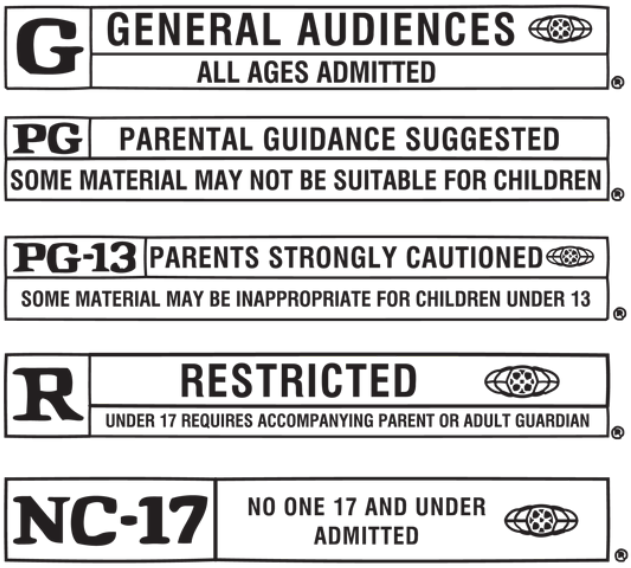
IST 664
NATURAL LANGUAGE PROCESSING

FINAL PROJECT

Kishan Rathor
SUID : 307229979

MS Applied Data Science

INTRODUCTION



Finding movie reviews was more difficult and time consuming prior to the introduction of the internet. The only options were to talk to family and friends, or to read movie reviews in newspapers or magazines. Today, before going to the movies or watching a movie at home, there are a variety of sources for movie feedback, such as IMdb.com. Scholars such as Bo Pang and Lillian Lee became interested in using Natural Language Processing (NLP) to classify the sentiment in these reviews as the data became more widely available.

Pang and Lee collected one of the more interesting datasets with movie reviews in 2005 for their academic paper, Seeing stars: Using class relationships to categorize sentiments with respect to rating scales.

OBJECTIVE

The goal of this assignment is to use various NLP techniques to train models to predict labels with the highest accuracy possible as measured by precision, recall, and F1 scores

GETTING THE DATA

1. Kaggle dataset

The dataset is made up of tab-separated files containing Rotten Tomatoes phrases. The train/test split has been kept for benchmarking purposes, but the sentences have been shuffled from their original order. The Stanford parser has broken each sentence down into many phrases. Each phrase has its own PhraseId. SentenceIds are assigned to each sentence. Repeated phrases (such as short/common words) are only included in the data once.

- The phrases and sentiment labels are contained in **train.tsv**. We've also included a SentenceId so you can keep track of which phrases belong to which sentences.
- The file **test.tsv** only contains phrases. Each phrase must be labeled with a sentiment.

Sentiments Labels

- 0 - negative
- 1 - somewhat negative
- 2 - neutral
- 3 - somewhat positive
- 4 - positive

Link: *https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/data*

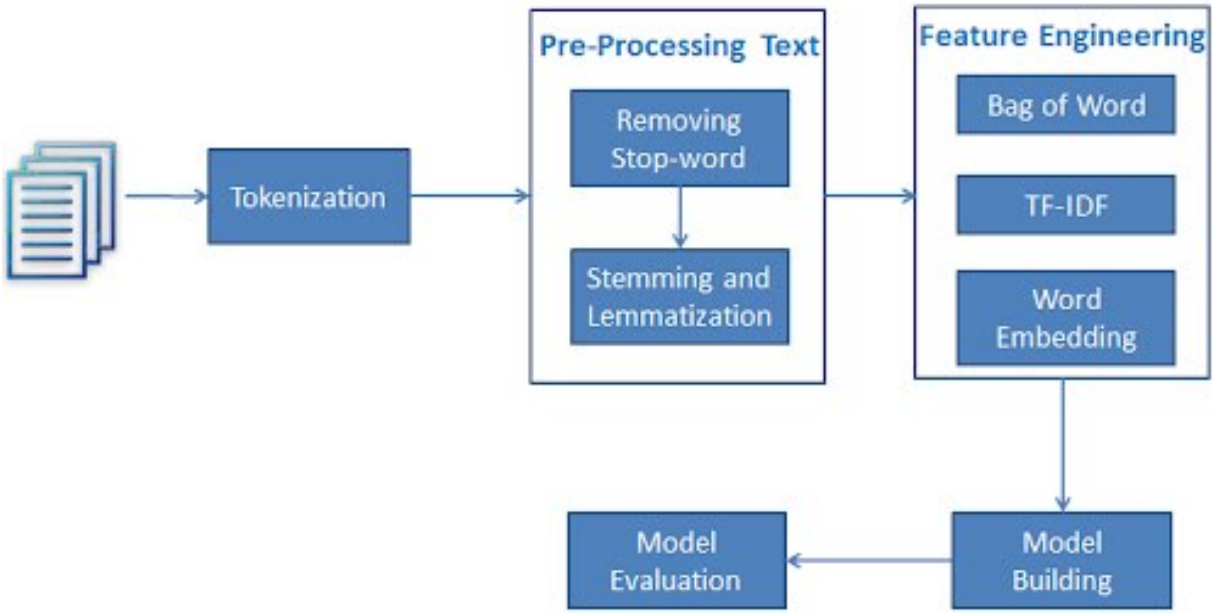
2. Cornell's movie review data, organized into sentiment polarity, sentiment scale, and subjectivity sections for sentiment analysis.

