Sentiment Analysis Model

Natural Language Processing

NGUYEN THIEN TOAN

Outline:

INTRODUCTION

MODELS

INVESTIGATION OF IMPROVING MODELS

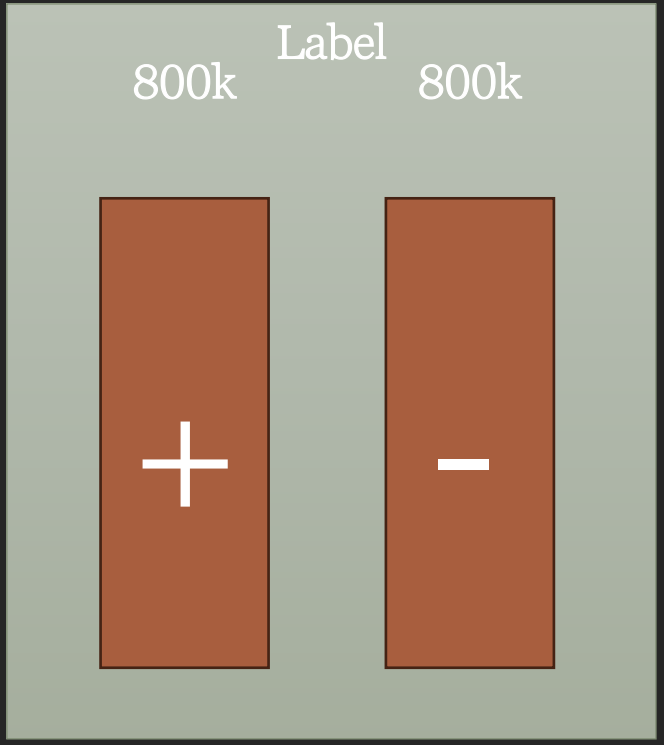
Part 1: Introduction

This project's aim, is to explore the world of *Natural Language Processing* (NLP) by building what is known as a Sentiment Analysis Model. A sentiment analysis model is a model that analyses a given piece of text and predicts whether this piece of text expresses positive or negative sentiment.

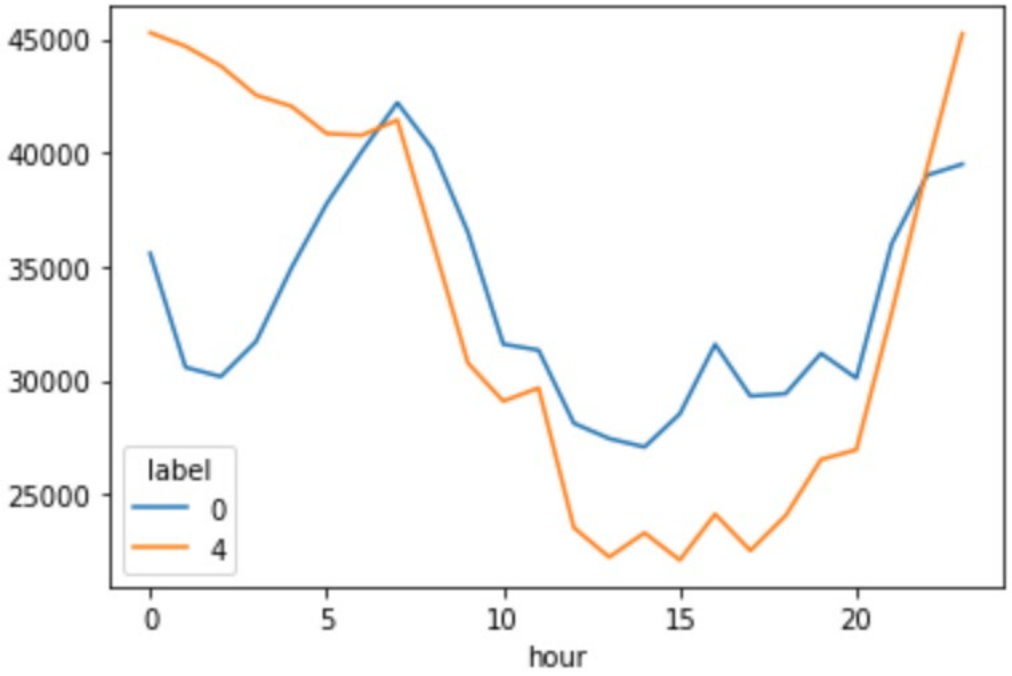
To this end, we will be using the **sentiment140** dataset containing data collected from twitter. An impressive feature of this dataset is that it is perfectly balanced (From Kaggle)

	label	time	date	query	username	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

