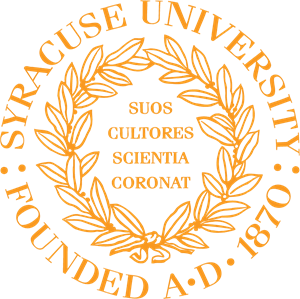
Final Project –CIS 668/IST 664

Classification of Kaggle Movie Reviews



*Student: Professor:* Michael Larche

Midhuneshwar Kandasamy Kanivalavan

SUID: 351564889

Mail: mkandasa@syr.edu

# Table of Contents

[Introduction 4](#_bookmark0)

[Data Description 4](#_bookmark1)

[Reading the Train Data 6](#_bookmark2)

[Tokenization and Filtering 6](#_bookmark3)

1. [Pre processing documents 7](#_bookmark4)
2. [Retrieving words/tokens 8](#_bookmark5)

[Creating Feature Sets 8](#_bookmark6)

1. [Bag of Words (BOW)/Unigram features 9](#_bookmark7)
2. [Bigram Features 9](#_bookmark8)
3. [Sentiment Lexicons 9](#_bookmark9)
4. [Negation word features: 10](#_bookmark10)
5. [POS features 11](#_bookmark11)

[Classification: Naïve Bayes Classifier 11](#_bookmark12)

1. [Normal Features without preprocessing 12](#_bookmark13)
2. [Bigram Features 13](#_bookmark14)
3. [Negation word Features 14](#_bookmark15)
4. [Preprocessed Features 15](#_bookmark16)
5. [Sentiment Lexicon features 16](#_bookmark17)

[Combination of features sets 17](#_bookmark18)

[Comparative Analysis of Logistic Regression and Decision Tree classifier 18](#_bookmark19)

[Logistic Regression 18](#_bookmark20)

1. [Normal Features without preprocessing 20](#_bookmark21)
2. [Bigram Features 21](#_bookmark22)
3. [Negation Word Features 22](#_bookmark23)
4. [Preprocessed Features 23](#_bookmark24)
5. [Sentiment Lexicon Features 24](#_bookmark25)

[Decision Tree Classifier 25](#_bookmark26)

1. [Normal Features without preprocessing 26](#_bookmark27)
2. [Bigram Features 27](#_bookmark28)
3. [Negation Word Features 28](#_bookmark29)
4. [Preprocessed Features 29](#_bookmark30)
5. [Sentiment Lexicon Features 30](#_bookmark31)

[Highlights of Sci-kit learn algorithms 31](#_bookmark32)

[Comparison of evaluation metrics 31](#_bookmark33)

[Conclusion 31](#_bookmark34)

[Lessons Learned 32](#_bookmark35)

# Introduction

The primary objective of this project is to classify and predict the sentiment of move reviews. The scale of sentiment classification is 0-4 with equal intervals. The details of scale for sentiment follows,

|  |  |
| --- | --- |
| Classification Scale | Sentiment |
| 0 | Negative |
| 1 | Strong Negative |
| 2 | Neutral |
| 3 | Positive |
| 4 | Strong Positive |

Based on the classification intervals, it can be said that the scaling is biased towards negative and positive reviews leaving neutral sentiment scoring a single interval classification.

The dataset was produced for the Kaggle competition, described here [data.](http://www.kaggle.com/c/sentiment-analysis-on-movie-reviews) This dataset data from the sentiment analysis by Socher et al, detailed at this web site: [http://nlp.stanford.edu/sentiment/.](http://nlp.stanford.edu/sentiment/)

The data was taken from the original Pang and Lee movie review corpus based on reviews from the Rotten Tomatoes web site. Socher’s group used crowd-sourcing to manually annotate all the subphrases of sentences with a sentiment label ranging over: “negative”, “somewhat negative”, “neutral”, “somewhat positive”, “positive”.

# Data Description

The data provided for this project has two data files namely ‘train.csv’ and ‘test.csv’. The details of train

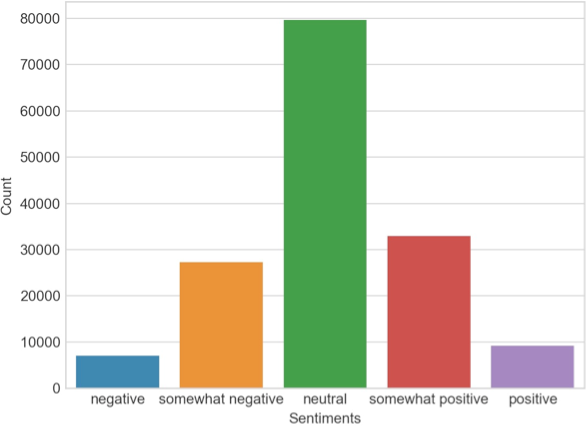
data are,

* 1,56,060 rows of movie reviews. Each row represents review for a movie
* 4 columns, these are

|  |  |
| --- | --- |
| Column\_name | Description |
| PhraseId | Represents id number of a phrase |
| SentenceId | Represents id for each sentence |

|  |  |
| --- | --- |
| Column\_name | Description |
| Phrase | Represents the review for movie |
| Phrase Sentiment | Represents the sentiment for each review |

Histogram of count of sentiments for train data,



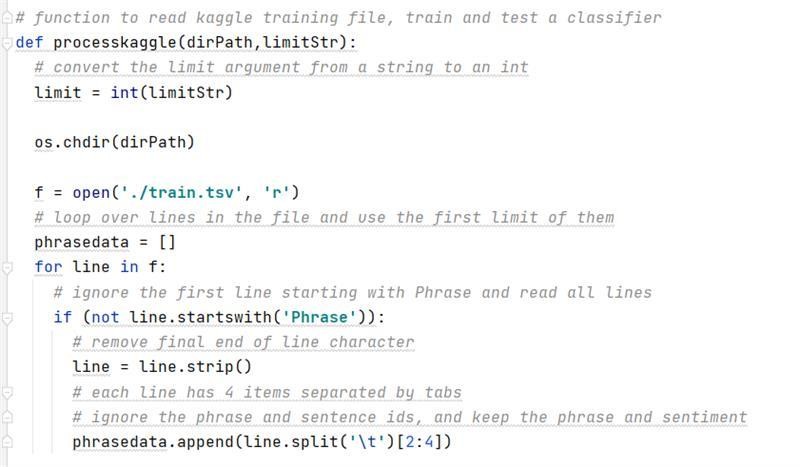
The details of test data are ,

* 66,292 rows of movie reviews. Each row represents review for a movie
* 3 columns, these are

|  |  |
| --- | --- |
| Column\_name | Description |
| PhraseId | Represents id number of a phrase |
| SentenceId | Represents id for each sentence |
| Phrase | Represents the review for movie |

It must be noted that test data has no sentiment, which means that sentiment column is the outcome/target label which must be predicted. Therefore, for all of our analysis Phraseid, SentenceID and Phrase will be features and Sentiment will be target label.

