Capstone Project

Ashish Kumar Verma

11-April-2019

Title: Sentiment Analysis

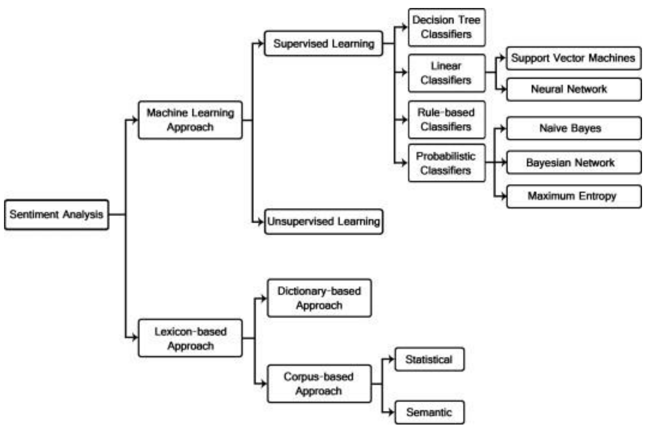
Definition

Project Overview

Sentiment Analysis is contextual mining of text which identifies and extracts subjective information in the source material and helping a business to understand the social sentiment of their brand, product or service while monitoring online conversations.

It is the most common text classification tool that analyses an incoming message and tells Whether the underlying sentiment is positive or negative.
Sentiment analysis systems allow companies to make sense of this sea of unstructured text by automating business processes, getting actionable insights, and saving hours of manual data processing, in other words, by making teams more efficient.

Techniques used for solving this problem



Neural Network and Deep Learning have attained state of the art results in the field of finding insights and proved successful in attaining better results than Simple Machine Learning approaches. Therefore expecting a good performance and robustness on this problem from models which I am going to use.

There is a lot of interesting academic research which is being done as well as have been published on Sentiment Analysis.

Some of them are

1. Multi-Task Deep Neural Networks for Natural Language Understanding 2. Disconnected Recurrent Neural Networks for Text Categorization

2. Problem Statement

The objective of the task is to understand the underlying sentiment of the reviews and classify them as positive or negative.

Sentiment Analysis can be used as a building block in several applications such as :

1. Computing customer satisfaction metrics:- One can get an idea of how happy customers are with your products from the ratio of positive to negative reviews about them.
2. Identifying detractors and promoters:- It can be used for customer service, by spotting dissatisfaction or problems with products.

The problem of automatically identifying underlying sentiment is difficult because of the nearly infinite number of permutations of words, positions, phrases and so on. It's a really hard problem. This is a well-studied problem in Natural Language Processing and more recently an important demonstration of the capability of deep learning.

To Tackle this problem, I have used three models :

1. Neural Bag Of Words + MLP Model
2. Embedding + 1D CNN Model.
3. N-Gram CNN Model (using multiple parallel convolutional neural networks)

Metrics

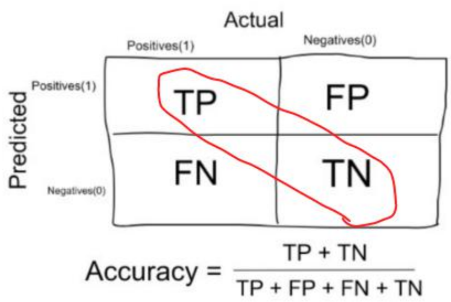
Accuracy Score -

Accuracy is the ratio of the number of correct predictions on the total number of input samples.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

.

Accuracy in classification problems is the number of correct predictions made by the model over all kinds of predictions made



.

Accuracy Score is the optimal choice of metrics for this data-set as well as for my model as the target variable classes in my data are balanced.

ie. 50 % classes in our data-set are positive reviews and rest 50 % are negative reviews.

Analysis

Data Exploration

The [Movie Review Data]() is a collection of movie reviews retrieved from the imdb.com website in the early 2000s by Bo Pang and Lillian Lee. The reviews were collected and made available as part of their research on natural language processing. The reviews were originally released in 2002, but an updated and cleaned up version was released in 2004, referred to as v2.0. The dataset is comprised of 1,000 positive and 1,000 negative movie reviews drawn from an archive of the rec.arts.movies.reviews newsgroup hosted at IMDB.

Data Files:

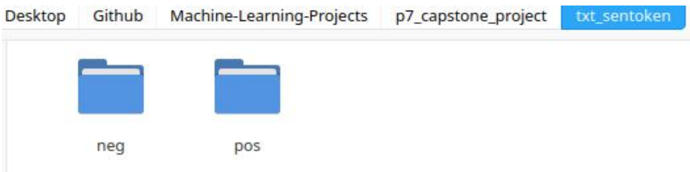
Dataset has a directory called **txt sentoken** with two subdirectories containing the text **neg** and **pos** for negative and positive reviews. Reviews are stored one per file with a naming convention from **cv000** to **cv999** for each of neg and pos.