https://www.kaggle.com/datasets/kazanova/sentiment140?resource=download



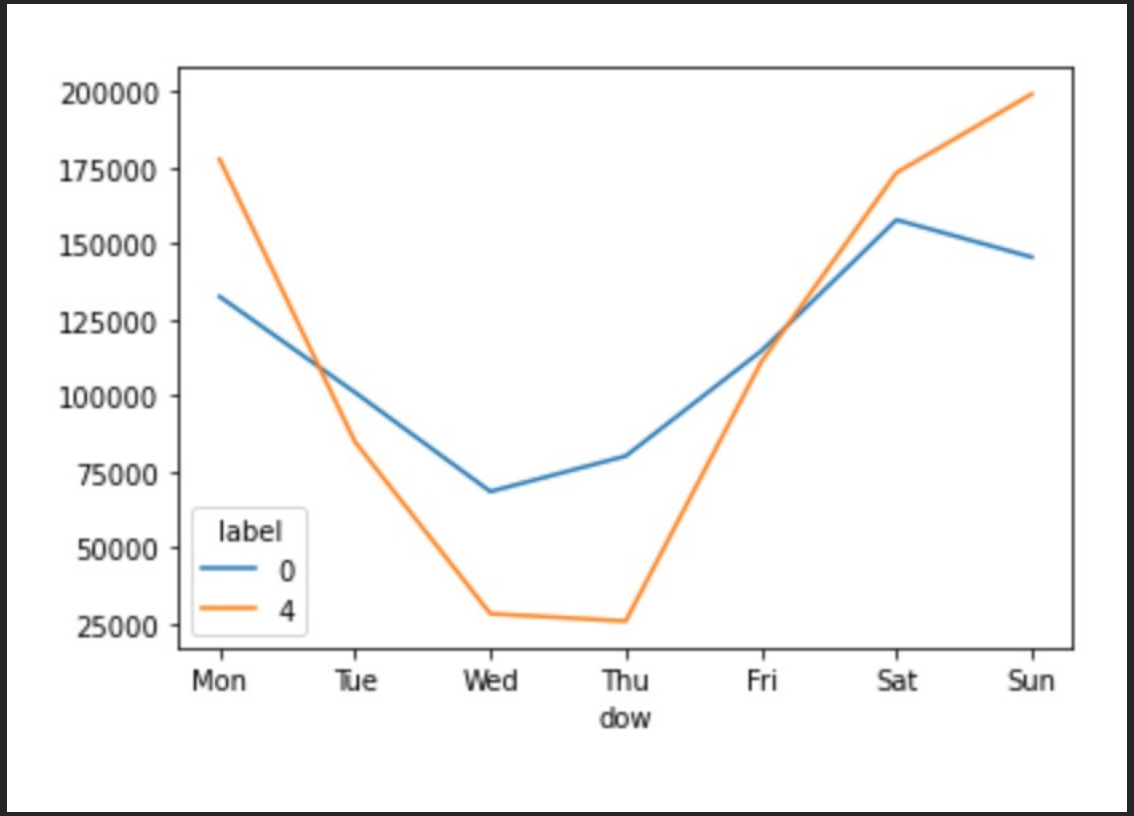
This illustrate the distribution of positive and negative tweet



In fact, from this graph I concluded two things:

- There's a specific interval of time in which positive tweets outnumber the negative tweets, and another interval in which the opposite occurs.
- When the total number of tweets is relatively high, the majority of the tweets tend to have a positive sentiment.

  This could also mean that positive tweets come in big batches, maybe relating to a similar subject (soccer game, movie release, etc), whereas negative tweets are more sparsely distributed.



Here also, two insights can be drawn from the graph:

- People tend to tweet more positively during the weekends, when they relax and attend events, whereas negative tweets are dominant during the rest of the week, showcasing people's dissatisfaction with their work.
- A much bigger number of tweets is circulated during the weekend, relative to the rest of the week.

the plan anticipates possible snags that might be incorporated into preliminary hypothesis of the data.

No model is perfect. They sometimes get some troubles. As a Machine Learning Engineer/Data Scientist, It should consider some points during building this deep learning model for sentiment analysis task:

- Data is raw. The variety of words can affect model recognition. Cleaning is a necessary task to improve the model as well. Keep cleaning.
- Every day, there is a ton of tweet on Tweeter meaning that the amount of data to train model is huge also. Cleaning/Analyzing data is a important step to deal with it in order to find out a key value in data.

Model

Naive Bayesian model

Naive Bayesian classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but rather a family of algorithms where all of them make the following *naive* assumptions:

- All features are independent from each other.
- Every feature contributes equally to the output.

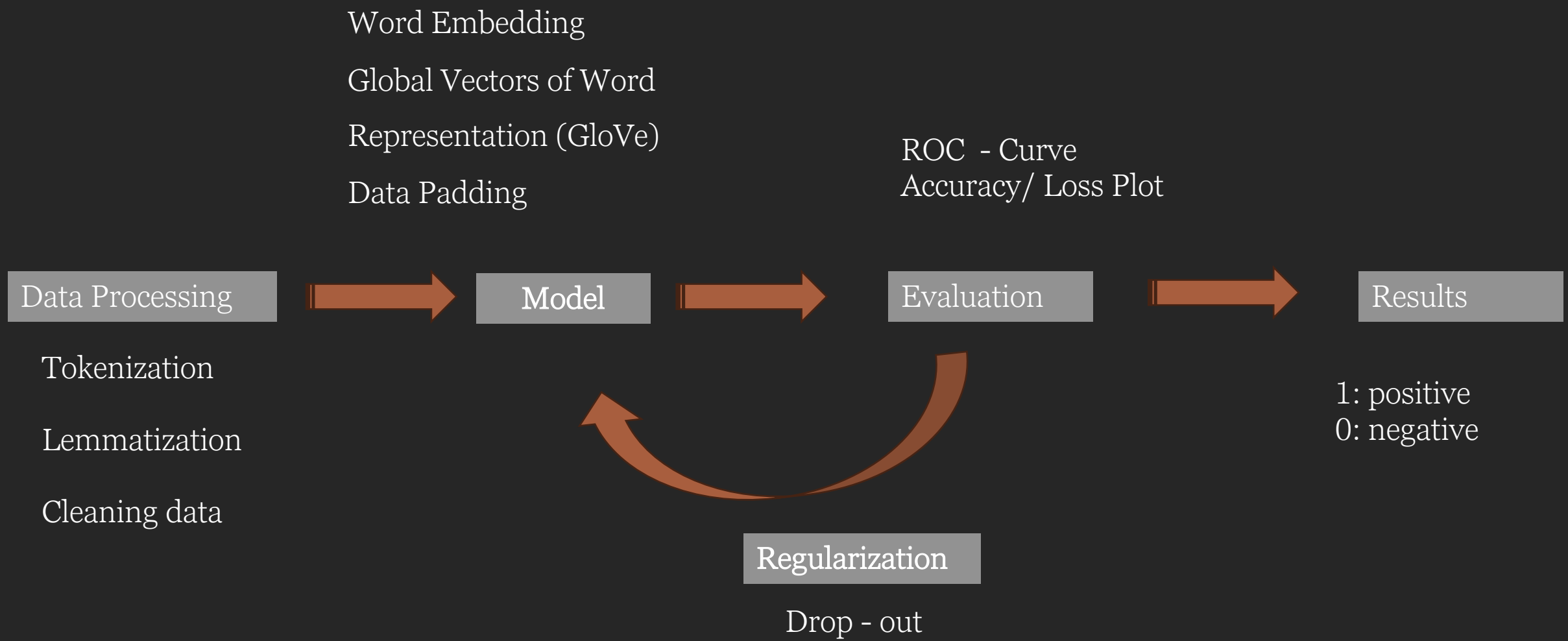
A Long Short-Term Memory, or LSTM

A Long Short-Term Memory, or **LSTM**, is a type of machine learning neural networks. More specifically, it belongs to the family of Recurrent Neural Network (**RNN**) in Deep Learning, which are specifically conceived in order to process *temporal data*.