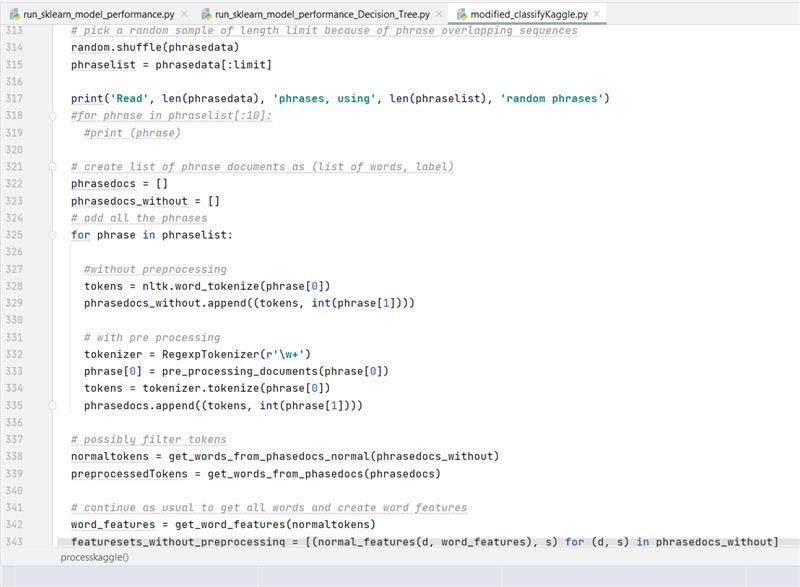
# Reading the Train Data

The first step is to read the data into python environment. This is done using he code below,

Here, we are writing a function which will loop through each line of the csv line and create a list called ‘phrasedata’. Further, we have a option to limit the number of rows we can include in the loaded data. This limiting the loaded data is helpful when we want to run the analysis on a limited number of the instances. Since we are trying to classify and then predict, the better approach would be to load all the available data, however, the constraint is the computing power. Though, we have a constraint we tried to run our analysis on entire dataset.

# Tokenization and Filtering

Once we have loaded the entire train dataset, we need to tokenize each word of the a phrase to proceed with our analysis. Using these tokenized phrases, we can create multiple different feature sets. The code developed is below,

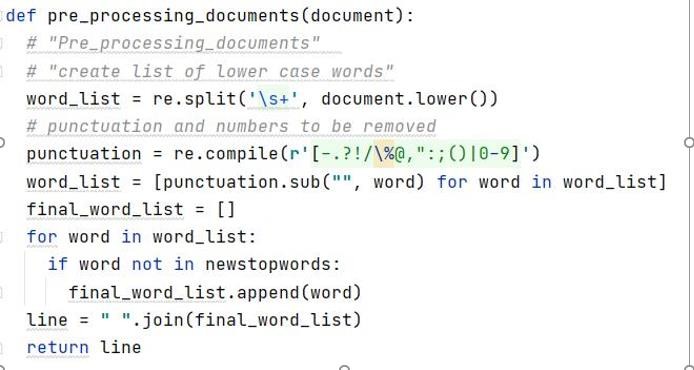


From the code it can be observed that we are leveraging tokenize function of nltk to tokenize the phrases. Before tokenization, we are preprocessing for all those tokens using a regular expression ‘\w+’, this regex pattern matches any alphanumeric with underscore with one or more occurrences. Once we have tokenized the data, we create two different lists *‘normaltokens’* which includes word phrases without filtering and the second list is *‘preprocessedTokens’* which includes word phrases with filtering.

To perform the above said tokenization and filtering we have defined the following functions,

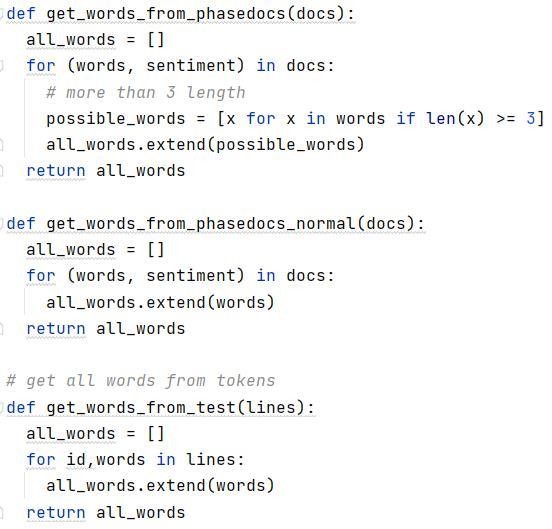
## Pre processing documents

This function splits each phrase into individual lines and converts them into lower case. Later, this lower case lines are run through regular expressions wherein any word with punctuation marks is removed and stored in a list called *‘word\_list’*. Further this list is checked for any stop words i.e. all the stop words are removed from the list and a list called *‘final\_word\_list’* is created.



## Retrieving words/tokens

There are three functions defined in this code. The first function returns list which contains tokens/words from the documents where the length is greater than 3. The second function returns a list of all words with corresponding sentiment. The third function returns all lines from the tokens.

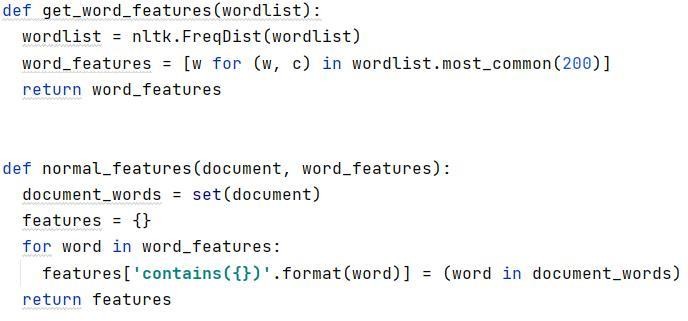


# Creating Feature Sets

The next key step in classification tasks in NLP is to create features from raw tokens. We had to define multiple features generating functions such as bag of words(BOW), bigrams, sentiment lexicons, negation words, POS tag features etc. Let us try to understand each of these functions,

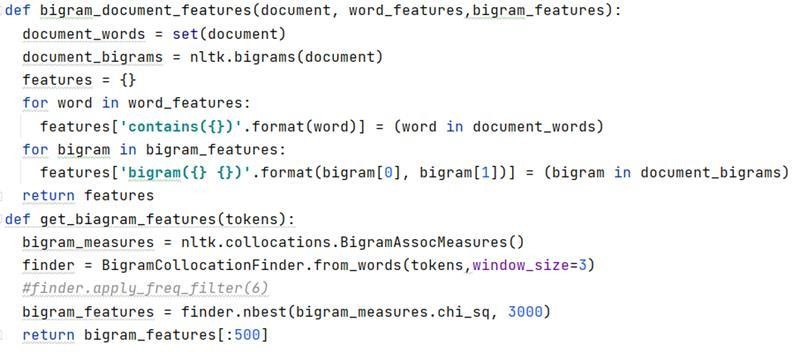
## Bag of Words (BOW)/Unigram features

For creating unigrams we defined two different functions. The first function creates a list of most repeated 200 words from the *‘wordlist’* which has processed tokens. The second function creates unique list of words from the documents and returns them as features.