Total Positive Files =1000

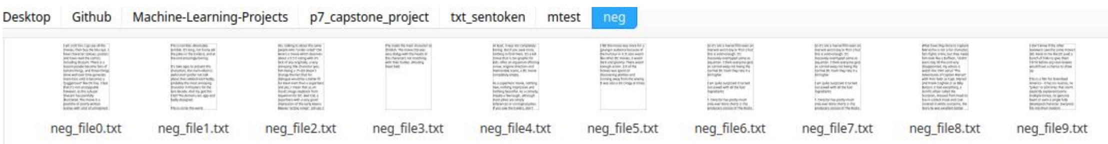
Total Negative Files =1000

Train Files From each folder contains file from **cv000 to cv899.**

Test Files From each folder contains file from **cv899 to cv999**



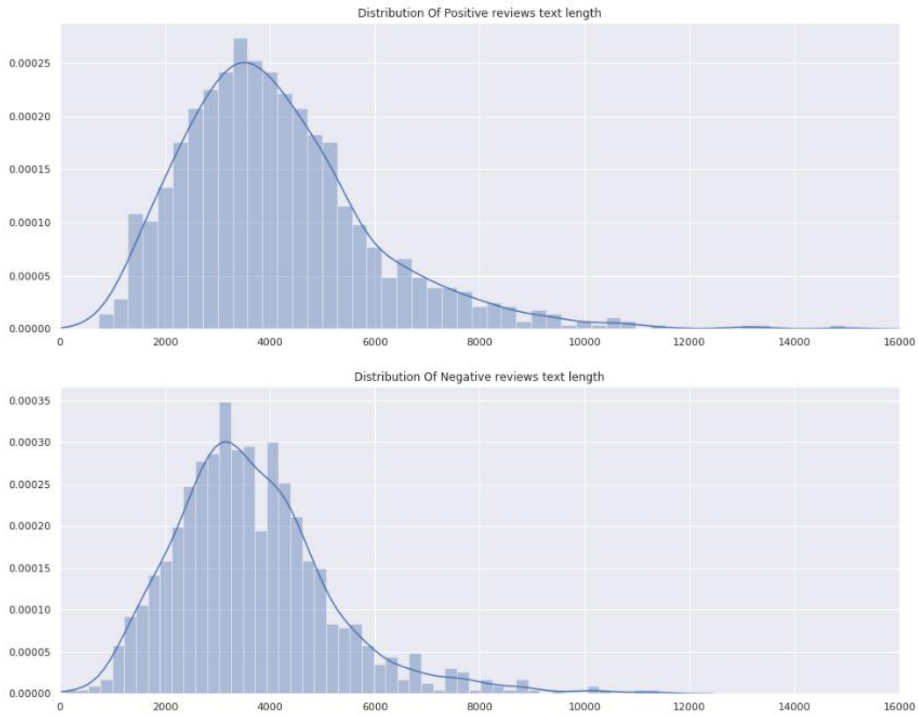
I have scrapped 10 positive as well as 10 negative reviews of movies from IMDB.com and kept them in "mtest" folder for the testing model on unseen data.



Each text file contains a negative or positive review of a movie.

Exploratory Visualization.