Part 2: Models

We should run 3 models:

- Naive Bayesian
- LSTM models
- Drop out LSTM Models
- CNN


Word Embedding
Global Vectors of Word
Representation (GloVe)
Data Padding
ROC - Curve
Accuracy/ Loss Plot
Data Processing
Model
Evaluation
Results
Tokenization
Lemmatization
Cleaning data
1: positive
0: negative
Regularization
Drop - out

```python
start_time = time()

from nltk.tokenize import TweetTokenizer
# The reduce_len parameter will allow a maximum of 3 consecutive repeating characters, while trimming the rest
# For example, it will tranform the word: 'Hellooooooooo' to: 'Hellooo'
tk = TweetTokenizer(reduce_len=True)

data = []

# Separating our features (text) and our labels into two lists to smoothen our work
X = df['text'].tolist()
Y = df['label'].tolist()

# Building our data list, that is a list of tuples, where each tuple is a pair of the tokenized text
# and its corresponding label
for x, y in zip(X, Y):
    if y == 4:
        data.append((tk.tokenize(x), 1))
    else:
        data.append((tk.tokenize(x), 0))

# Printing the CPU time and the first 5 elements of our 'data' list
print('CPU Time:', time() - start_time)
data[:5]
```

```
CPU Time: 47.039098262786865


[(['I', 'LOVE', '@Health4UandPets', 'u', 'guys', 'r', 'the', 'best', '!', '!'],
  1),
 (['im',
   'meeting',
   'up',
   'with',
   'one',
   'of',
   'my',
   'besties',
   'tonight',
   '!',
   'Cant',
   'wait',
   '!',
   '!',
   '-',
   'GIRL',
   'TALK',
   '!',
   '!'],
  1),
 (['@DaRealSunisaKim',
```

Tokenization

Lemmatization

```python
def lemmatize_sentence(tokens):
    lemmatizer = WordNetLemmatizer()
    lemmatized_sentence = []
    for word, tag in pos_tag(tokens):
        # First, we will convert the pos_tag output tags to a tag format that the WordNetLemmatizer can interpret
        # In general, if a tag starts with NN, the word is a noun and if it stars with VB, the word is a verb.
        if tag.startswith('NN'):
            pos = 'n'
        elif tag.startswith('VB'):
            pos = 'v'
        else:
            pos = 'a'
        lemmatized_sentence.append(lemmatizer.lemmatize(word, pos))
    return lemmatized_sentence

# Previewing the WordNetLemmatizer() output
print(lemmatize_sentence(data[0][0]))
```

```
['I', 'LOVE', '@Health4UandPets', 'u', 'guy', 'r', 'the', 'best', '!', '!']
```

```python
def cleaned(token):
    if token == 'u':
        return 'you'
    if token == 'r':
        return 'are'
    if token == 'some1':
        return 'someone'
    if token == 'yrs':
        return 'years'
    if token == 'hrs':
        return 'hours'
    if token == 'mins':
        return 'minutes'
    if token == 'secs':
        return 'seconds'
    if token == 'pls' or token == 'plz':
        return 'please'
    if token == '2morow':
        return 'tomorrow'
    if token == '2day':
        return 'today'
    if token == '4got' or token == '4gotten':
        return 'forget'
    if token == 'amp' or token == 'quot' or token == 'lt' or token == 'gt' or token == '½25':
        return ''
    return token
```

```python
def remove_noise(tweet_tokens):

    cleaned_tokens = []

    for token, tag in pos_tag(tweet_tokens):
        # Eliminating the token if it is a link
        token = re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+#]|[!*\(\),]|'\
                       '(?:%[0-9a-fA-F][0-9a-fA-F]))+','', token)
        # Eliminating the token if it is a mention
        token = re.sub("(@[A-Za-z0-9_]+)","", token)

        if tag.startswith("NN"):
            pos = 'n'
        elif tag.startswith('VB'):
            pos = 'v'
        else:
            pos = 'a'

        lemmatizer = WordNetLemmatizer()
        token = lemmatizer.lemmatize(token, pos)

        cleaned_token = cleaned(token.lower())


        # Eliminating the token if its length is less than 3, if it is a punctuation or if it is a stopword
        if cleaned_token not in string.punctuation and len(cleaned_token) > 2 and cleaned_token not in STOP_WORDS:
            cleaned_tokens.append(cleaned_token)


    return cleaned_tokens
```

Problem: There are some sentences with teencode or abbrev

Example: u = you ; r = are; 2morrow = tomorrow

```python
# Prevewing the remove_noise() output
print(remove_noise(data[0][0]))
```

```
['love', 'guy', 'best']
```

As Naïve Bayesian model only accepts format of input as dict – like structure. Now, need a step of transformation

```
def list_to_dict(cleaned_tokens):
    return dict([token, True] for token in cleaned_tokens)

cleaned_tokens_list = []

# Removing noise from all the data
for tokens, label in data:
    cleaned_tokens_list.append((remove_noise(tokens), label))

print('Removed Noise, CPU Time:', time() - start_time)
start_time = time()

final_data = []

# Transforming the data to fit the input structure of the Naive Bayesian classifier
for tokens, label in cleaned_tokens_list:
    final_data.append((list_to_dict(tokens), label))

print('Data Prepared for model, CPU Time:', time() - start_time)

# Previewing our final (tokenized, cleaned and lemmatized) data list
final_data[:5]
```