## Bigram Features

We have worked on generating bigram feature from documents to get high frequent bigrams. We have filtered out special characters as well as filter by frequency. We have used the nbest function which just returns the highest scoring bigrams, using the number specified in both the measures.



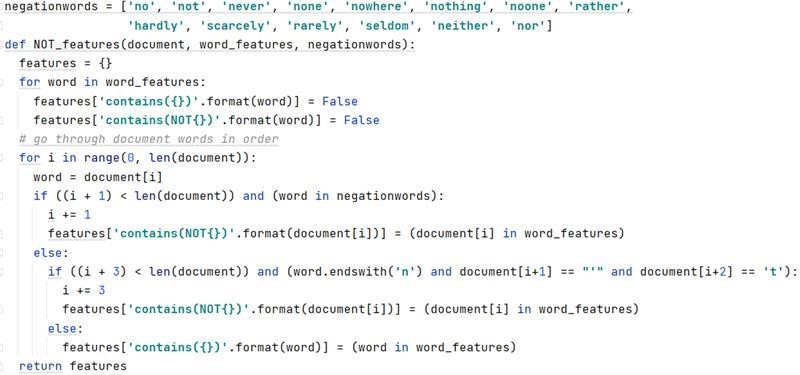
## Sentiment Lexicons

We will first read in the subjectivity words from the subjectivity lexicon file created by Janyce Wiebe and her group at the University of Pittsburgh in the MPQA project. Although these words are often

used as features themselves or in conjunction with other information, we will create two features that involve counting the positive and negative subjectivity words present in each document. I copy and pasted the definition of the readSubjectivity function from the Subjectivity.py module which is provided by Professor. It creates a Subjectivity Lexicon that is represented here as a dictionary, where each word is mapped to a list containing the strength and polarity. A feature extraction function that has all the word features as before, but also has two features ‘positivecount’ and ‘negativecount’. These features contain counts of all the positive and negative subjectivity words, where each weakly subjective word is counted once and each strongly subjective word is counted twice.

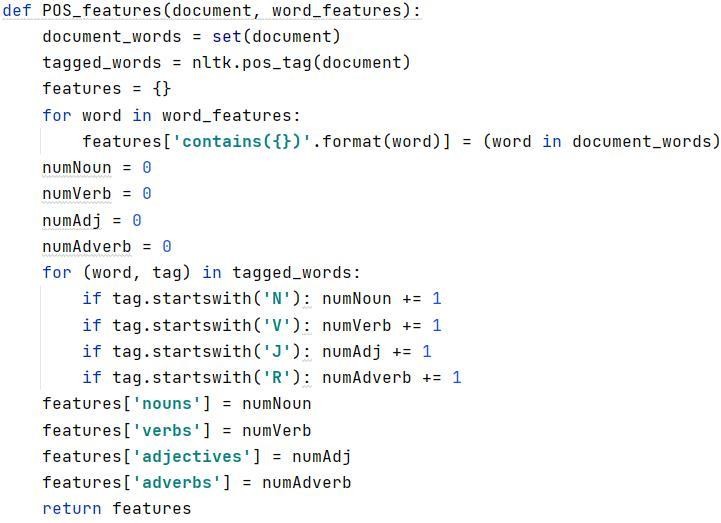
## Negation word features:

Negation of opinions is an important part of sentimental classification. Here I tried a simple strategy which professor explained in Lab-10. I look for negation words "not", "never" and "no" and negation that appears in contractions of the form "doesn", "'", "t". For example, my first document has the following words: if', 'you', 'don', "'", 't', 'like', 'this', 'film', ',', 'then', 'you', 'have', 'a', 'problem', 'with', 'the', 'genre', 'itself', One strategy with negation words is to negate the word following the negation word, while other strategies negate all words up to the next punctuation or use syntax to find the scope of the negation. I followed the first strategy here, and I go through the document words in order adding the word features, but if the word follows a negation words, change the feature to negated word.



## POS features

We have done this classification task with help of part-of-speech tag features. This is more likely for shorter units of classification; such as sentence level classification or shorter social media such as tweets. In this dataset, we have large training dataset and moreover, in the NLTK, this is difficult to demonstrate, since on computer, it takes the default NLTK POS tagger too much time. Because of this limitation we tested on only 2000 training sentences. The most common way to use POS tagging information is to include counts of various types of word tags.



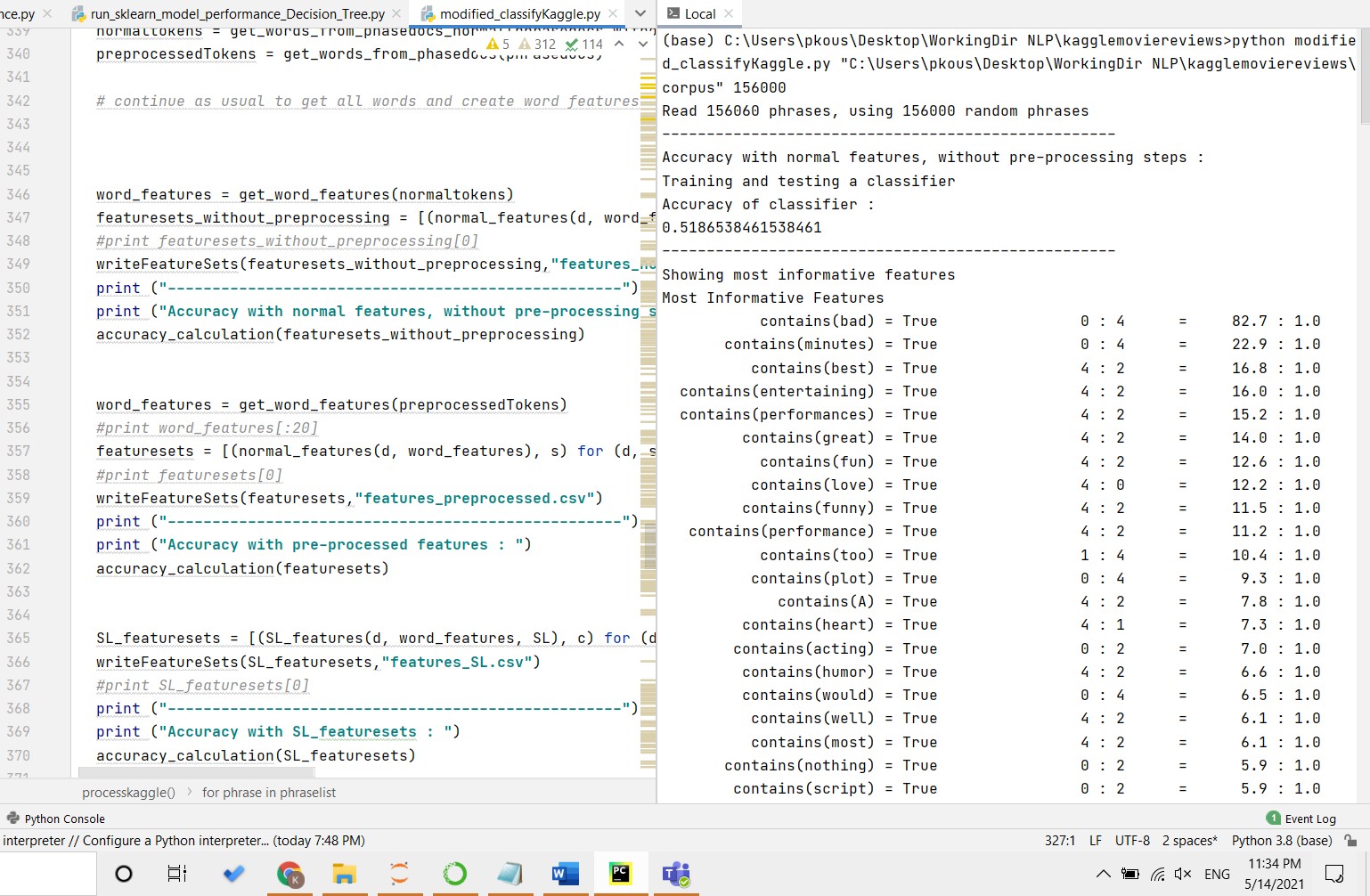
# Classification: Naïve Bayes Classifier