Distribution of Reviews Text Length

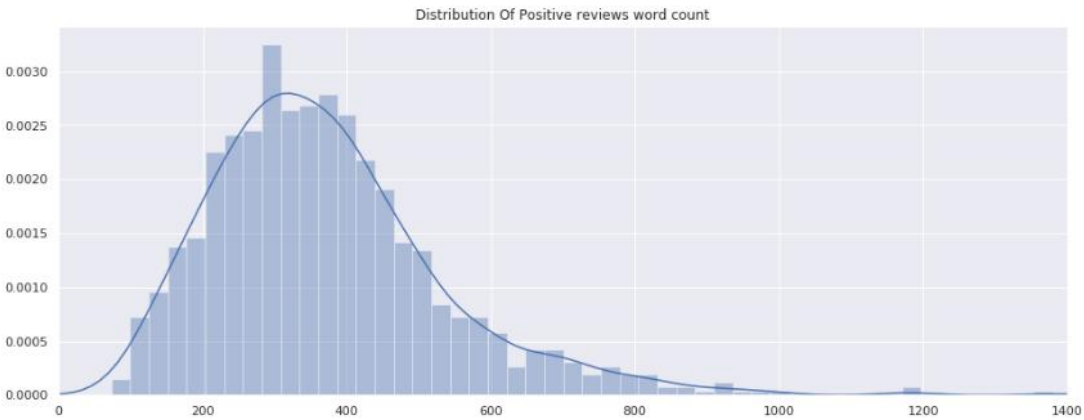


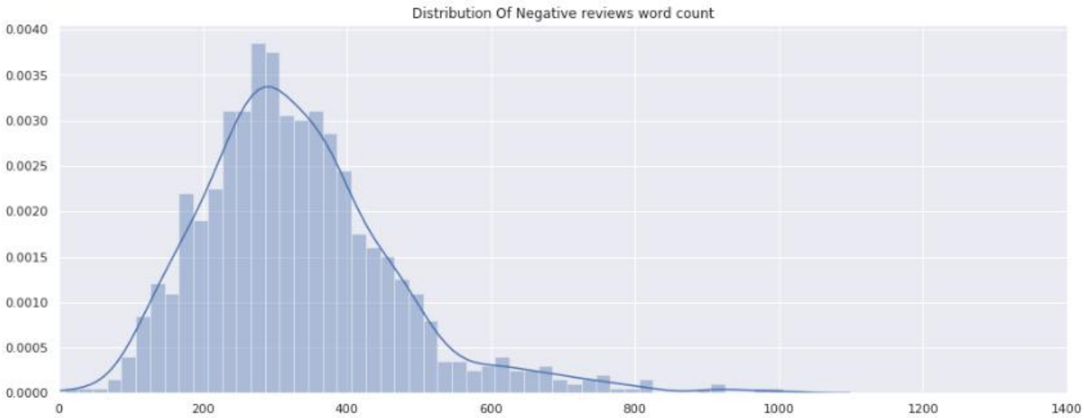
The length of most of the reviews is usually between 2000 - 6000 characters.

Length of both positive as well as negative reviews is normally distributed.

There are few reviews containing more than 8000 characters in the data-set.

Distribution of Reviews word count after Data Cleaning.



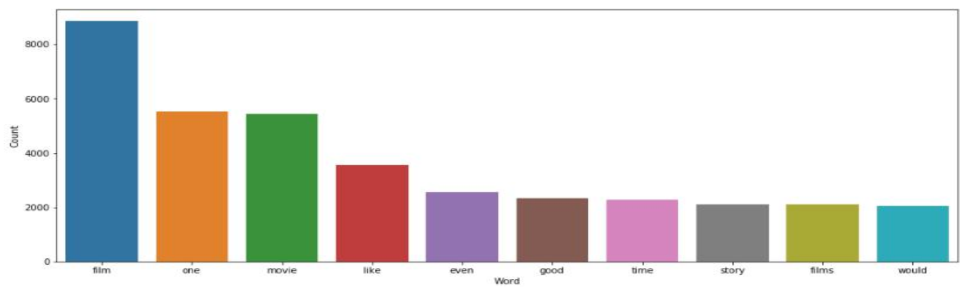


After cleaning the length of the reviews (No of words in the reviews) has dropped significantly which will be encoded and passed to model for training or testing.
(**Data Cleaning** steps has been discussed in detail later in this report.)

Top Ten Most Common Words

```
vocabmcdf=pd.DataFrame(data=vocab.most_common(10),columns=['Word','Count'])
plt.figure(figsize=(16,6))
sns.barplot(x='Word',y="Count",data=vocabmcdf)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbb5760b198>
```



I have divided the data-set into a 90-10 % ratio. I have used 900 files from both neg(negative) as well as a pos(positive) folder for training the model and rest 100 files from both folders for testing accuracy of the model.

I have explained this process in detail in data preprocessing section.

Algorithms And Techniques

For this problem, I will basically use 3 different techniques:

Benchmark Models.

Neural Bag Of Words + MLP Model

In this model, I have used Neural Bag of Words(used Keras Tokenizer)and then used a simple Multilayer Perceptron Model to predict sentiment of encoded reviews.

I have used a Sequential Model with an input layer, single hidden layer with 25 neurons and rectified linear activation function.Output layer has a single neuron with a sigmoid activation function.

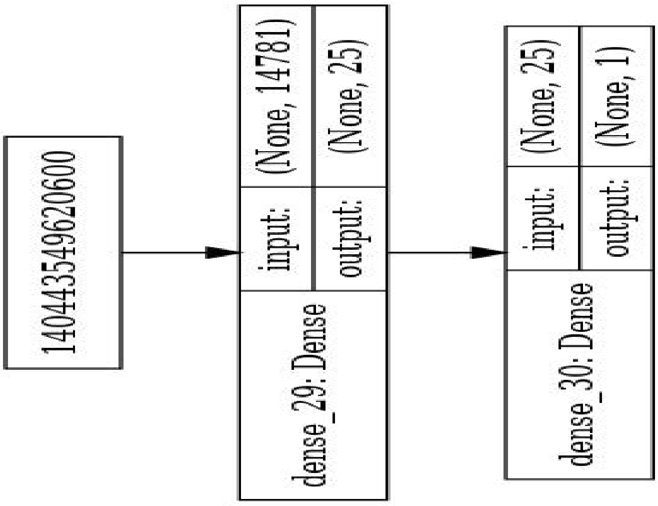
I have used “Adam” optimizer with binary cross entropy loss function.

Working:
Bag Of Words(BOW) is a feature extraction technique from the text so that it can be used with the models. (such as machine learning algorithms).BOW discards any information about the order and structure of the words in the document. The intuition here is that if two documents have similar words content then the two documents are similar.

Ex - John likes to watch movies. Mary likes movies too.
BOW vector [1, 2, 1, 1, 2, 1, 1, 0, 0, 0]

Bow vector is then passed to the MLP model where weights for the equation of the model is adjusted through feedforward and backward propagation technique to fit the data and minimize the loss.