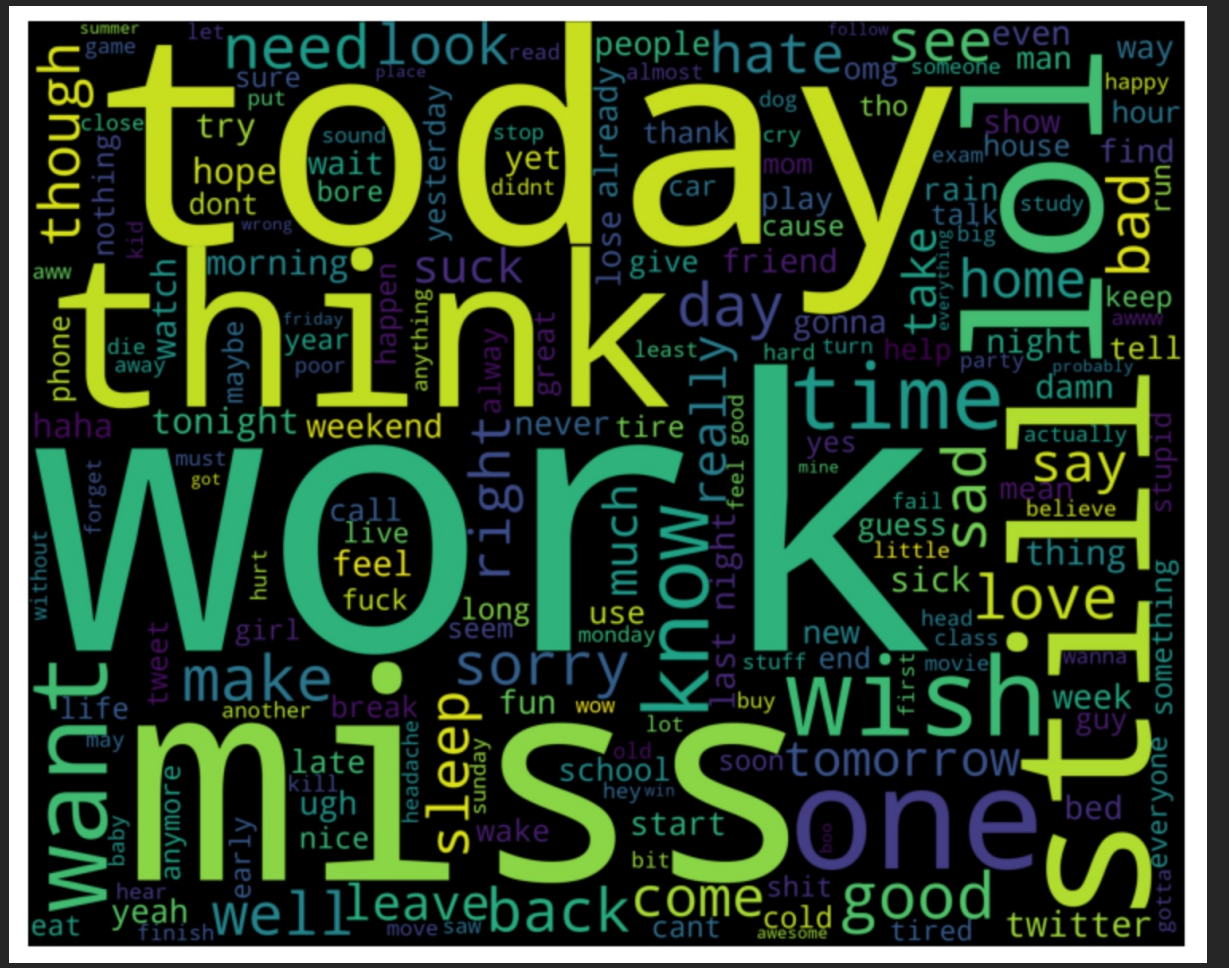
```
Removed Noise, CPU Time: 497.73784041404724
Data Prepared for model, CPU Time: 1.777745246887207
```

Out[12]:

```
[({'love': True, 'guy': True, 'best': True}, 1),
 ({'meet': True,
   'one': True,
   'besties': True,
   'tonight': True,
   'cant': True,
   'wait': True,
   'girl': True,
   'talk': True},
  1),
 ({'thanks': True,
   'twitter': True,
   'add': True,
   'sunisa': True,
   'get': True,
   'meet': True,
   'hin': True,
   'show': True,
   'area': True,
   'sweetheart': True},
  1),
 ({'sick': True,
   'really': True,
```

Positive

Negative



```
Accuracy on train data: 0.8107888888888889
Accuracy on test data: 0.755725
Most Informative Features
                depressed = True              0 : 1      =     49.0 : 1.0
                toothache = True              0 : 1      =     45.0 : 1.0
                     roni = True              0 : 1      =     34.3 : 1.0
                  unhappy = True              0 : 1      =     31.4 : 1.0
                    strep = True              0 : 1      =     31.0 : 1.0
                   asthma = True              0 : 1      =     26.3 : 1.0
                  unloved = True              0 : 1      =     25.0 : 1.0
                   #movie = True              1 : 0      =     23.0 : 1.0
                   gutted = True              0 : 1      =     22.3 : 1.0
                    hates = True              0 : 1      =     21.9 : 1.0
                heartburn = True              0 : 1      =     21.7 : 1.0
                      bom = True              1 : 0      =     21.4 : 1.0
                    ugggh = True              0 : 1      =     19.7 : 1.0
               depressing = True              0 : 1      =     19.4 : 1.0
                      ftl = True              0 : 1      =     19.0 : 1.0
                    waaah = True              0 : 1      =     18.6 : 1.0
                 coughing = True              0 : 1      =     18.2 : 1.0
           congratulation = True              1 : 0      =     18.1 : 1.0
                      sad = True              0 : 1      =     17.9 : 1.0
          congratulations = True              1 : 0      =     17.5 : 1.0
None

CPU Time: 35.97064781188965
```

a 75.5% accuracy on the test set training a very *Naive* algorithm and in just 36 seconds!

Taking a look at the 20 most informative features of the model, we can notice the high volume of negative to positive (0:1) informative features. This is very interesting as it means that negative tweets have a much more concentrated and limited vocabulary when compared to positive tweets.

I personally interpret this as follows: *Whenever people are in a bad mood, they are confined in such a limited space of words and creativity, in contrast with when they are in a happy mood*.

Word Embedding

Word embeddings are basically a way for us to convert words to representational *vectors*. What I mean by this is that, instead of mapping each word to an index, we want to map each word to a vector of real numbers, representing this word.

Global Vectors of Word Representation (GloVe)

Building and training good word embeddings is a tremendous process requiring millions of data samples and exceptional computational power. Luckily for us, folks at the University of Stanford already did this for us and published their results for free on their official website Their model is called **GloVe**.