In this section we will discuss the classifier obtained for each of the feature sets created by running the above functions. Before delving into details of outputs for each of the feature sets, let’s look at the accuracies obtained for each of them,

|  |  |
| --- | --- |
| Feature Set | Accuracy achieved |
| Normal Features without preprocessing | 51.86% |
| Bigram Features | 53.75% |
| Negation word Features | 55.47% |
| Preprocessed Features | 53.75% |
| Sentiment Lexicon features | 55.08% |

Since, we have established the accuracies for each of the feature sets, let us now focus on the confusion matrix and make some definitive conclusions.

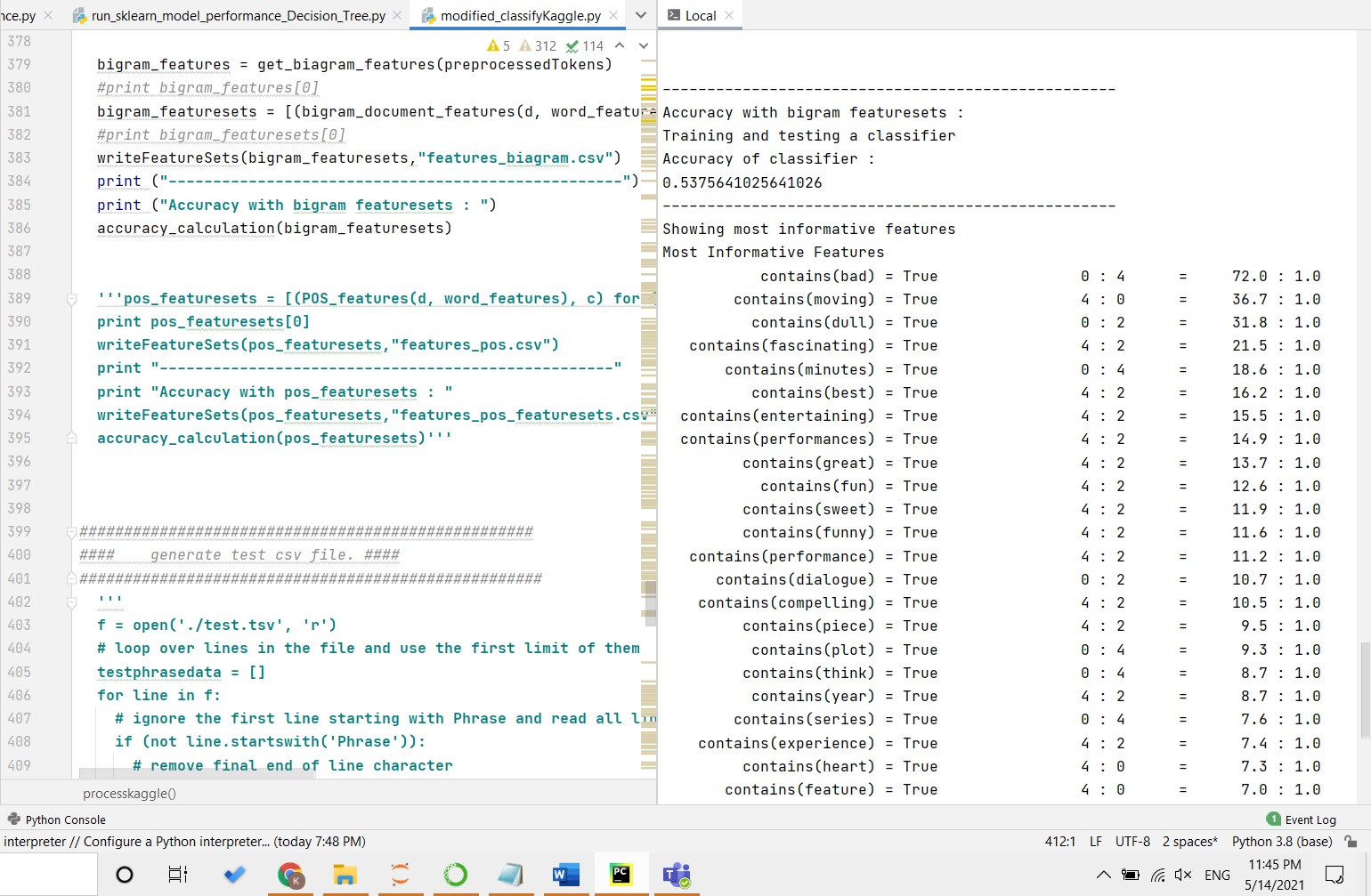
## Normal Features without preprocessing



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Classification  Scale | Sentiment |  |
| 0 | Negative |  |
| 1 | Strong  Negative |  |
| 2 | Neutral |  |
| 3 | Positive |  |
| 4 | Strong  Positive |  |

From the above confusion matrix, it can be said that most of neutral sentiments are predicted accurately and the least accurately predicted label is positive sentiment labels.