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 25)                369550
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 26
=================================================================
Total params: 369,576
Trainable params: 369,576
Non-trainable params: 0
```




```python
# define the model
def define_model(n_words):
    # define network
    model = Sequential()
    model.add(Dense(25, input_shape=(n_words,), activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # compile network
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    # summarize defined model
    model.summary()
    plot_model(model, to_file='model.png', show_shapes=True)
    return model
```

```python
#fit network
model=define_model(n_words)
model.fit(Xtrain, ytrain, epochs=10, verbose=1)
```

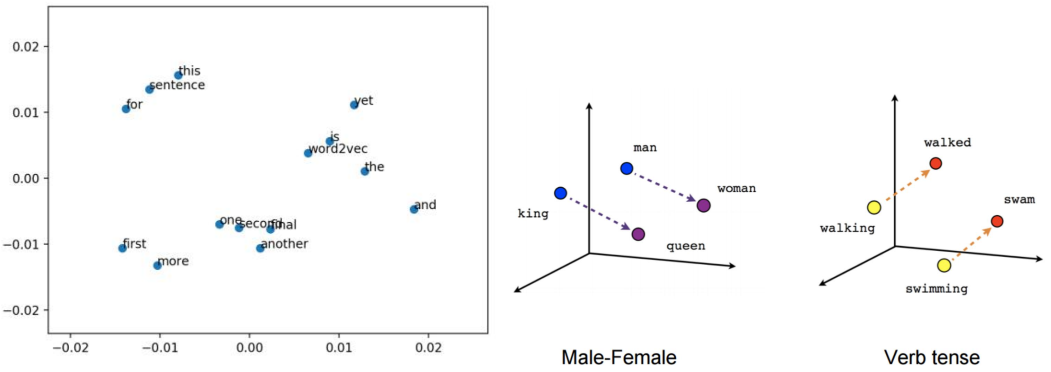
Embedding + 1D CNN Model.

In this Model, First I have encoded the movie reviews to a sequence of integers and then used the model as described including Embedding layer. (CNN) as they have proven to be successful at document classification problems. I have used CNN layer with a kernel size of 8 with “relu” activation function, followed by a pooling layer which reduces the output of the CNN layer by half. Next, the 2D output from the CNN layer is flattened to one long 2D vector to represent the features extracted by the CNN. The back-end of the model is a standard Multilayer Perceptron layers to interpret the CNN features. The output layer is a sigmoid activation function to output a value between 0 and 1.I have used “Adam” optimizer with binary cross entropy loss function.

Working:

Embedding:

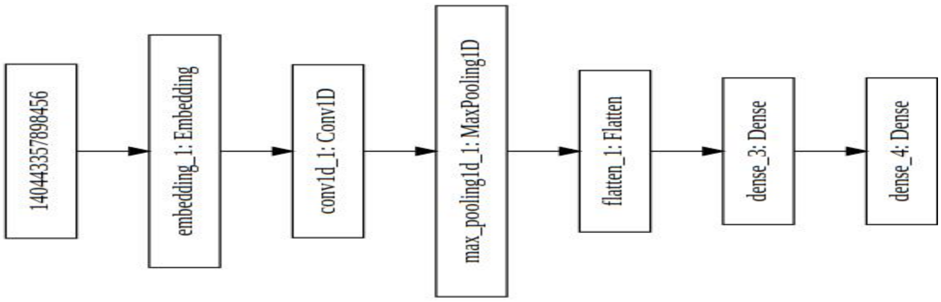
Word embeddings are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network.it is a distributed representation of words where different words that have a similar meaning (based on their usage) also have similar representation. This is a more expressive representation for text than more classical methods like bag-of-words.



CNN - The vector from the embedding layer is passed to CNN layer which learns the salient feature from the represented vector.

Fully Connected Layer: Output from the CNN layer is passed to this for interpretation of extracted features in terms of the output required.

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 1244, 100)         1478100
conv1d_5 (Conv1D)            (None, 1237, 32)          25632
max_pooling1d_5 (MaxPooling1 (None, 618, 32)           0
flatten_5 (Flatten)          (None, 19776)             0
dense_13 (Dense)             (None, 10)                197770
dense_14 (Dense)             (None, 1)                 11
=================================================================
Total params: 1,701,513
Trainable params: 1,701,513
Non-trainable params: 0
```



```
def define_embed_cnn_model(vocab_size, max_length):
    model = Sequential()
    model.add(Embedding(vocab_size, 100, input_length=max_length))
    model.add(Conv1D(filters=32, kernel_size=8, activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # compile network
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

#     filepath="weights.best.hdf5"
#     checkpoint = ModelCheckpoint(filepath, monitor= 'val_acc' , verbose=1, save_best_on
e,
#     mode= max )
#     callbacks_list = [checkpoint]
    # summarize defined model
    model.summary()
    plot_model(model, to_file='cnn_model.png', show_shapes=True)
#     return model,callbacks_list
    return model
```