Link: *http://www.cs.cornell.edu/people/pabo/movie-review-data/*

3. IMDB movie review data from Stanford - containing 50,000 reviews.
   This dataset is split equally into 25,000 training and 25,000 test sets.
   It also provides unannotated data as well.

Link: *http://ai.stanford.edu/~amaas/data/sentiment/*

PROCESS OVERVIEW



EXPLORATORY ANALYSIS

Reading the train and test tsv files from the corpus folder into dataframes after importing the libraries & Loading the Data.

In [47]:
```
#Reading the train and test tsv files from the corpus folder into dataframes

try:
    train = pd.read_csv("kagglemoviereviews/corpus/train.tsv",sep="\t")
    print('Train data no. of rows: ',len(train))

except:
    print("Wrong Directory")

try:
    test = pd.read_csv("kagglemoviereviews/corpus/test.tsv",sep="\t")
    print('Train data no. of rows: ',len(test))

except:
    print("Wrong Directory")
```

```
Train data no. of rows:  156060
Train data no. of rows:  66292
```

The first five rows of information in the training data set are depicted below

In [8]:
```
#Analyze Dataframe  - metadata
train.info()

#Analyze Dataframe - shape
train.shape

#Analyze Dataframe - top 5 rows
train.head()
```

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage that what is good for the goose is also good for the gander , some of which occasionally amuses but none of which amounts to much of a story .	1
1	2	1	A series of escapades demonstrating the adage that what is good for the goose	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

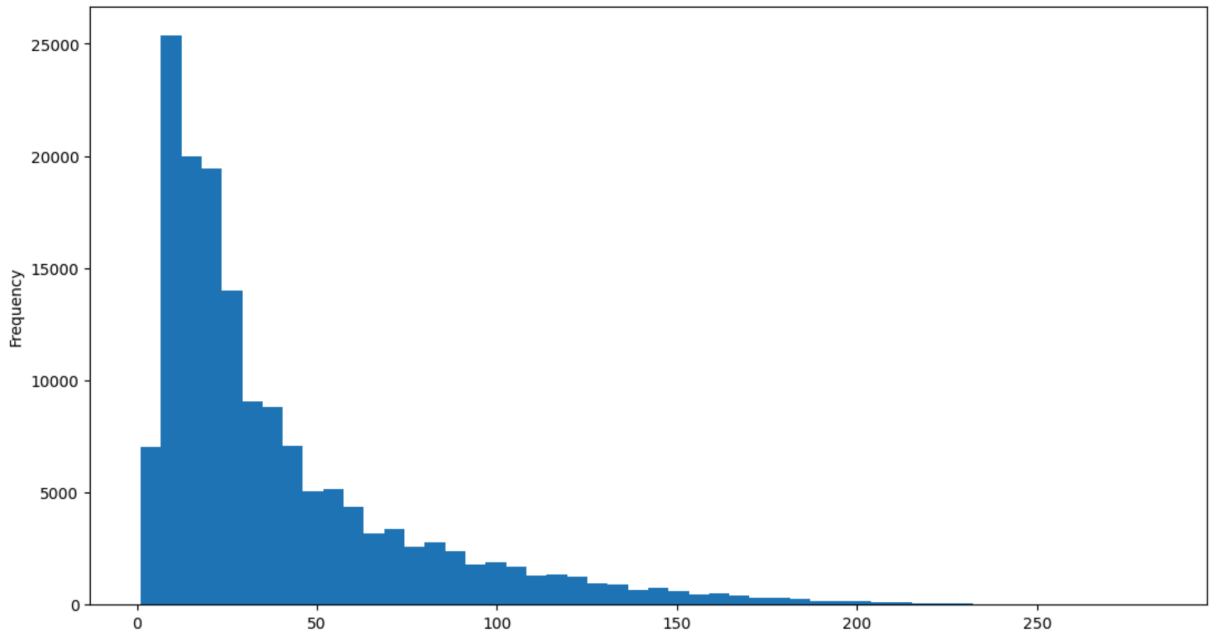
1. We created conditions depending on the score, i.e.
   - 0, 1 - Negative
   - 2 - Neutral
   - 3, 4 - Positive

2. And we assign the conditional values to the new column
3. We created a new column, which also displays the length of the phrase

	PhraseId	SentenceId	Phrase	Sentiment	Score	length
0	1	1	A series of escapades demonstrating the adage that what is good for the goose is also good for the gander , some of which occasionally amuses but none of which amounts to much of a story .	1	negative	188
1	2	1	A series of escapades demonstrating the adage that what is good for the goose	2	neutral	77
2	3	1	A series	2	neutral	8
3	4	1	A	2	neutral	1
4	5	1	series	2	neutral	6

