1D

model = Sequential()
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlen ,
trainable=False)
model.add(embedding_layer)
model.add(SpatialDropout1D(0.3))
model.add(Bidirectional(LSTM(100, dropout=0.3, recurrent_dropout=0.2)))
model.add(Dense(1, activation='sigmoid'))
opt = tensorflow.keras.optimizers.Adam(learning_rate = 0.001)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])

print(model.summary())
history = model.fit(X_train, y_train, batch_size=32, epochs=15, verbose=1, validation_split=0.33)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
pred_lstm = model.predict(X_test)
pred_label_lstm = (pred_lstm>0.5).astype(int)
report = metrics.classification_report(y_test, pred_label_lstm)
print(report)
import seaborn as sns
conf_mat= confusion_matrix(y_test, pred_label_lstm)
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(conf_mat, annot=True, fmt='d',
```

```
        xticklabels=['Negative','Positive'], yticklabels = ['Negative','Positive'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.savefig('lstm_cm.jpg', dpi = 720)
plt.show()
from sklearn.metrics import roc_curve, roc_auc_score
pred_lstm = model.predict(X_test)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, pred_lstm)

from sklearn.metrics import auc

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='Area Under Curve (AUC) =  {:.3f}'.format(auc(fpr,tpr)))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC Curve for Predicting Positive Sentiments')
plt.legend(loc='best')
plt.savefig('auc_lstm.jpg', dpi = 720)
plt.show()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, LSTM, Bidirectional, Dense, SpatialDropout1D
from tensorflow.keras.layers import Embedding, Flatten
from tensorflow.keras.layers import MaxPooling1D, Dropout, Activation, Conv1D

model = Sequential()
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlen ,
trainable=False)
```

```
model.add(embedding_layer)
model.add(SpatialDropout1D(0.3))
model.add(LSTM(128,recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
opt = tensorflow.keras.optimizers.Adam(learning_rate = 0.01)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])

print(model.summary())
from tensorflow.keras.callbacks import LearningRateScheduler
# Train our model and evaluate it on the training set
history = model.fit(X_train, y_train, batch_size=128, epochs=15, verbose=1, validation_split=0.33)

score = model.evaluate(X_test, y_test, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()
```