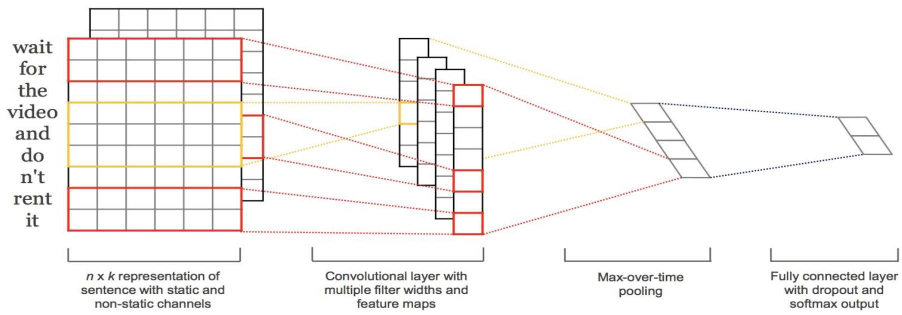
N-Gram CNN Model

In this model, I have developed a multichannel CNN. I have chosen to work also with this model as this allows the document to be processed at different n-grams (groups of words) at a time. Each channel consists of an embedding layer as input, followed by a 1D CNN layer, dropout Layer to prevent overfitting, pooling layer and then a prediction output layer.

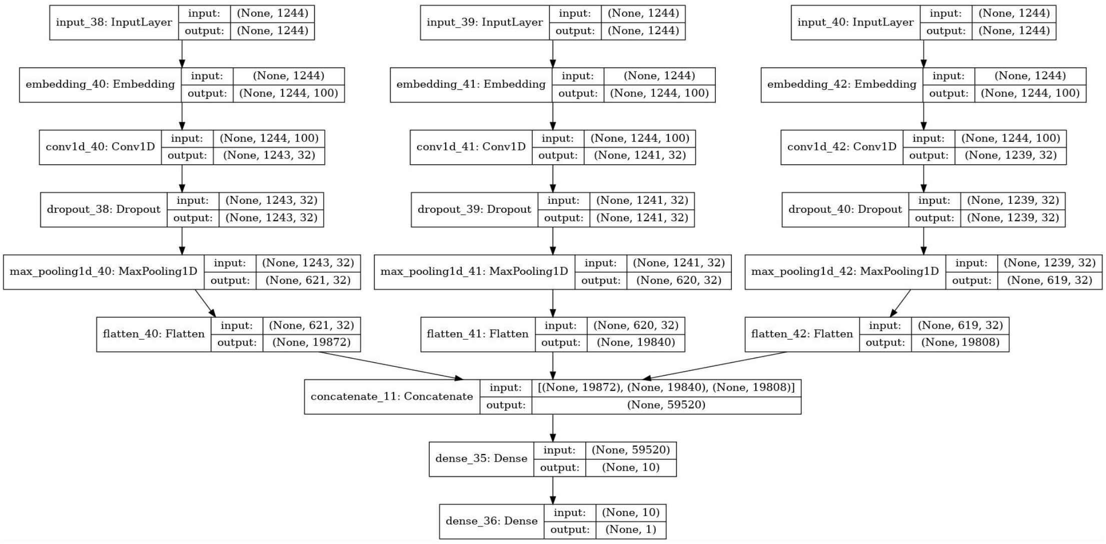
I have used "Adam" optimizer with binary cross entropy loss function.

Working:

In N-Gram model standard 1D CNN model is merged together in order to get the better result in capturing the salient features from the input.



.




```python
def define_ncnn_model(length, vocab_size):
    # channel 1
    inputs1 = Input(shape=(length,))
    embedding1 = Embedding(vocab_size, 100)(inputs1)
    conv1 = Conv1D(filters=32, kernel_size=2, activation='relu')(embedding1)
    drop1 = Dropout(0.5)(conv1)
    pool1 = MaxPooling1D(pool_size=2)(drop1)
    flat1 = Flatten()(pool1)
    # channel 2
    inputs2 = Input(shape=(length,))
    embedding2 = Embedding(vocab_size, 100)(inputs2)
    conv2 = Conv1D(filters=32, kernel_size=4, activation='relu')(embedding2)
    drop2 = Dropout(0.5)(conv2)
    pool2 = MaxPooling1D(pool_size=2)(drop2)
    flat2 = Flatten()(pool2)
    # channel 3
    inputs3 = Input(shape=(length,))
    embedding3 = Embedding(vocab_size, 100)(inputs3)
    conv3 = Conv1D(filters=32, kernel_size=6, activation='relu')(embedding3)
    drop3 = Dropout(0.5)(conv3)
    pool3 = MaxPooling1D(pool_size=2)(drop3)
    flat3 = Flatten()(pool3)
#     # channel 3
#     inputs4 = Input(shape=(length,))
#     embedding4 = Embedding(vocab_size, 100)(inputs4)
#     conv4 = Conv1D(filters=32, kernel_size=8, activation='relu')(embedding4)
#     drop4 = Dropout(0.5)(conv4)
#     pool4 = MaxPooling1D(pool_size=2)(drop4)
#     flat4 = Flatten()(pool4)
    # merge
    merged = concatenate([flat1, flat2, flat3])
    # interpretation
    dense1 = Dense(10, activation='relu')(merged)
    outputs = Dense(1, activation='sigmoid')(dense1)
    model = Model(inputs=[inputs1, inputs2, inputs3], outputs=outputs)
    # compile
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    # summarize
    model.summary()
    plot_model(model, show_shapes=True, to_file='ncnn_model.png')
    return model
```

```python
ncnn_model= define_ncnn_model(length, vocab_size)
```

Methodology

Data Preprocessing.