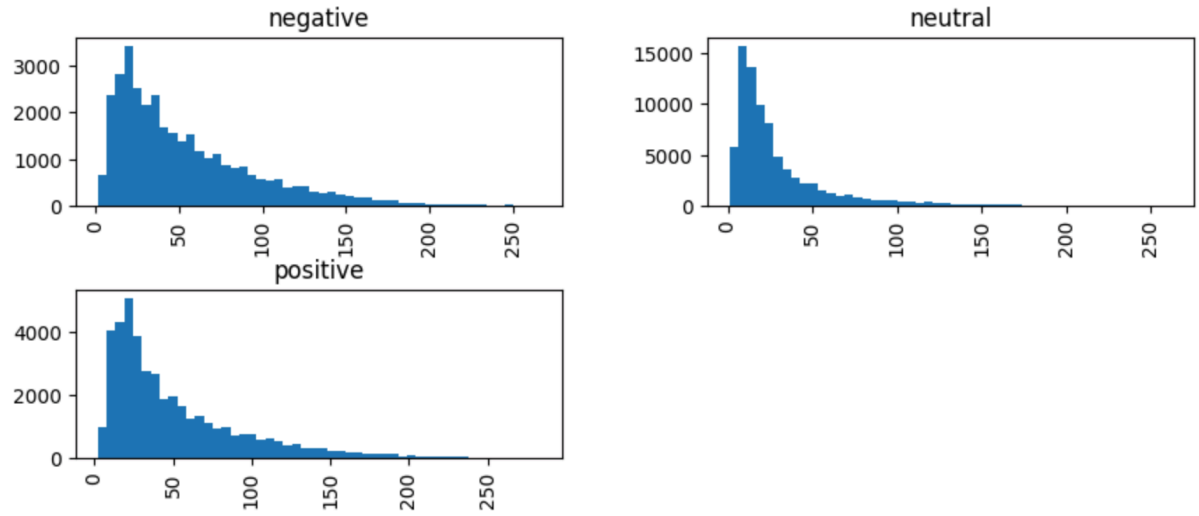
The distribution of sentiment scores was then examined to see if the dataset was balanced.

- 0 - negative
- 1 - somewhat negative
- 2 - neutral
- 3 - somewhat positive
- 4 - positive



On analyzing the distribution of reviews based on the number of characters, we found that 95% of the phrases have fewer than 25 words.

Analyzing the distribution of reviews based on sentiments



STOPPING THE STOP WORDS

```
#Stopping the Stop words

nltk_stopwords = nltk.corpus.stopwords.words('english')
print('Number of stopwords: ',len(nltk_stopwords))
nltk_stopwords[:10]
```

```
Number of stopwords:  179

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

```
#More Stop words

more_stopwords = ['-rrb-','...','-lrb-','``','-','.','\'s','```','--','\'','n\'t','film','movie','films']
more_stopwords = nltk_stopwords + more_stopwords
print('Number of stopwords: ',len(more_stopwords))
more_stopwords[:10]
```

```
Number of stopwords:  194

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

CREATING BAG OF WORDS

```
In [24]: # Corpus Creation

         corpus=[]
         for i in range(0,len(train)):
             review = re.sub('^[^a-z]+$', ' ', train['Phrase'][i] )
             review=review.lower()
             review=review.split()
             review=' '.join(review)
             corpus.append(review)
```

```
In [25]: corpus[:10]

Out[25]: ['a series of escapades demonstrating the adage that what is good for the goose is also good for the gander , some
         of which occasionally amuses but none of which amounts to much of a story .',
          'a series of escapades demonstrating the adage that what is good for the goose',
          'a series',
          '',
          'series',
          'of escapades demonstrating the adage that what is good for the goose',
          'of',
          'escapades demonstrating the adage that what is good for the goose',
          'escapades',
          'demonstrating the adage that what is good for the goose']
```

TOKENISATION

Printing top 5 Tokens
Removing stop words, more stop words and checking

In [134]:
```
#Tokenisation

all_tokens = [tok for msg in corpus for tok in nltk.word_tokenize(msg)]
print('number of tokens: ',len(all_tokens))

#Printing top 5 tokens
all_tokens[:5]
```

```
number of tokens:  1124612
```

Out[134]:
```
['a', 'series', 'of', 'escapades', 'demonstrating']
```

```
#apply stop words
filtered_sent=[]
for w in all_tokens:
    if w not in nltk_stopwords:
        filtered_sent.append(w)
```

```
#apply more stop words
more_filtered=[]
for w in filtered_sent:
    if w not in more_stopwords:
        more_filtered.append(w)
```

```
#Printing top 30 words in terms of frequency
clean_reviewsFD = nltk.FreqDist(more_filtered)
clean_top_words = clean_reviewsFD.most_common(30)
for word, freq in clean_top_words:
    print(word, ",", freq)
```

one	3609
like	3071
story	2520
good	2043
characters	1882
much	1862
time	1747
comedy	1721
even	1597
little	1573
funny	1522
way	1511

life	1484
make	1396
movies	1345
love	1296
new	1278
enough	1248
work	1243
us	1218
bad	1211
something	1152
would	1118
never	1114
director	1099
many	1094
people	1073
made	1060
best	1059
two	1032

PRE PROCESSING

This step entails loading and cleaning the data in preparation for text classification.

1. **Tokenization** is the process of breaking down large chunks of text into smaller pieces. NLTK includes a default processing pipeline that starts with tokenization, making this process a breeze. You can do sentence tokenization or word tokenization in NLTK. A token is a single entity that serves as the foundation for a sentence or paragraph.