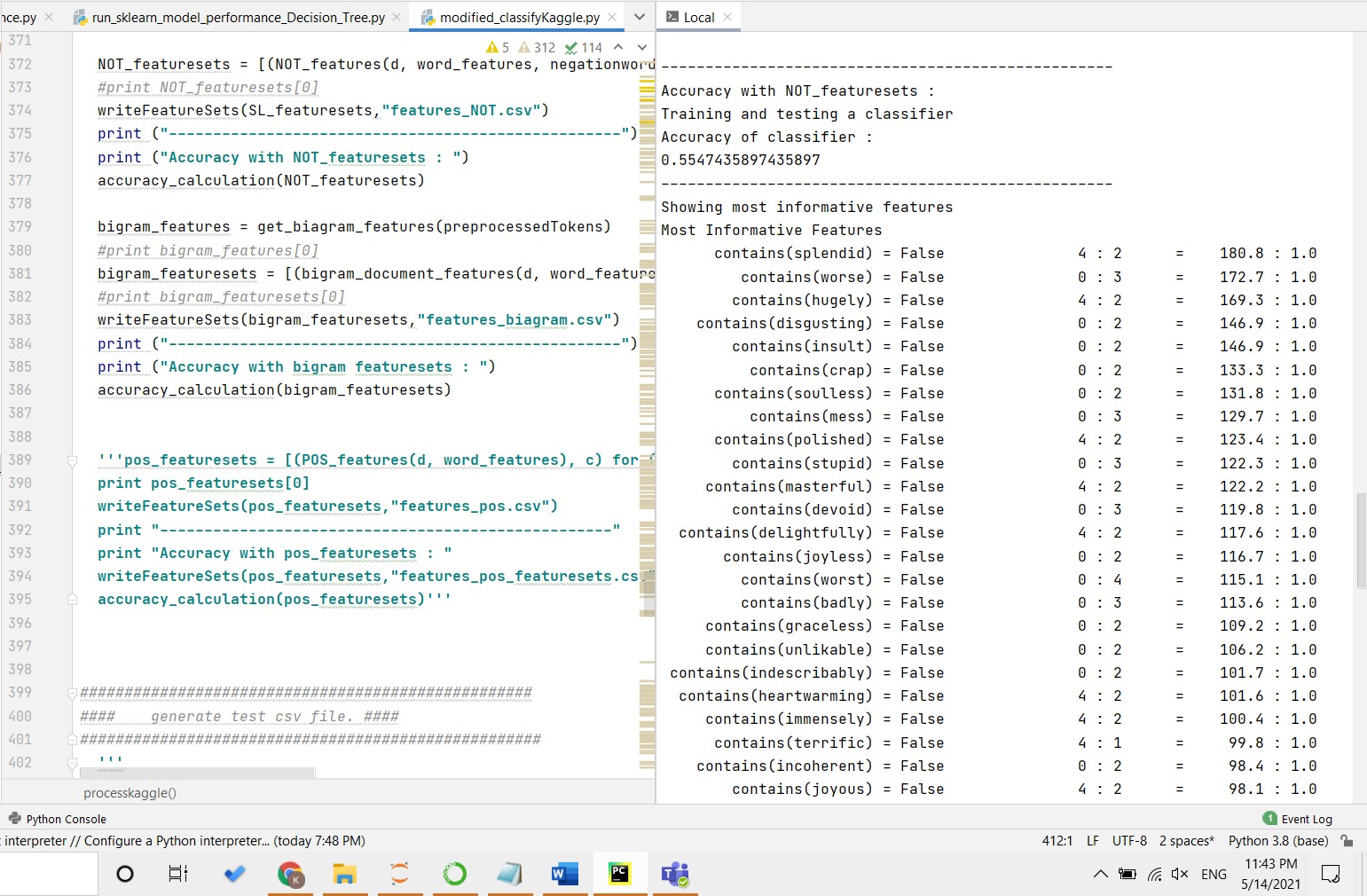
## Bigram Features



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Classification  Scale | Sentiment |  |
| 0 | Negative |  |
| 1 | Strong  Negative |  |
| 2 | Neutral |  |
| 3 | Positive |  |
| 4 | Strong  Positive |  |

From the above confusion matrix, it can be said that most of neutral sentiments and strong negative labels are predicted accurately for most of the training data and the least accurately predicted label is positive sentiment labels.

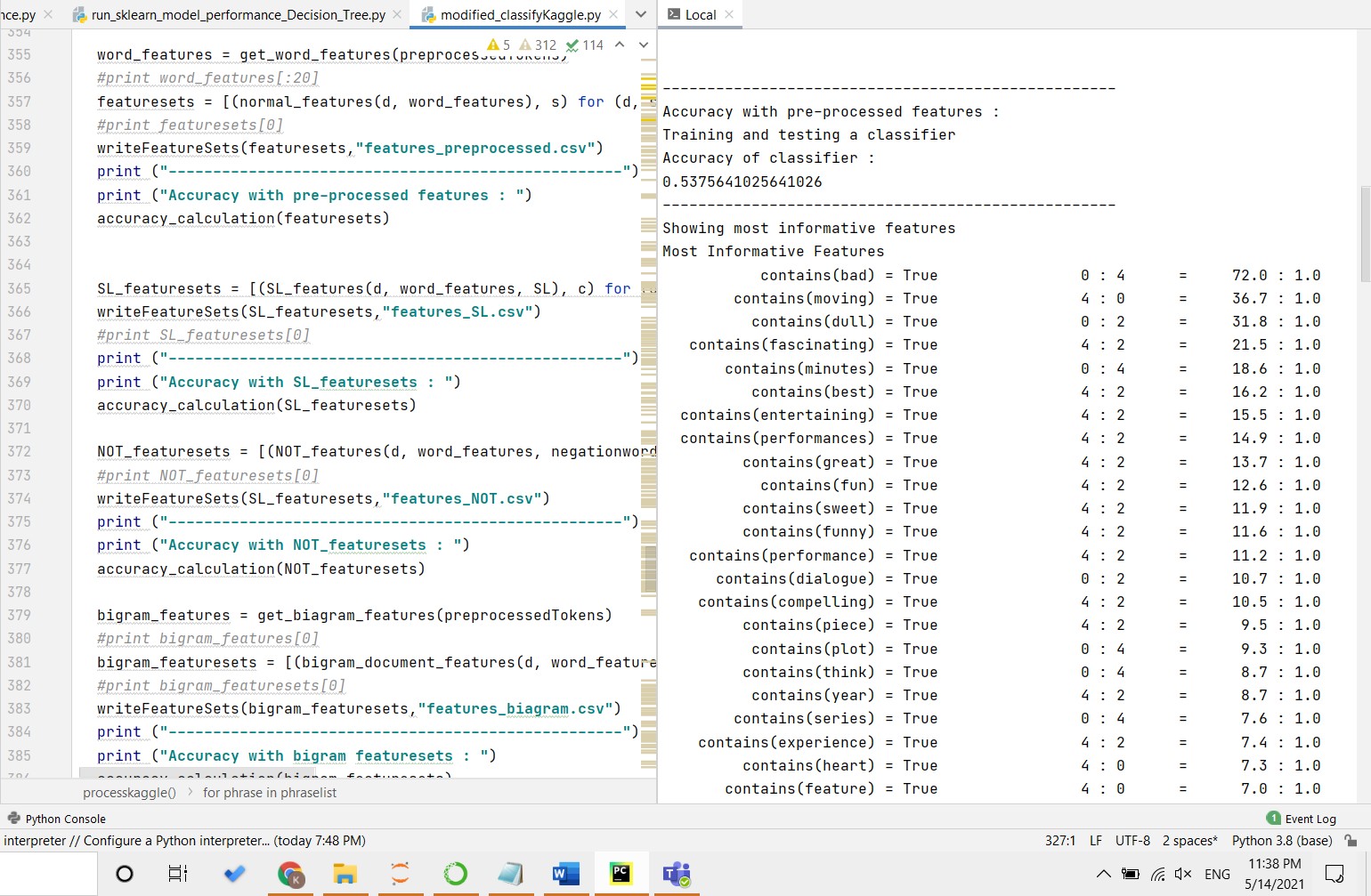
## Negation word Features



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Classification  Scale | Sentiment |  |
| 0 | Negative |  |
| 1 | Strong  Negative |  |
| 2 | Neutral |  |
| 3 | Positive |  |
| 4 | Strong  Positive |  |

From the above confusion matrix, it can be said that most of neutral and negative labels are predicted accurately.