Looking at the text in the files of the dataset.We need to preprocess it in order to make it work with our model to successfully give sentiment analysis of the reviews.

My data preprocessing Steps:

1. Data Cleaning
2. Developing a vocabulary.

Data Cleaning

Splitting token into white space. (Tokenization)
Removing all punctuation from words.
Removing all words that are not purely comprised of alphabetical characters.
Removing all words that are known stop words.
Removing all words that have a length <= 1 character.

```
def clean_document(document,m_type="mlp"):
    document=document.lower()
    #split the review into tokens by white space
    tokens=document.split()
    # regex for char filtering
    re_punc = re.compile('[%s]' % re.escape(string.punctuation))
    # remove punctuation from each word
    tokens = [re_punc.sub('', w) for w in tokens]
    # removetokens which are not alphabetis
    if m_type=="mlp":
        tokens = [word for word in tokens if word.isalpha()]
        # remove stop words
        ##  A stop word is a commonly used word (such as "the", "a", "an", "in")
        stop_words = set(stopwords.words('english'))
        tokens = [w for w in tokens if not w in stop_words]
        # remove out short tokens
        tokens = [word for word in tokens if len(word) > 1]

    return tokens
```

```
filename="txt_sentoken/pos/cv001_18431.txt"
text=load_document(filename)
tokens=clean_document(text)
print(tokens)
```

```
['every', 'movie', 'comes', 'along', 'suspect', 'studio', 'every', 'indication', 'stinker', 'ever
ybodys', 'surprise', 'perhaps', 'even', 'studio', 'film', 'becomes', 'critical', 'darling', 'mt
v', 'films', 'election', 'high', 'school', 'comedy', 'starring', 'matthew', 'broderick', 'reese',
'witherspoon', 'current', 'example', 'anybody', 'know', 'film', 'existed', 'week', 'opened', 'plo
t', 'deceptively', 'simple', 'george', 'washington', 'carver', 'high', 'school', 'student', 'elec
tions', 'tracy', 'flick', 'reese', 'witherspoon', 'overachiever', 'hand', 'raised', 'nearly', 'ev
ery', 'question', 'way', 'way', 'high', 'mr', 'matthew', 'broderick', 'sick', 'megalomaniac', 'st
udent', 'encourages', 'paul', 'popularbutslow', 'jock', 'run', 'pauls', 'nihilistic', 'sister',
```

....

Which on passing a text file returns a list of tokens which is further executed on these two lines

```
tokens=[w for w in tokens if w in vocab]
line=' '.join(tokens)
```

Line obtained is encoded (on the basis of the model requirement) and further passed to model for analysis.

Example -

Review: [Give me a break. Top 200 movie of all time? Not even close. The bad guy in the movie was one of the worst characters I ever seen. It just was not a very good flick. It tried to build up the love between Peter Parker and the girl and then all of a sudden, he just can't be with her? Please. This movie will become a cult movie and will get good rating because people will be afraid to speak the truth, which was, this movie wasnt very good.However, I feel that the sequel might be better because they dont have to build up the character so much..]

After **Data cleaning**,

[give break top movie time even close bad guy movie one worst character ever seen good flick tried build love peter parker girl sudden cant please movie become cult movie get good rating people afraid speak truth movie wasn't feel sequel might better dont build character much]

Then this line is passed to the model.

Developing a Vocabulary :

Development of a Vocabulary is also needed when working with a predictive model of text, in order to reduce the size as more words will have a larger representation of documents. Therefore it is important to constrain the words to only those believed to be predictive.

```
vocab = Counter()

def develop_vocab():
    global vocab
#     vocab= Counter()
    process_documents('txt_sentoken/neg', vocab)
    process_documents('txt_sentoken/pos', vocab)


    min_occur = 5

    tokens = [k for k,c in vocab.items() if c >= min_occur]
    save_list(tokens,"vocab.txt")

develop_vocab()
```

```
#print Length of the Vocabulary
print("Total Length Of The Vocabulary %s" %len(vocab))
```

```
Total Length Of The Vocabulary 46557
```

```
# 50 Most Common Words
print(vocab.most_common(50))
```

```
[('film', 8860), ('one', 5521), ('movie', 5440), ('like', 3553), ('even', 2555), ('good', 2320),
('time', 2283), ('story', 2118), ('films', 2102), ('would', 2042), ('much', 2024), ('also', 196
5), ('characters', 1947), ('get', 1921), ('character', 1906), ('two', 1825), ('first', 1768), ('s
ee', 1730), ('well', 1694), ('way', 1668), ('make', 1590), ('really', 1563), ('little', 1491),
('life', 1472), ('plot', 1451), ('people', 1420), ('movies', 1416), ('could', 1395), ('bad', 137
4), ('scene', 1373), ('never', 1364), ('best', 1301), ('new', 1277), ('many', 1268), ('doesnt', 1
267), ('man', 1266), ('scenes', 1265), ('dont', 1210), ('know', 1207), ('hes', 1150), ('great', 1
141), ('another', 1111), ('love', 1089), ('action', 1078), ('go', 1075), ('us', 1065), ('directo
r', 1056), ('something', 1048), ('end', 1047), ('still', 1038)]
```