   - **Sentence tokenization** is the process of splitting text into individual sentences.
   - **Word tokenization** is the process of splitting a large sample of text into words.

2. **Removing Stop words** - The idea is to simply remove words that appear frequently in all of the documents in the corpus. Articles and pronouns are typically classified as stop words.
3. **Frequency Distribution -** A frequency distribution keeps track of how many times each outcome of an experiment has occurred.
4. **Normalization** - We normalize text to reduce its randomness and bring it closer to a predefined "standard."
   - Stemming
   - Lemmatization
   - Bigrams
   - PMI
5. **POS Tagging -** Part-of-speech (POS) Tagging is a popular Natural Language Processing process that refers to categorizing words in a text (corpus) in accordance with a specific part of speech, based on the definition of the word and its context.

```
def alpha_filter(w):

    #Pattern to match non-alphabetical characters in a word
    pattern = re.compile('^[^a-z]+$')
    if (pattern.match(w)):
        return True
    else:
        return False

def Preprocess(df):

    for i in df['Phrase']:
        tokenizer = nltk.RegexpTokenizer(r"\w+")
        i = tokenizer.tokenize(i)
#-----------------------------------------------------------------------------------------------
    tokenized_review_1 = df['Phrase'].apply(lambda x: x.split())
#-----------------------------------------------------------------------------------------------
    ps = PorterStemmer()
    WL = WordNetLemmatizer()

    stemmed_review = tokenized_review_1.apply(lambda x: [ps.stem(i) for i in x])
    lemmatized_review = tokenized_review_1.apply(lambda x: [WL.lemmatize(i) for i in x])
#-----------------------------------------------------------------------------------------------
    stop = stopwords.words('english')
    morestopwords = ['could','would','might','must','need','sha','wo','y',"'s","'d","'ll"
```

```
                ,"'t","'m","'re","'ve", "n't",'wa','gon','na']
    stop = stop + morestopwords
    stemmed_review = stemmed_review.apply(lambda x: [item for item in x if item not in stop])
    lemmatized_review = lemmatized_review.apply(lambda x: [item for item in x if item not in
stop])
    #Add in negation
#-------------------------------------------------------------------------------------------
    for i in range(len(stemmed_review)):
        stemmed_review[i] = ' '.join(stemmed_review[i])

    df['stemmed_review'] = stemmed_review


    for i in range(len(lemmatized_review)):
        lemmatized_review[i] = ' '.join(lemmatized_review[i])

    df['lemmatized_review'] = lemmatized_review
    df = df[df["stemmed_review"] != '']
    df = df[df["lemmatized_review"] != '']
```

Pre Processing Train Data

```
# pre-processing train data
starttime = timeit.default_timer()
print("The start time is :",starttime)

try:
    Preprocess(train)
    print("pre-processing completed.")

except AttributeError as e:
    print("Oops..! something went wrong!\n")
    print(e)

print("The time difference is :", round(timeit.default_timer() - starttime,0)," seconds")
```

```
The start time is : 9218.689998721
pre-processing completed.
The time difference is : 27.0  seconds
```

Pre Processing Test Data

```
# pre-processing test data
starttime = timeit.default_timer()
print("The start time is :",starttime)

try:
    Preprocess(test)
    print("pre-processing completed.")

except AttributeError as e:
    print("Oops..! something went wrong!\n")
    print(e)

print("The time difference is :", round(timeit.default_timer() - starttime,0)," seconds")
```

```
The start time is : 9245.795787042
pre-processing completed.
The time difference is : 12.0  seconds
```

Checking out Top 5 Negative Words

	PhraseId	SentenceId	Phrase	Sentiment	Score	length	stemmed_review	lemmatized_review
0	1	1	A series of escapades demonstrating the adage that what is good for the goose is also good for the gander , some of which occasionally amuses but none of which amounts to much of a story .	1	negative	188	A seri escapad demonstr adag good goos also good gander , occasion amus none amount much stori .	A series escapade demonstrating adage good goose also good gander , occasionally amuses none amount much story .
33	34	1	the gander , some of which occasionally amuses but none of which amounts to much of a story	1	negative	91	gander , occasion amus none amount much stori	gander , occasionally amuses none amount much story
47	48	1	but none of which amounts to much of a story	1	negative	44	none amount much stori	none amount much story
49	50	1	none of which amounts to much of a story	1	negative	40	none amount much stori	none amount much story
81	82	3	Even fans of Ismail Merchant 's work , I suspect , would have a hard time sitting through this one .	1	negative	100	even fan ismail merchant work , I suspect , hard time sit thi one .	Even fan Ismail Merchant work , I suspect , hard time sitting one .