Data Padding

Further in our training we would like to speed the process up by splitting data into *mini-batches*. Batch learning is basically the process of training on several examples at the same time, which greatly decreases the training time!
However, and in order to be able to utilize batch learning, keras requires all data within the same batch to have the same length or dimension. Whereas in our text data, each example could have a variable sentence length. In order to overcome this issue, we will go over all of our data, and calculate the length of the longest phrase (in terms of words). Then, we will 0-pad all of the data sequences so that they will all have the same *'max_len'* calculated.

```
Removed Noise, CPU Time: 0.0004973411560058594
max_len: 25
Data Prepared for model, CPU Time: 3.57861328125
[[226278. 169725.  74390.      0.      0.      0.      0.      0.      0.
       0.      0.      0.      0.      0.      0.      0.      0.      0.
       0.      0.      0.      0.      0.      0.      0.]
 [239785. 269953. 372306. 361859.  91041. 382320. 161844. 352214.      0.
       0.      0.      0.      0.      0.      0.      0.      0.      0.
       0.      0.      0.      0.      0.      0.      0.]
 [357161. 368306.  46173. 372306. 160418. 239785. 179025. 329974.  58999.
  349437.      0.      0.      0.      0.      0.      0.      0.      0.
       0.      0.      0.      0.      0.      0.      0.]
 [330826. 302352.  97698. 184322. 251645. 132701. 302292. 151204. 286963.
  154049. 231458. 338210.      0.      0.      0.      0.      0.      0.
       0.      0.      0.      0.      0.      0.      0.]
 [133896. 141948.      0.      0.      0.      0.      0.      0.      0.
       0.      0.      0.      0.      0.      0.      0.      0.      0.
       0.      0.      0.      0.      0.      0.      0.]]
[1. 1. 1. 1. 1.]
```

The first picture is the input of model LSTM after many steps GloVe/ Padding

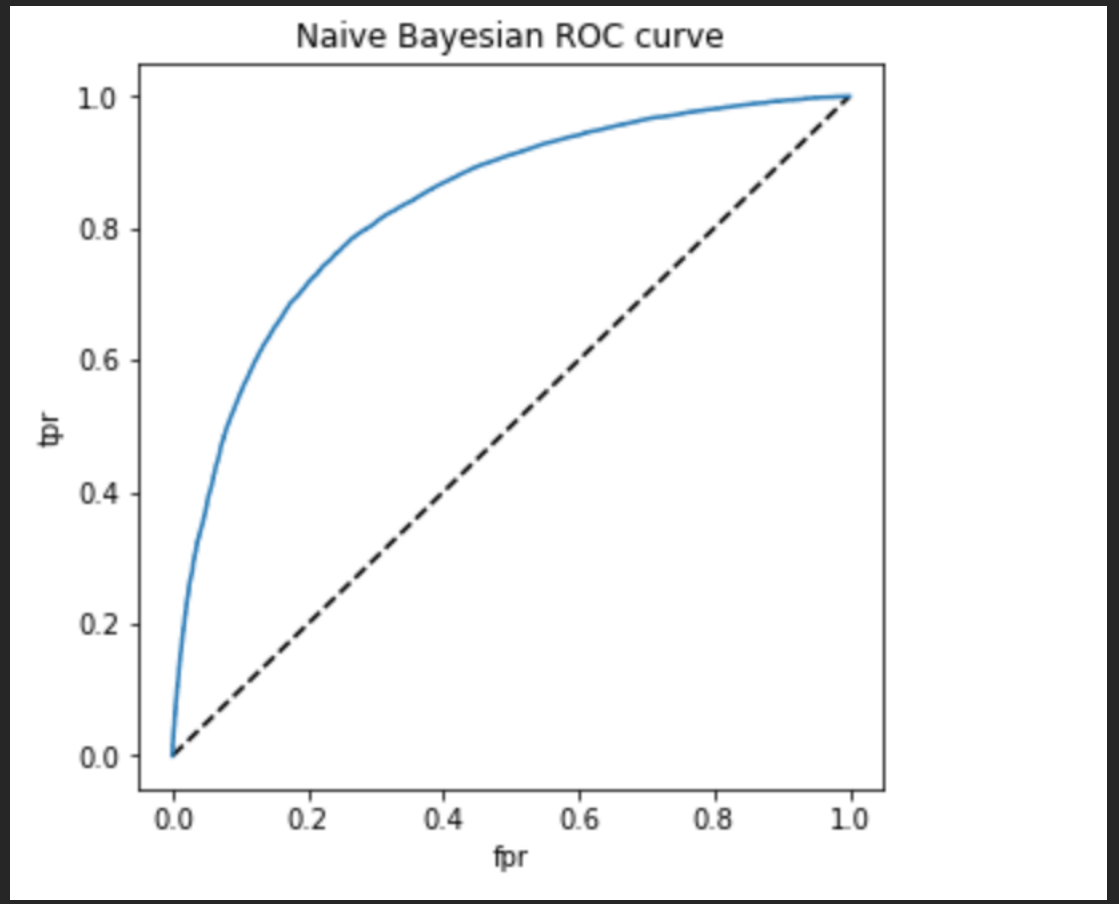
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 25, 50)            20000050
_________________________________________________________________
bidirectional (Bidirectional (None, 25, 256)           183296
_________________________________________________________________
bidirectional_1 (Bidirection (None, 256)               394240
_________________________________________________________________
dense (Dense)                (None, 1)                 257
=================================================================
Total params: 20,577,843
Trainable params: 577,793
Non-trainable params: 20,000,050
_________________________________________________________________
```

```
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs = 20, batch_size
= 128, shuffle=True)
```