After extracting all the words in the Counter Class and saving it in a file. We have a total number of words 14803.

```
#Load The Vocabulary from Vocab.txt File.

vocab_data=load_document("vocab.txt")
vocab_data=vocab_data.split()
vocab=set(vocab_data)
print("Number of Words in Vocab.txt is %s" %len(vocab))
```

```
Number of Words in Vocab.txt is 14803
```

Implementation

As stated earlier, I have created 3 models with dataset split in the ratio 90/10 % ratio. After that, as explained in the Data Preprocessing section, performed an operation on the raw reviews and then encoded(using tokenizer api of keras) the processed reviews and trained all the models with the same.

Neural Bag Of Words + MLP Model

```
#fit network
model=define_model(n_words)
model.fit(Xtrain, ytrain, epochs=10, verbose=1)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 25)                369550
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 26
=================================================================
Total params: 369,576
Trainable params: 369,576
Non-trainable params: 0
```

Embedding + 1D CNN Model.

```
embed_cnn_model.fit(cnn_Xtrain, ytrain, epochs=10, verbose=1)

# history=embed_cnn_model.fit(cnn_Xtrain, ytrain,validation_split=0.20 ,epochs=10, verbose=0,bat
ch_size=100,callbacks=callb)
```

N-Gram CNN Model

```
ncnn_model.fit( [ncnn_Xtrain,ncnn_Xtrain,ncnn_Xtrain,ncnn_Xtrain] , ytrain, epochs=7, batch_size
=16)
```

Complications

I would say, deciding data cleaning steps for a different model (since Data Cleaning of MLP differs from CNN/NCNN model in my implementation) as well as experimenting with kernel size, no of filters for CNN layer and implementing multiple channels in N-gram is where I took time.

Though, I didn't face any difficulty in implementing logic for the problem.

Refinement.

I experimented with my model quite a bit. I decreased the number of words in vocabulary.txt which resulted in better performance in **Embedding + 1D CNN Model** as well as **N-Gram CNN Model.**

I tried for different no of epochs, larger no of epochs led to a worse performance for the **N-Gram CNN Model.** as well as **Embedding + 1D CNN Model.**

In N-Gram CNN Model, I experimented with different no of channels such as 2,3,4,5,6 with different kernel size for 1D CNN configuration but I got better results in the configuration as reported above.

Results

Model Evaluation and Validation.

As stated in the previous section, I experimented with my model by increasing/decreasing its epochs as well as changing no of channels in N-gram model, changing optimizers, including dropouts, pooling layer in my model. In each model, I got a slightly better result than the previous one.

Models Used In this report performs well in aspects of generalizing the unseen data, These tests prove this and We can trust these models as these models perform well even since models are trained on longer reviews and expect reviews with approx 1000 characters or more.

MLP Model

```
best movie ever great recommend
Review: [Best movie ever! It was great, I recommend it.]
Sentiment: POSITIVE (82.057%)

bad movie
Review: [This is a bad movie.]
Sentiment: NEGATIVE (99.110%)

average one one time watch
Review: [An above average one for one time watch.]
Sentiment: POSITIVE (51.494%)
```

Embedding + CNN Model.

```
everyone enjoy film love recommended
0.513243
Review: [Everyone will enjoy this film. I love it, recommended!]
Sentiment: POSITIVE (51.324%)

bad movie watch sucks
0.4449024
Review: [This is a bad movie. Do not watch it. It sucks.]
Sentiment: NEGATIVE (55.510%)
```

```
loved movie movie good would recommend movie everyone
0.50684905
Review: [I loved This Movie. This Movie is too good ,I would recommend this movie to everyone.]
Sentiment: POSITIVE (50.685%)
```

N-gram CNN Model.

```
ok movie one time watch
0.5051639
Review: [It is an ok movie one time watch]
Sentiment: POSITIVE (50.516%)

loved would recommend movie loved movie goodness goodness goodness would recommend movie everyone
0.6596375
Review: [I loved This Movie.I would recommend this movie to everyone.I loved This Movie. GoodNess Goodness Goodn
ess I would recommend this movie to everyone.]
Sentiment: POSITIVE (65.964%)

bad bad bad hate movie one waste money watching
0.18181247
Review: [Bad , Bad , Bad I Hate this Movie No One Should waste money watching this]
Sentiment: NEGATIVE (81.819%)
```