Checking out Top 5 Positive Words

	PhraseId	SentenceId	Phrase	Sentiment	Score	length	stemmed_review	lemmatized_review
21	22	1	good for the goose	3	positive	18	good goos	good goose
22	23	1	good	3	positive	4	good	good
46	47	1	amuses	3	positive	6	amus	amuses
63	64	2	This quiet , introspective and entertaining independent is worth seeking .	4	positive	74	thi quiet , introspect entertain independ worth seek .	This quiet , introspective entertaining independent worth seeking .
64	65	2	This quiet , introspective and entertaining independent	3	positive	55	thi quiet , introspect entertain independ	This quiet , introspective entertaining independent

Tokenizing by Regular Word Tokeniser

```
letters_only = re.sub("[^a-zA-Z]",      #Searches for all non-letters
                      " ",              #Replaces all non-letters with spaces
                      str(train["lemmatized_review"]))

filetokens = nltk.word_tokenize(letters_only)
# choose to treat upper and lower case the same
# by putting all tokens in lower case

filewords = [w.lower() for w in filetokens]
data = train["lemmatized_review"]

# Tokenization using WordTokenizer
filetokens = nltk.word_tokenize(data[100])

# choose to treat upper and lower case the same
# by putting all tokens in lower case
filewords = [w.lower() for w in filetokens]
len(filewords)
```

Setup to process bigrams

```
from nltk.collocations import *
bigram_measures = nltk.collocations.BigramAssocMeasures()
finder = BigramCollocationFinder.from_words(filetokens)
```

Choose to use both the non-alpha word filter and a stopwords filter
finder.apply_word_filter(alpha_filter)

Score by frequency and display the top 50 bigrams

```
scored = finder.score_ngrams(bigram_measures.raw_freq)
print ()
print ("Bigrams from file with top 50 frequencies")
for item in scored[:50]:
        print (item)
```

```
Bigrams from file with top 50 frequencies
(('hard', 'time'), 0.2)
(('sitting', 'one'), 0.2)
(('time', 'sitting'), 0.2)
```

Applying Frequency Filter to get good results

finder.apply_freq_filter(5)
scored = finder.score_ngrams(bigram_measures.pmi)
print ("Bigrams from file with top 50 by Mutual Information:")
for bscore in scored[:50]:
    print (bscore)

train[“Sentiment"].value_counts()

```
test.head()
```

	PhraseId	SentenceId	Phrase	stemmed_review	lemmatized_review
0	156061	8545	An intermittently pleasing but mostly routine effort .	intermitt pleas mostli routin effort .	An intermittently pleasing mostly routine effort .
1	156062	8545	An intermittently pleasing but mostly routine effort	intermitt pleas mostli routin effort	An intermittently pleasing mostly routine effort
2	156063	8545	An		An
3	156064	8545	intermittently pleasing but mostly routine effort	intermitt pleas mostli routin effort	intermittently pleasing mostly routine effort
4	156065	8545	intermittently pleasing but mostly routine	intermitt pleas mostli routin	intermittently pleasing mostly routine

EXPERIMENTATION

1. Multinomial Naïve Bayes With Stemming & Classification Report

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB

pipeline = Pipeline([
    ('bow',CountVectorizer(analyzer="word")), # strings to token integer counts
    ('classifier', MultinomialNB()), # train on TF-IDF vectors w/ Naive Bayes classifier
])

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report,accuracy_score

x_train,x_test,y_train,y_test = train_test_split(NB_data['stemmed_review'],
NB_data['Sentiment'], test_size=0.2, random_state=42)

pipeline.fit(x_train,y_train)
n_b_predictions = pipeline.predict(x_test)
print(classification_report(n_b_predictions,y_test))
print("-"*100)
print(confusion_matrix(n_b_predictions,y_test))
print("-"*100)
print(pipeline.score(x_train,y_train))
print(accuracy_score(n_b_predictions,y_test))
```

Precision	Recall	F1 Score	Support	Accuracy
0.66	0.61	0.63	31212	0.61

2. Multinomial Naïve Bayes With Lemmatization & Classification Report

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report,accuracy_score

x_train,x_test,y_train,y_test = train_test_split(NB_data['lemmatized_review'],
NB_data['Sentiment'], test_size=0.2, random_state=42)

pipeline.fit(x_train,y_train)
n_b_predictions = pipeline.predict(x_test)
print(classification_report(n_b_predictions,y_test))
print("-"*100)
print(confusion_matrix(n_b_predictions,y_test))
print("-"*100)
print(pipeline.score(x_train,y_train))
print(accuracy_score(n_b_predictions,y_test))
```

Precision	Recall	F1 Score	Support	Accuracy
0.65	0.61	0.63	31212	0.61

3. Deep Learning - Tensorflow

```
#import libraries for tensorflow
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding
from tensorflow.keras.layers import LSTM
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
tokenize = Tokenizer()
tokenize.fit_on_texts(x_train.values)

X_test = test.stemmed_review
X_train = tokenize.texts_to_sequences(x_train)
X_test = tokenize.texts_to_sequences(X_test)

X_train = pad_sequences(X_train)
X_test = pad_sequences(X_test)
EMBEDDING_DIM = 100
unknown = len(tokenize.word_index)+1

model = Sequential()
model.add(Embedding(unknown, EMBEDDING_DIM))
model.add(LSTM(units=128, dropout=0.2, recurrent_dropout=0.2 ))
model.add(Dense(5, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
# execute the model
model.fit(X_train, y_train, batch_size=128, epochs=7, verbose=1)
```