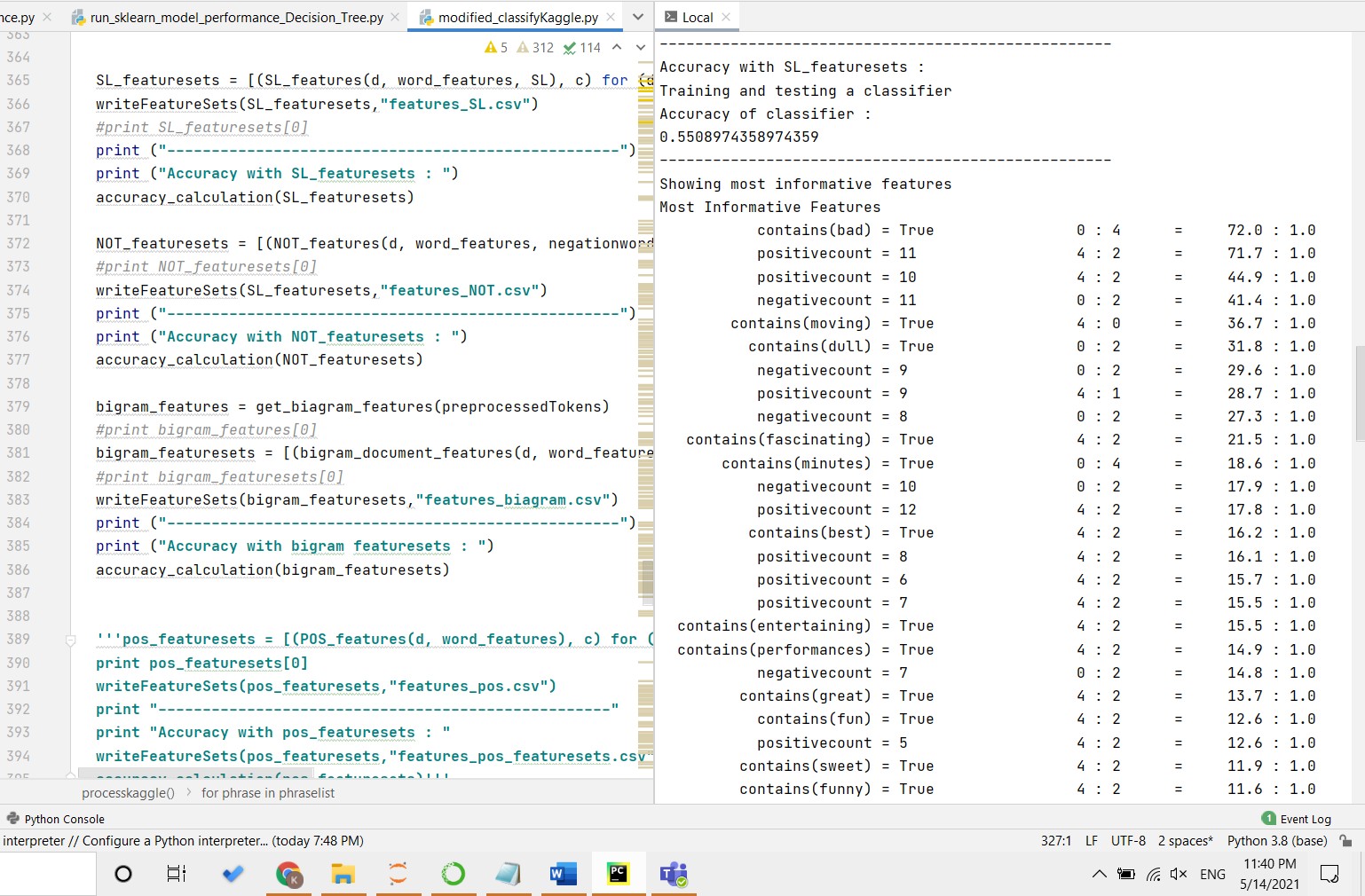
## Preprocessed Features



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Classification  Scale | Sentiment |  |
| 0 | Negative |  |
| 1 | Strong  Negative |  |
| 2 | Neutral |  |
| 3 | Positive |  |
| 4 | Strong  Positive |  |

From the above confusion matrix, neutral sentiment labels are predicted accurately.

## Sentiment Lexicon features

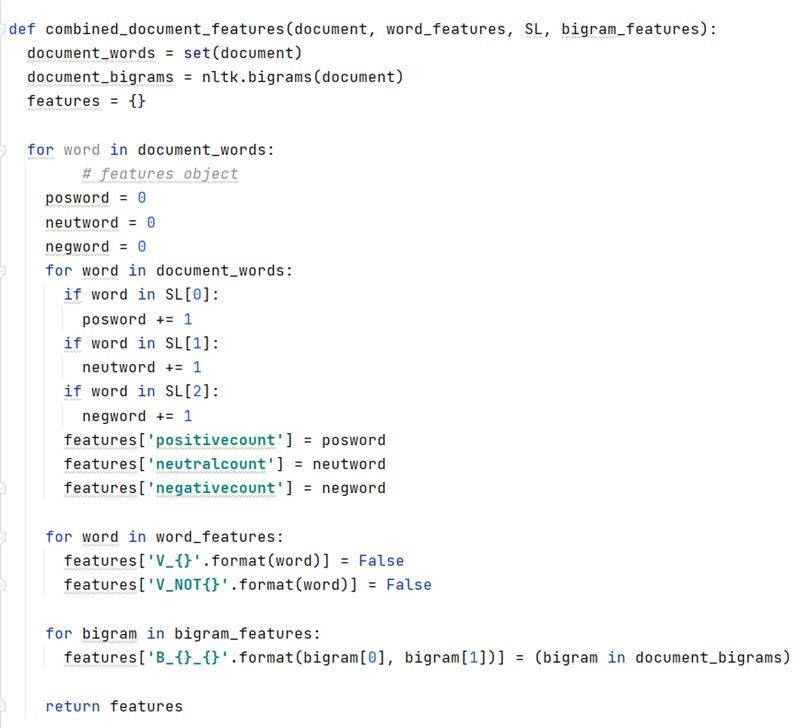


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Classification  Scale | Sentiment |  |
| 0 | Negative |  |
| 1 | Strong  Negative |  |
| 2 | Neutral |  |
| 3 | Positive |  |
| 4 | Strong  Positive |  |

From the above confusion matrix, neutral and positive sentiment labels are accurately predicted.

# Combination of features sets:

In this section, we will try to create a new function which will combine different feature sets such as Sentiment Lexicons, Bigram features and unigram features. The code follows,



From the above code we tried to create one single feature set which is a combination of bigrams and sentiment lexicons.