Justification

```
print("\n---------Evaluating For MLP Model----------------\n")
loss, acc  = model.evaluate(Xtest, ytest, verbose=1)
print('Test Accuracy: %f' % (acc*100))

print("\n---Evaluating For Embedding + 1DCNN Model--------\n")
cnn_loss, cnn_acc  = embed_cnn_model.evaluate(cnn_Xtest, ytest, verbose=1)
print('Test Accuracy: %f' % (cnn_acc*100))

print("\n---Evaluating For N-gram CNN Model--------\n")
ncnn_loss, ncnn_acc = ncnn_model.evaluate([ncnn_Xtest,ncnn_Xtest,ncnn_Xtest], ytest, verbose=1)
print('Test Accuracy: %.2f' % (ncnn_acc*100))
```

```
---------Evaluating For MLP Model----------------

200/200 [==============================] - 0s 132us/step
Test Accuracy: 86.000000

---Evaluating For Embedding + 1DCNN Model--------

200/200 [==============================] - 1s 4ms/step
Test Accuracy: 86.500000

---Evaluating For N-gram CNN Model--------

200/200 [==============================] - 2s 8ms/step
Test Accuracy: 87.00
```

Due to the stochastic nature of neural networks, results kept varying in accuracy with a difference of 0.5-3.0 for each model, but most time the later model was better from the earlier one.

But the confidence of sentiment analysis kept getting better in each subsequent model.

Conclusion

Free Form Visualization

Analysis for Positive File.

	FILE_NAME	MLP		CNN		NCNN	
		SENTIMENT	PERCENT	SENTIMENT	PERCENT	SENTIMENT	PERCENT
0	pos_file0.txt	NEGATIVE	68.4756	NEGATIVE	60.1269	NEGATIVE	71.4369
1	pos_file1.txt	POSITIVE	63.4451	POSITIVE	89.2659	POSITIVE	57.2307
2	pos_file2.txt	POSITIVE	79.4919	POSITIVE	99.9031	POSITIVE	93.8895
3	pos_file3.txt	POSITIVE	63.6199	POSITIVE	84.136	NEGATIVE	62.6304
4	pos_file4.txt	POSITIVE	71.7437	POSITIVE	100	POSITIVE	99.9602
5	pos_file5.txt	POSITIVE	58.3309	POSITIVE	99.9953	POSITIVE	95.0724
6	pos_file6.txt	NEGATIVE	57.8446	NEGATIVE	86.5762	NEGATIVE	62.8652
7	pos_file7.txt	POSITIVE	66.0336	POSITIVE	99.6336	POSITIVE	95.6444
8	pos_file8.txt	POSITIVE	64.7007	POSITIVE	98.1397	POSITIVE	93.685
9	pos_file9.txt	POSITIVE	51.4868	POSITIVE	57.1966	POSITIVE	56.7932

Analysis for Negative File.

	FILE_NAME	MLP		CNN		NCNN	
		SENTIMENT	PERCENT	SENTIMENT	PERCENT	SENTIMENT	PERCENT
0	neg_file0.txt	NEGATIVE	69.1851	NEGATIVE	99.9608	NEGATIVE	97.4781
1	neg_file1.txt	NEGATIVE	85.5158	NEGATIVE	97.6648	NEGATIVE	96.1784
2	neg_file2.txt	NEGATIVE	79.9565	NEGATIVE	96.6374	NEGATIVE	90.2806
3	neg_file3.txt	NEGATIVE	84.9834	NEGATIVE	64.9796	NEGATIVE	70.162
4	neg_file4.txt	NEGATIVE	81.3019	NEGATIVE	99.5586	NEGATIVE	98.9984
5	neg_file5.txt	NEGATIVE	63.7474	NEGATIVE	59.4952	NEGATIVE	68.2308
6	neg_file6.txt	NEGATIVE	71.4874	NEGATIVE	95.7333	NEGATIVE	81.5003
7	neg_file7.txt	NEGATIVE	71.4874	NEGATIVE	95.7333	NEGATIVE	81.5003
8	neg_file8.txt	NEGATIVE	57.9819	NEGATIVE	57.1348	NEGATIVE	59.736
9	neg_file9.txt	NEGATIVE	67.6197	NEGATIVE	96.9229	NEGATIVE	86.7453

* Confidence % is lower in the N-gram Cnn model as it has been trained on text containing approx 1000 or more words and is expecting more or at least 1000 words.

Reflection

The following steps were taken to complete this process:

1.Downloaded Dataset From the source.

2. Performed Development of Vocabulary.
3. Performed Data Cleaning.
4. Split Dataset into Train set and Test set.
5. Designed Model Architecture.
6. Evaluated.

This was the first time I dealt with predictive text analysis problem. I have learned a lot of new concepts from this project such as Tokenization, Embedding Layers, Multichannel Models as well as steps involved in Data Cleaning Process. The most challenging part of the project was finding out the architecture of the N-Gram CNN model as well as tunning hyperparameters.

Improvement

There are few ways in which I think more accuracy can be achieved on this accuracy :

1. Exploring better Data Cleaning procedure.
2. Exploring tuning CNN Hyperparameters.
3. Deeper Network
4. Longer Test Reviews.
5. Using Pre-Trained Embedding .such as Word2Vec or Glove.
6. Using a Recurrent Neural Network.