```
Epoch 1/20
2500/2500 [==============================] - 33s 13ms/step - loss: 0.5516 - accurac
y: 0.7125 - val_loss: 0.5210 - val_accuracy: 0.7406
Epoch 2/20
2500/2500 [==============================] - 32s 13ms/step - loss: 0.5070 - accurac
y: 0.7458 - val_loss: 0.4987 - val_accuracy: 0.7535
Epoch 3/20
2500/2500 [==============================] - 32s 13ms/step - loss: 0.4851 - accurac
y: 0.7611 - val_loss: 0.4869 - val_accuracy: 0.7627
Epoch 4/20
2500/2500 [==============================] - 32s 13ms/step - loss: 0.4679 - accurac
y: 0.7717 - val_loss: 0.4863 - val_accuracy: 0.7636
Epoch 5/20
2500/2500 [==============================] - 31s 12ms/step - loss: 0.4496 - accurac
y: 0.7834 - val_loss: 0.4817 - val_accuracy: 0.7670
Epoch 6/20
2500/2500 [==============================] - 32s 13ms/step - loss: 0.4293 - accurac
y: 0.7960 - val_loss: 0.4899 - val_accuracy: 0.7642
Epoch 7/20
2500/2500 [==============================] - 31s 12ms/step - loss: 0.4040 - accurac
y: 0.8104 - val_loss: 0.5057 - val_accuracy: 0.7630
Epoch 8/20
```

Naïve Bayesian

```
start_time = time()

from nltk import classify
from nltk import NaiveBayesClassifier
classifier = NaiveBayesClassifier.train(train_data)

# Output the model accuracy on the train and test data
print('Accuracy on train data:', classify.accuracy(classifier, train_data))
print('Accuracy on test data:', classify.accuracy(classifier, test_data))

# Output the words that provide the most information about the sentiment of a tweet.
# These are words that are heavily present in one sentiment group and very rarely presen
t in the other group.
print(classifier.show_most_informative_features(20))

print('\nCPU Time:', time() - start_time)
```



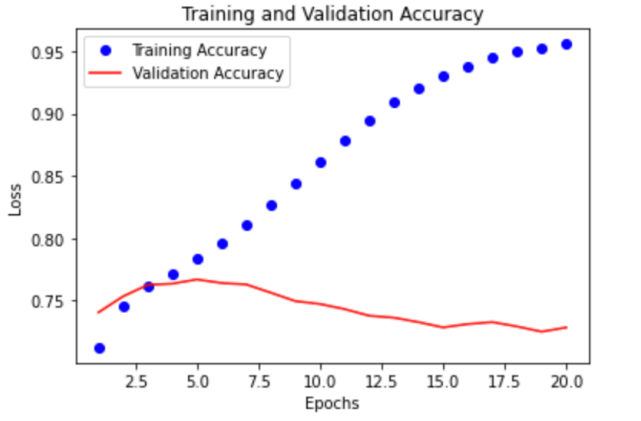
Predicted	0	1	All
Actual			
0	16528	3452	19980
1	6319	13701	20020
All	22847	17153	40000

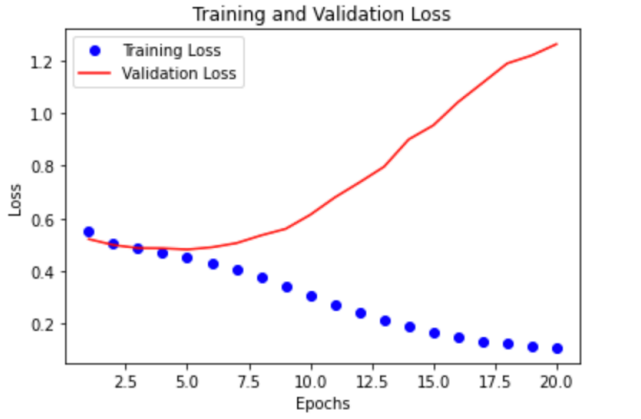
LSTM Models

```
# Defining a function that will initialize and populate our embedding layer

def pretrained_embedding_layer(word_to_vec_map, word_to_index, max_len):
    vocab_len = len(word_to_index) + 1
    emb_dim = word_to_vec_map["unk"].shape[0] #50

    emb_matrix = np.zeros((vocab_len, emb_dim))

    for word, idx in word_to_index.items():
        emb_matrix[idx, :] = word_to_vec_map[word]

    embedding_layer = Embedding(vocab_len, emb_dim, trainable=False, input_shape=(max_
len,))
    embedding_layer.build((None,))
    embedding_layer.set_weights([emb_matrix])

    return embedding_layer
```

```
# Defining a sequencial model composed of firstly the embedding layer, than a pair of Bi
directional LSTMs,
# that finally feed into a sigmoid layer that generates our desired output betwene 0 and
1.

model = Sequential()

model.add(pretrained_embedding_layer(word_to_vec_map, word_to_index, max_len))
model.add(Bidirectional(LSTM(units=128, return_sequences=True)))
model.add(Bidirectional(LSTM(units=128, return_sequences=False)))
model.add(Dense(units=1, activation='sigmoid'))