The accuracy we obtained for combined features is 80%. One of possible reasons for such high accuracy is because we are capturing the non-linearity in the data by creating and combining multiple features because of which the algorithm is maximizing learning.

The main constraint running this function is that the processing time to generate feature sets was time taking i.e. the execution time was high.

# Comparative Analysis of Logistic Regression and Decision Tree classifier:

In this section we will try to use Sci-kit learn algorithms to classify the sentiments. Since our outcome labels are like classification type of labels, we have decided to implement Logistic Regression and Decision tree classifier algorithms. Further we will run both the algorithms on each of the five feature sets.

Before delving into any details, we will do a comparative analysis of the metrics,

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Feature set type | Logistic Regression | | | Decision Tree Classifier | | |
| Precision | Recall | f1-score | Precision | Recall | f1-score |
| Normal Features without preprocessing | 0.49 | 0.47 | 0.46 | 0.47 | 0.53 | 0.42 |
| Preprocessed Features | 0.46 | 0.44 | 0.43 | 0.49 | 0.52 | 0.38 |
| Bigram Features | 0.46 | 0.44 | 0.43 | 0.49 | 0.52 | 0.38 |
| Negation word Features | 0.53 | 0.51 | 0.51 | 0.51 | 0.55 | 0.50 |
| Sentiment Lexicon features | 0.53 | 0.50 | 0.51 | 0.51 | 0.55 | 0.50 |

For Logistic regression classifier, Sentiment Lexicon features performance better in classification compared with other features functions because fewer words are unseen in train data as features. These words or tokens covers on Lexicon dictionary. Plus, we observed recall score lesser than F-measure which is greater than Precision.

For Decision Tree classifier, Sentiment Lexicon feature and Negation word features perform better in classification compared with other features functions. These words or tokens covers on Lexicon dictionary. Plus, we observed recall score higher than F-measure which is lesser than Precision.

Let us now discuss outputs for each of feature set for an algorithm,

## Logistic Regression:

The parameters used in logistic regression are,

* Class\_weight: Weights associated with classes in the form {class\_label: weight}. If not given, all classes are supposed to have weight one. The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n\_samples / (n\_classes \* np.bincount(y)).
* solver: Algorithm to use in the optimization problem.
* max\_iter: Maximum number of iterations taken for the solvers to converge.

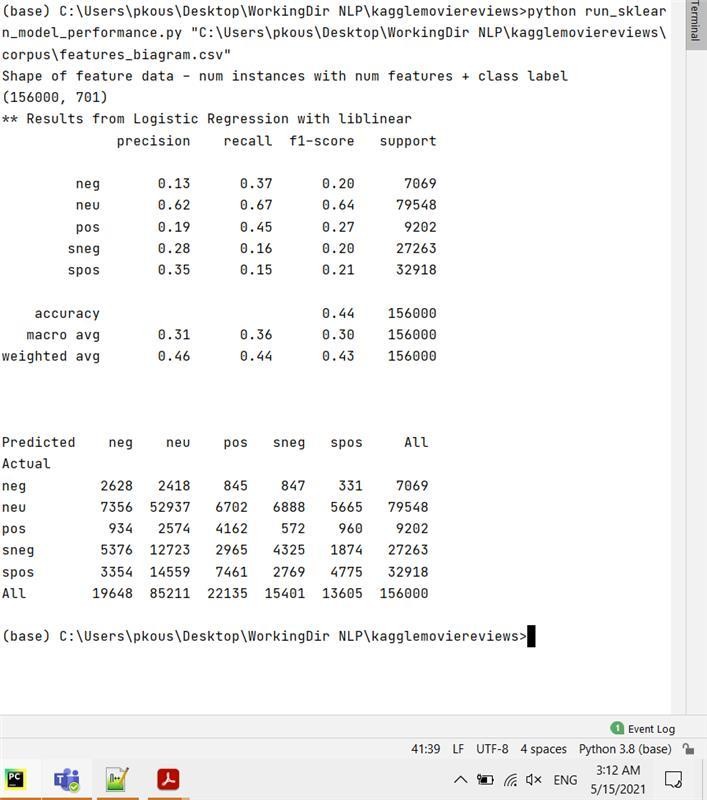
For our analysis, we have used the following parameters,

|  |  |
| --- | --- |
| Parameter\_Description | Parameter\_value |
| Class\_weight | Balanced |
| Solver | Lbfgs |
| max\_iter | 1000 |

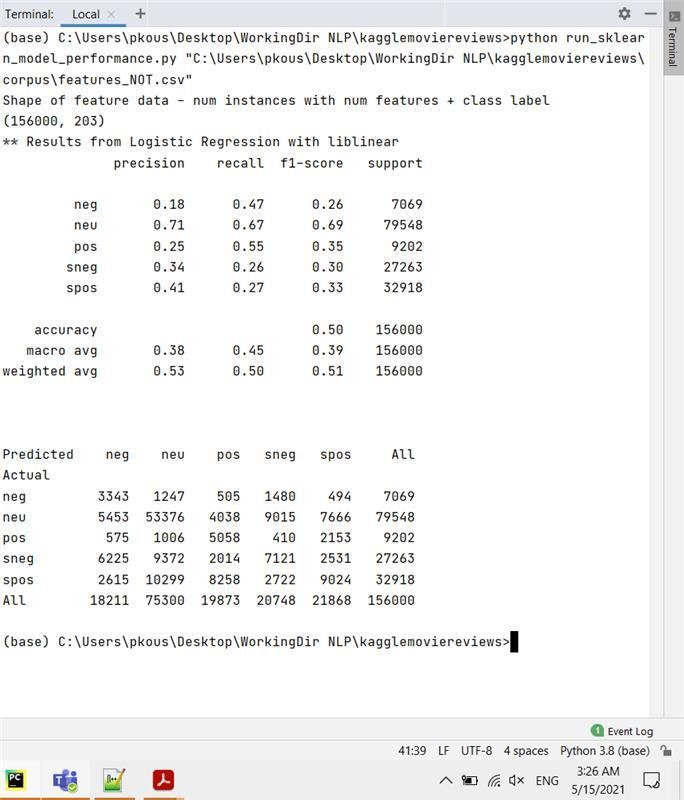
We have done experiments on different feature sets created using Logistic regression algorithm,