Precision	Recall	F1 Score	Support	Accuracy
				0.74

More Models

4. Random Forest classifier

```
from sklearn.model_selection import train_test_split

# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

starttime = timeit.default_timer()
print("The start time is :",starttime)
```

```
#Fitting Random Forest classifier with 50 trees to the Training set
from sklearn.ensemble import RandomForestClassifier
pipelineRF = RandomForestClassifier(n_estimators = 50, criterion = 'entropy', random_state =
0)
pipelineRF.fit(X_train, y_train)

print("The time difference is :", round(timeit.default_timer() - starttime,0)," seconds”)

#Confusion Matrix & Classification Report

from sklearn.metrics import confusion_matrix, classification_report,accuracy_score

#Predicting the Test set results
rf_predictions = pipelineRF.predict(X_test)
print(classification_report(rf_predictions,y_test))
print("-"*100)
print(confusion_matrix(rf_predictions,y_test))
print("-"*100)
print(pipelineRF.score(X_train,y_train))
print(accuracy_score(rf_predictions,y_test))
```

Precision	Recall	F1 Score	Support	Accuracy
0.73	0.71	0.72	31212	0.71

5. Multinomial Naïve Bayes & Classification Report

```
from sklearn.metrics import confusion_matrix, classification_report,accuracy_score
from sklearn.feature_extraction.text import TfidfVectorizer as TV
tv = TV(ngram_range =(1,1),max_features = 3000)
X_tv = tv.fit_transform(corpus).toarray()
X_train, X_test, y_train, y_test = train_test_split(X_tv, y, test_size = 0.20, random_state = 0)
from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy of the classifier: ",acc,"\n")
# print("Confusion matrix is :\n",metrics.confusion_matrix(y_test,y_pred))
# print("Classification report: \n" ,metrics.classification_report(y_test,y_pred))
print(classification_report(y_pred,y_test))
print("-"*100)
print(confusion_matrix(y_pred,y_test))
print("-"*100,"\n")

print(classifier.score(X_train,y_train))
print(accuracy_score(y_pred,y_test))
```