model.summary()
```

LSTM Models





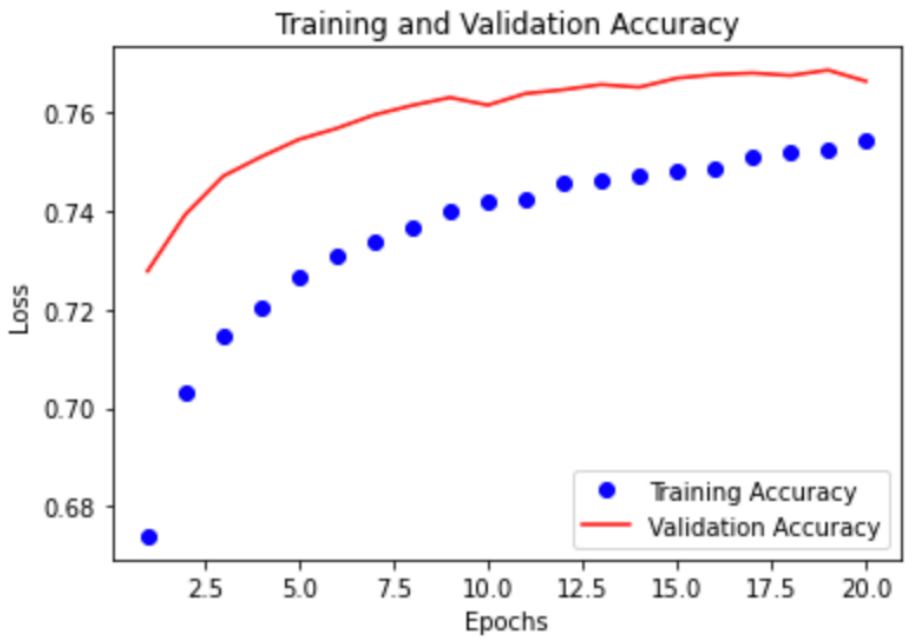
The training accuracy is rocketing, exceeding 95% after 20 epochs while the validation accuracy increased slightly in few early epochs (reaching 76.7%) then consistantly gradual decrease.

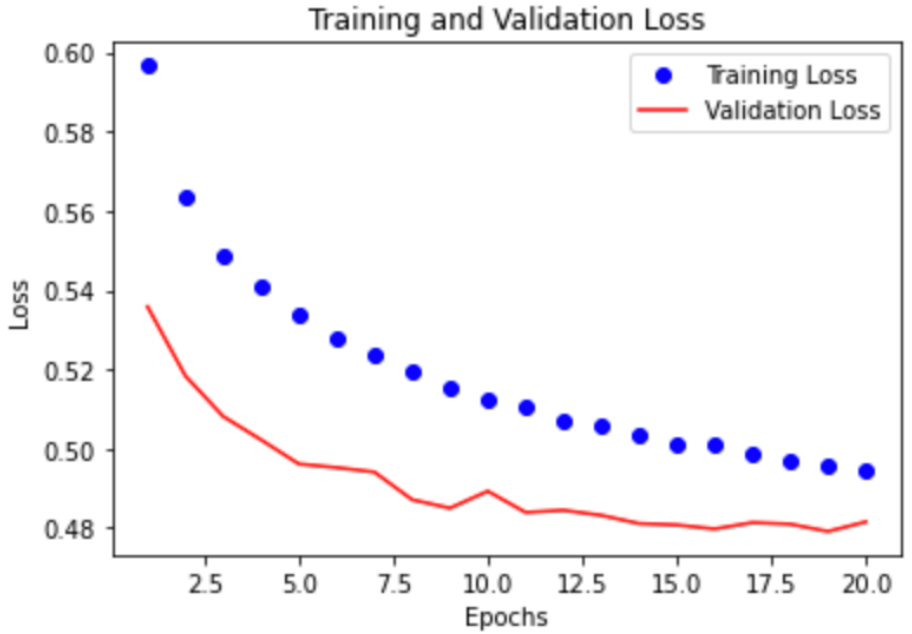
From those signs, it show that this model receiving high variance and low bias which mean ‘overfitting’.

Regularization – Drop Out LSTM

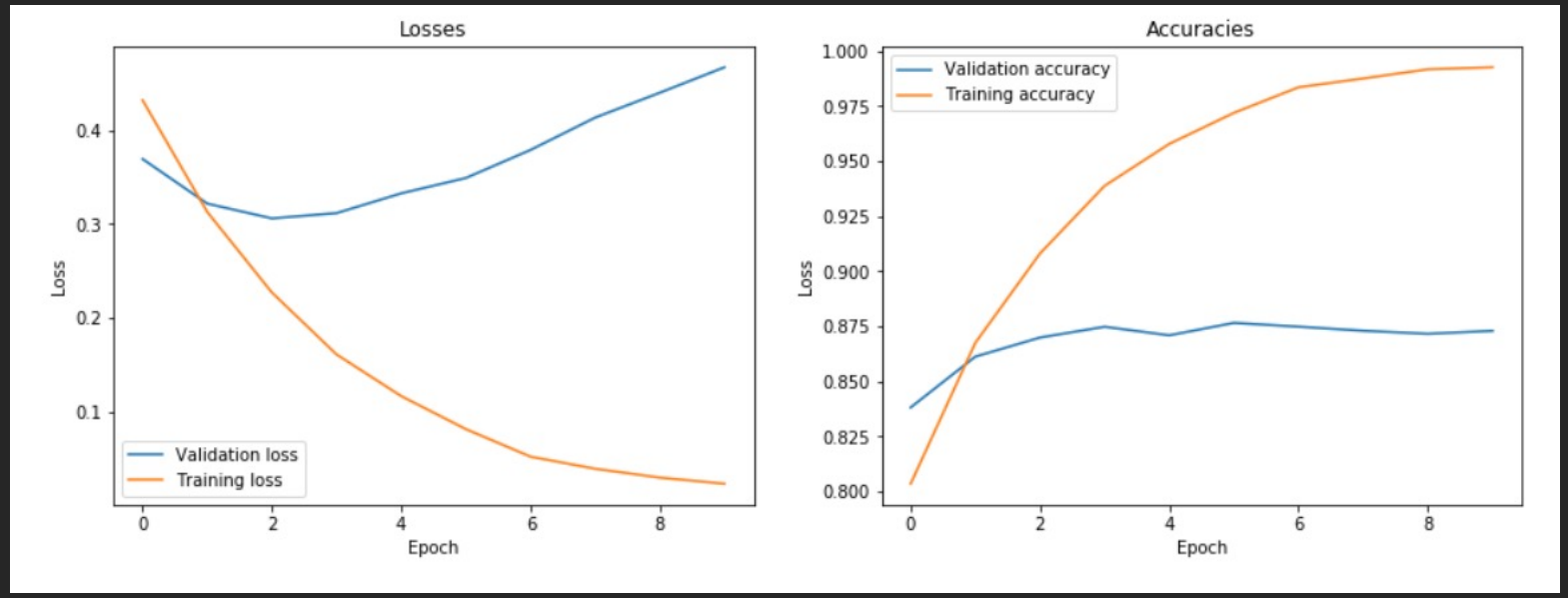
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 25, 50)            20000050
_________________________________________________________________
dropout (Dropout)            (None, 25, 50)            0
_________________________________________________________________
bidirectional (Bidirectional (None, 25, 256)           183296
_________________________________________________________________
dropout_1 (Dropout)          (None, 25, 256)           0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 256)               394240
_________________________________________________________________
dense (Dense)                (None, 1)                 257
=================================================================
Total params: 20,577,843
Trainable params: 577,793
Non-trainable params: 20,000,050
_________________________________________________________________
```

The model now can erase the over fitting phenomeno. But the accuracy was not increase. This is why I need to investigate on the dataset.