## Normal Features without preprocessing

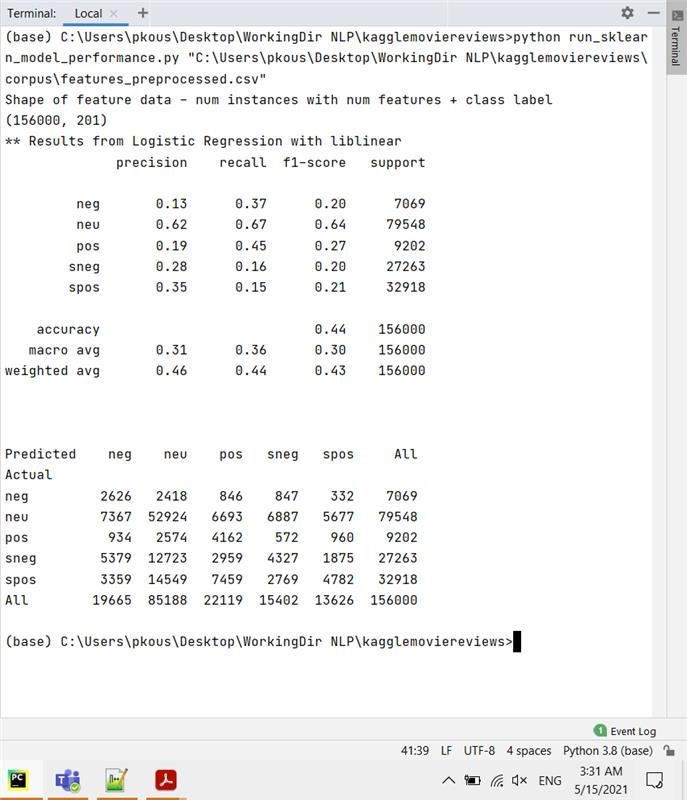
## Bigram Features



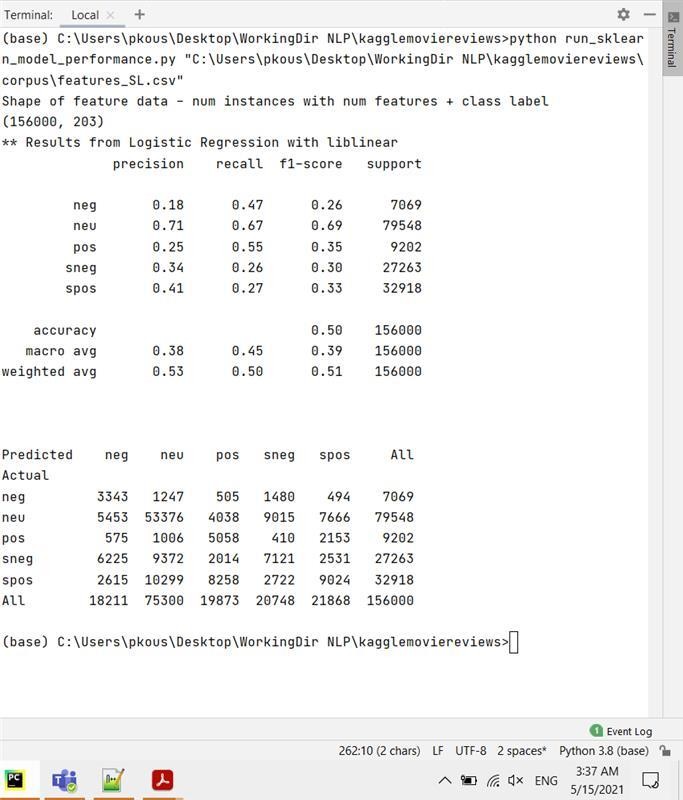
## Negation Word Features



## Preprocessed Features



## Sentiment Lexicon Features



## Decision Tree Classifier

The parameters used in Decision Tree Classifier are,

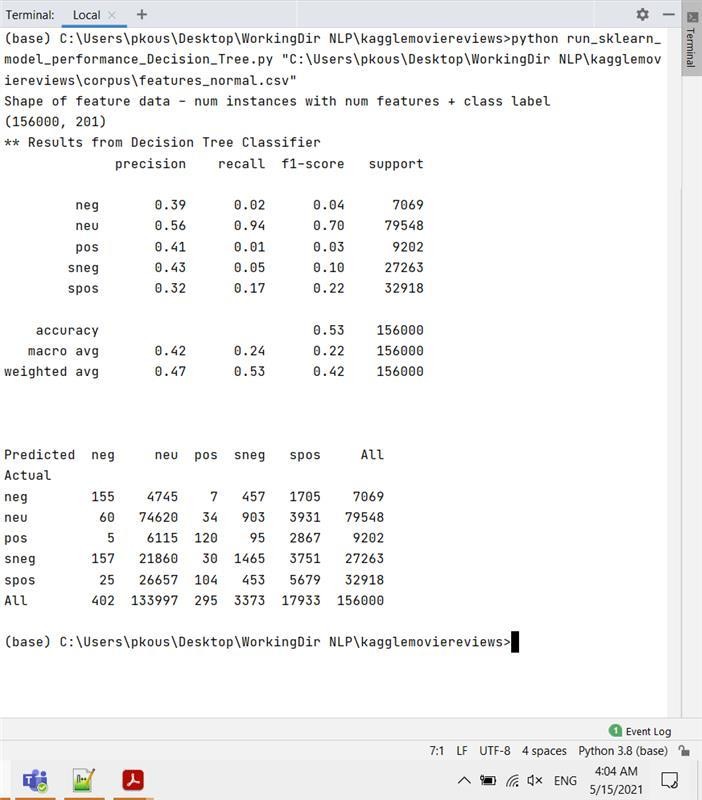
For our analysis, we have used the following parameters,

* + criterion: The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.
  + max\_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.
  + min\_samples\_split: The minimum number of samples required to split an internal node

|  |  |
| --- | --- |
| Parameter\_Description | Parameter\_value |
| criterion | gini |
| max\_depth | 7 |
| min\_samples\_split | 5 |

We have done experiments on different feature sets created using Logistic regression algorithm,

## Normal Features without preprocessing



## Bigram Features

## Negation Word Features

## Preprocessed Features

## Sentiment Lexicon Features

## Highlights of Sci-kit learn algorithms:

After running through different experiments, it can be said that Sentiment Lexicon feature sets work best when classifying the movie review datasets. The important aspect is that both logistic regression and decision tree classifier work similarly for Sentiment Lexicon features with highest precision, recall and f1- measures.

# Comparison of evaluation metrics:

Average subjectivity accuracy was 1.35% higher than its corresponding unigram feature set for un- processed tokens. Average subjectivity accuracy was 0.45% higher than its corresponding unigram feature set for pre-processed tokens. In fact, Logistic regression produced best accuracy using subjectivity feature set on pre-processed version.

Based on NLTK classifiers and Sci-kit learn algorithms outputs, highest precision, recall and f1-measures was obtained for Sci-kit learn algorithms. The reason for this might be because the logic for Naïve Bayes is based on naïve algorithm wherein each of the feature is take multiple times i.e., the train data has some reoccurrences of data because of which redundant learning is made by the Naïve Bayes algorithm however, this is not the case with Logistic regression and Decision tree classifier algorithms. Therefore, in our case Sci-kit learn algorithms work best.

# Conclusion:

The maximum accuracy attained is 70%, we obtained this accuracy for combined feature sets where we ran the Naïve Bayes classifier. After running through the entire analysis, we concluded that grouping the target variables into three categories such as Negative, Neutral and Positive would give more accuracy, however, changing the target variable composition is something which we need to consider in extreme situations.

# Lessons Learned:

* Learned how to combine and implement most of the concepts learned during the semester.
* Leveraged python file system to write reusable code. Gained experience in dealing with problems that arise from working with large datasets.
* Observed how combining various feature sets affects the accuracy of the model.

Learned how different machine learning models are implemented and how their outputs differ, and what causes these differences in output.