Precision	Recall	F1 Score	Support	Accuracy
0.79	0.64	0.68	31212	0.64

6. Multinomial Naive Bayes with Cross Validation

```
def document_features(document, word_features):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['V_{}'.format(word)] = (word in document_words)
    return features

def cross_validation_PRF(num_folds, featuresets, labels):
    subset_size = int(len(featuresets)/num_folds)
    print('Each fold size:', subset_size)
    # for the number of labels - start the totals lists with zeroes num_labels = len(labels)
    total_precision_list = [0] * num_labels
    total_recall_list = [0] * num_labels
    total_F1_list = [0] * num_labels


    # iterate over the folds
    for i in range(num_folds):
        test_this_round = featuresets[(i*subset_size):][:subset_size]
        train_this_round = featuresets[:(i*subset_size)] + featuresets[((i+1)*subset_size):]
        #train using train_this_round
        classifier = nltk.NaiveBayesClassifier.train(train_this_round)
        #evaluate against test_this_round to produce the gold and predicted labels
        goldlist = []
        predictedlist = []
        for (features, label) in test_this_round:
            goldlist.append(label)
            predictedlist.append(classifier.classify(features))

        # computes evaluation measures for this fold and
        # returns list of measures for each label
        print('Fold', i)
        (precision_list, recall_list, F1_list) \ = eval_measures(goldlist, predictedlist, labels)

        # take off triple string to print precision, recall and F1 for each fold
        '''
        print('\tPrecision\tRecall\t\tF1')
        # print measures for each label
        for i, lab in enumerate(labels):
            print(lab, '\t', "{:10.3f}".format(precision_list[i]), \ "{:10.3f}".format(recall_list[i]),
"{:10.3f}".format(F1_list[i]))
        '''

        # for each label add to the sums in the total lists
        for i in range(num_labels):
```

```
            # for each label, add the 3 measures to the 3 lists of totals
            total_precision_list[i] += precision_list[i]
            total_recall_list[i] += recall_list[i]
            total_F1_list[i] += F1_list[i]

    # find precision, recall and F measure averaged over all rounds for all labels
    # compute averages from the totals lists
    precision_list = [tot/num_folds for tot in total_precision_list]
    recall_list = [tot/num_folds for tot in total_recall_list]
    F1_list = [tot/num_folds for tot in total_F1_list]
    #the evaluation measures in a table with one row per label
    print('\nAverage Precision\tRecall\t\tF1 \tPer Label')
    #print measures for each label
    for i, lab in enumerate(labels):
        print(lab, '\t', "{:10.3f}".format(precision_list[i]), \ "{:10.3f}".format(recall_list[i]),
"{:10.3f}".format(F1_list[i]))

    # print macro average over all labels - treats each label equally
    print('\nMacro Average Precision\tRecall\t\tF1 \tOver All Labels')
    print('\t', "{:10.3f}".format(sum(precision_list)/num_labels), \ "{:10.3f}".format(sum(recall_list)/
num_labels), \ "{:10.3f}".format(sum(F1_list)/num_labels))

    # for micro averaging, weight the scores for each label by the number of items
    # this is better for labels with imbalance
    # first intialize a dictionary for label counts and then count them

    label_counts = {}
    for lab in labels:
        label_counts[lab] = 0

    # count the labels
    for (doc, lab) in featuresets:
        label_counts[lab] += 1

    # make weights compared to the number of documents in featuresets
    num_docs = len(featuresets)
    label_weights = [(label_counts[lab] / num_docs) for lab in labels]
    print('\nLabel Counts', label_counts)

    #print('Label weights', label_weights)
    # print macro average over all labels
    print('Micro Average Precision\tRecall\t\tF1 \tOver All Labels')
    precision = sum([a * b for a,b in zip(precision_list, label_weights)])
    recall = sum([a * b for a,b in zip(recall_list, label_weights)])
    F1 = sum([a * b for a,b in zip(F1_list, label_weights)])
    print( '\t', "{:10.3f}".format(precision), \ "{:10.3f}".format(recall), “{:10.3f}".format(F1))

def eval_measures(gold, predicted, labels):

    # these lists have values for each label
    recall_list = []
    precision_list = []
    F1_list = []
```

```
    for lab in labels:
        # for each label, compare gold and predicted lists and compute values
        TP = FP = FN = TN = 0
        for i, val in enumerate(gold):
            if val == lab and predicted[i] == lab: TP += 1
            if val == lab and predicted[i] != lab: FN += 1
            if val != lab and predicted[i] == lab: FP += 1
            if val != lab and predicted[i] != lab: TN += 1

        # use these to compute recall, precision, F1
        # for small numbers, guard against dividing by zero in computing measures if (TP == 0) or
(FP == 0) or (FN == 0):
            recall_list.append (0)
            precision_list.append (0)
            F1_list.append(0)
        else:
            recall = TP / (TP + FP)
            precision = TP / (TP + FN)
            recall_list.append(recall)
            precision_list.append(precision)
            F1_list.append( 2 * (recall * precision) / (recall + precision))

    # the evaluation measures in a table with one row per label
    return (precision_list, recall_list, F1_list)

#Kaggle training file - pre processing for cross validation

starttime = timeit.default_timer()
print("The start time is :",starttime)

## function to read kaggle training file, train and test a classifier
#def processkaggle(dirPath,limitStr):
#convert the limit argument from a string to an int
#limit = int(limitStr)
#os.chdir(dirPath)

f = open('kagglemoviereviews/corpus/train.tsv', 'r')
# loop over lines in the file and use the first limit of them
phrasedata = []
for line in f:
# ignore the first line starting with Phrase and read all lines
    if (not line.startswith('Phrase')):
    # remove final end of line character
        line = line.strip()
    # each line has 4 items separated by tabs
    # ignore the phrase and sentence ids, and keep the phrase and sentiment
        phrasedata.append(line.split('\t')[2:4])
# possibly filter tokens
# lowercase - each phrase is a pair consisting of a token list and a label

#pick a random sample of length limit because of phrase overlapping sequences
random.shuffle(phrasedata)
phraselist = phrasedata[:len(phrasedata)]
```

```
print('Read', len(phrasedata), 'phrases, using', len(phraselist), 'random phrases')


# create list of phrase documents as (list of words, label)
phrasedocs = []
# add all the phrases
# each phrase has a list of tokens and the sentiment label (from 0 to 4) ### bin to only 3
categories for better performance
for phrase in phraselist:
    tokens = nltk.word_tokenize(phrase[0])
    phrasedocs.append((tokens, int(phrase[1])))

docs = []
for phrase in phrasedocs:
    lowerphrase = ([w.lower() for w in phrase[0]], phrase[1])
    docs.append (lowerphrase)

# # print a few
# for phrase in docs[:len(phrasedocs)]:
#     print (phrase)
# continue as usual to get all words and create word features

all_words_list = [word for (sent,cat) in docs for word in sent]
all_words = nltk.FreqDist(all_words_list)
print(len(all_words))

# get the 1500 most frequently appearing keywords in the corpus
word_items = all_words.most_common(1500)
word_features = [word for (word,count) in word_items]

# feature sets from a feature definition function
featuresets = [(document_features(d, word_features), c) for (d, c) in docs]

# train classifier and show performance in cross-validation
# make a list of labels
label_list = [c for (d,c) in docs]
labels = list(set(label_list)) # gets only unique labels

# starttime = timeit.default_timer()
# print("The start time is :",starttime)
# num_folds = 5
# cross_validation_PRF(num_folds, featuresets, labels)

print("\nThe time difference is :", round(timeit.default_timer() - starttime,0)," seconds")
```

- **Cross-validation & Classification Report | 5 Fold**

```
starttime = timeit.default_timer()
print("The start time is :",starttime)
num_folds = 5
cross_validation_PRF(num_folds, featuresets, labels)
print("\nThe time difference is :", round(timeit.default_timer() - starttime,0)," seconds”)
```

Precision	Recall	F1 Score	Support	Accuracy
0.56	0.53	0.53		

- **Cross-validation & Classification Report | 10 Fold**

```
starttime = timeit.default_timer()
print("The start time is :",starttime)
num_folds = 10
cross_validation_PRF(num_folds, featuresets, labels)
print("The time difference is :", round(timeit.default_timer() - starttime,0)," seconds")
```

Precision	Recall	F1 Score	Support	Accuracy
0.56	0.53	0.53		

SYNOPSIS

Overall, we were able to perform exploratory analysis, preprocessing techniques, tokenization using Wordtokenizer, removing stop words, stemming and lemmatization techniques, and finally, we were able to explore several machine learning classifiers, including NLTK's native classifiers and external classifiers, to predict the sentiment on the movie reviews dataset.