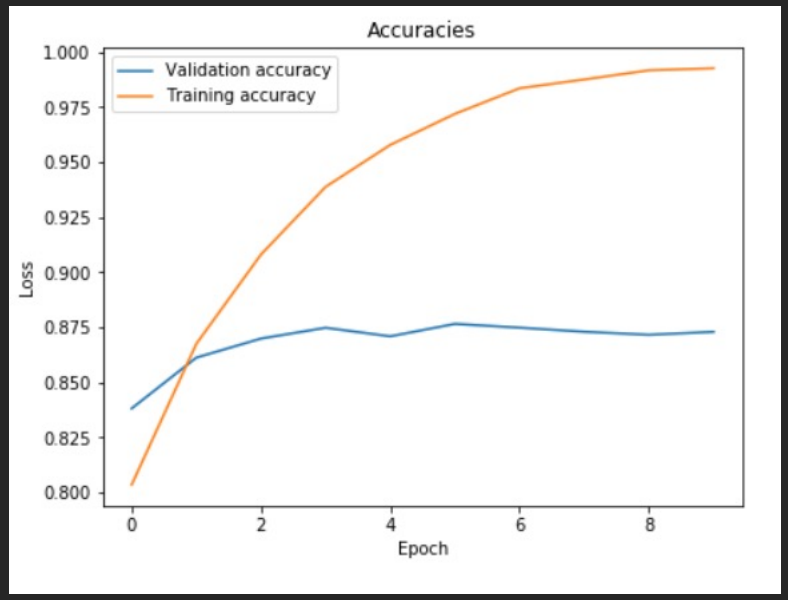
CNN Models



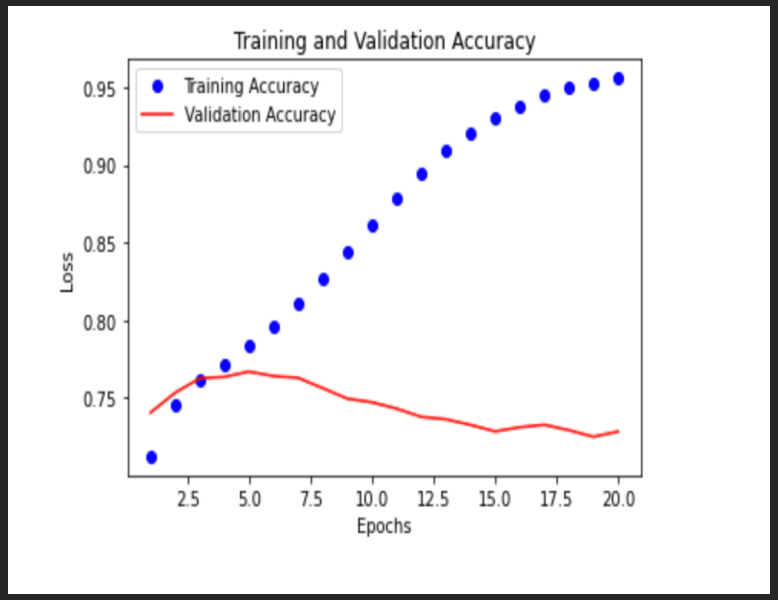
CNN model gives a high accuracy at around 85% atfter 8epochs but it receives a sign of overfitting since training accuracy was higher than validation accuracy

Results

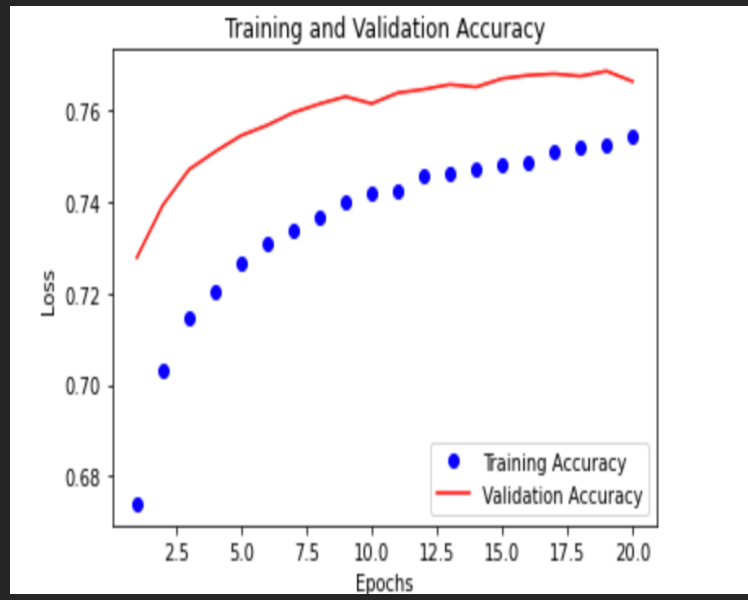
CNN Models



LSTM Models



Drop Out LSTM



I choose CNN models since high accuracy

Part 3: Investigation of improving models

Improve Model

Regularization – Drop Out

This method randomly drop out several neuron connections in the network, rendering the model less complex and forcing the model only look at part of a given example.

Inspecting the Data – Unknown Words

This method require us to track back the unknown words and find way to deal with them. Since the first LSTM model did not handle with Unk word. Let try figure out what are they.

Inspecting the Data – Unknown Words

```
[("i'm", 32149),
 ("can't", 11369),
 ("i'l", 6283),
 ("that's", 5478),
 ("i've", 5085),
 ("he's", 1976),
 ("mother's", 1878),
 ("i'd", 1855),
 ('hahaha', 1722),
 ("we're", 1578),
 ("there's", 1425),
 ("what's", 1356),
 ("they're", 1179),
 ('lmao', 1104),
 ("we'l", 914),
 ("let's", 802),
 ('bday', 618),
 ('. .', 600),
 ('idk', 588),
 ("it'l", 587),
 ('hahah', 486),
 ('. . .', 481),
 ("how's", 440),
 ("who's", 378),
 ('#fb', 358),
 ('hahahaha', 356),
```

Tracking back the unk word, I found that those are word that LSTM model could not handle.

x

To solve this problem, I need a custom function to redefine those unknown words.

After cleaning those word, the model gave a better result. (Picture on the right). The validation reach 81.1% on the 5th epoch.