There were 156060 observations in the training set, with one dependent and four independent variables.

We concentrated primarily on Phrase (actual reviews) and Sentiment (rating from 0 to 4). The sentiment score is further classified into three major categories, as shown below.

- Positive (3,4) - 27% of all reviews are positive.
- Negative (0,1) - 22% of all reviews are negative.
- Neutral (2) - 51% of all reviews are neutral. Yes, half of the data has been classified as neutral.

There were a total of 200+ Stop words, including NLTK's and custom words, with alphanumeric characters excluded to improve data quality. To better understand the data spread, several visualization techniques were used, including Barplots, Tree graphs, Wordclouds, and other histograms.

The preprocessed data was then divided into train:test (80/20) groups, and various models were used to predict sentiment. Please see the table below for an overall comparison of the results.

Classifier	Precision	Recall	F1 Score	Support	Accuracy
Multinomial Naïve Bayes	0.79	0.64	0.68	31212	0.64
Multinomial Naïve Bayes with Lemmatization	0.65	0.61	0.63	31212	0.61

Classifier	Precision	Recall	F1 Score	Support	Accuracy
Multinomial Naïve Bayes with Stemming	0.66	0.61	0.63	31212	0.61
Naïve Bayes with Cross Validation: Fold 10	0.56	0.53	0.53		
Naïve Bayes with Cross Validation: Fold 5	0.56	0.53	0.53		
Random Forest	0.73	0.71	0.72	31212	0.71
Tensorflow					0.74

Multinomial Naïve Bayes				
Sentiment	**Precision**	**Recall**	**F1 Score**	**Support**
Positive	0.44	0.76	0.56	4828
Negative	0.27	0.75	0.39	2487
Neutral	0.91	0.61	0.73	23897
Overall Accuracy	0.64			

Random Forest				
Sentiment	**Precision**	**Recall**	**F1 Score**	**Support**
Positive	0.63	0.74	0.68	6995
Negative	0.57	0.73	0.64	5463
Neutral	0.82	0.70	0.76	18754
Overall Accuracy	0.71			

Tensorflow clearly had the best result, with nearly 75% accuracy.

- There is little difference between stemmed and lemmatized data, and response length appears to be important (displayed with best cross validated model in the random forest).
- Precision is defined as the proportion of correctly predicted positive observations to all predicted positive observations. Precision was highest in the Neutral category on both the NLTK Classifier and the external Classifier. This is primarily due to the higher number of observations (51%), as evidenced by the Support values.

- Recall (Sensitivity) - The ratio of correctly predicted positive observations to all observations is defined as recall. The recall percentage is consistent across all categories. That's great news!
- The F1 Score is calculated as the weighted average of Precision and Recall. As a result, this score considers both false positives and false negatives. It is not as intuitive as accuracy, but F1 is usually more useful than accuracy, especially if the class distribution is uneven. Accuracy works best when the cost of false positives and false negatives is comparable. On the random forest model, our F1 score is 0.76. F1 Score = 2(Recall Precision)/(Recall + Precision)
- As one might expect, converting the problem to a binary classification improved accuracy significantly. This classification excluded neutral responses and focused solely on whether the note was positive or negative. In a practical application, this could be a good place to start.
- The tensor flow models were extremely accurate, but they used an experimental black box approach. Our ability to cross-validate the results was also limited by the training time. A general conclusion was that cross validation dampened our enthusiasm for accuracy.

Indeed, the cross validated models demonstrated were more consistent with the Kaggle submissions. On Kaggle, my submission scored between 55 and 60%.

CONCLUSION

- Finally, we can categorize the movie reviews dataset as positive, negative, or neutral. The overall process includes data collection, pre-processing, feature engineering, and experimentation with native NLTK libraries and external Python classifiers.
- Furthermore, we can achieve an accuracy of 74% using Tensorflow (deep learning), 71% using Random Forest, and 64% using the Multinomial Nave Bayes model. Given that the data is more skewed towards the neutral category, this is within our expectations.
- Precision and recall% were comparatively better with the dataset provided, with 91% precision on the neutral category using Multinomial Nave Bayes. The F1 score in the case of Random Forest is 76%, which is relatively higher.
- We also discovered that stemming and lemmatization did not significantly improve accuracy. It is possible that it is not the best technique to use in this situation. Although the cross validation results did not meet our expectations, they piqued our interest and compelled us to investigate further.
- Finally, with the dataset provided, we achieve our expected accuracy of 60 to 80%. However, with better data quality and a more balanced distribution, as well as more training, compute, and time, we can select a better model to improve accuracy above 90%. Overall, this is an exciting topic for learning about the fundamentals of NLP techniques and their effectiveness in real-time applications such as sentiment analysis.

REFERENCES

In addition to course materials, I consulted online resources.
Please see the list of references below.

- *https://www.kaggle.com/prashant111/naive-bayes-classifier-in-python*
- *https://www.nltk.org/book/ch06.html*
- *https://medium.com/python-in-plain-english/introduction-to-sentiment-analysis-using-python-nltk-library-f00c227ef56e*
- *https://stackoverflow.com/questions/16379313/how-to-use-the-a-k-fold-cross-validation-in-scikit-with-naive-bayes-classifier-a/34659712*