Kaggle Movie Reviews Classification

Michael Johnson

Garen Moghoyan

IST 664 Project

December 22, 2021

**Introduction**

[**Dataset**](https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews)**:**

While found on Kaggle, the Rotten Tomatoes movie review dataset is a corpus of movie reviews used for sentiment analysis that was originally collected by Bo Pang and Lillian Lee in 2005. The data consists of comments left by Rotten Tomatoes users that have been broken down as follows: each comment (phrase) gets assigned a SentenceId number and is then parsed word by word, receiving a different PhraseId with each parsing. The initial phrase is the most complete one and is therefore used in our analysis. The training dataset also contains a sentiment rating indicating the strength and polarity of each phrase. Those ratings are as follows:

0 - negative

1 - somewhat negative

2 - neutral

3 - somewhat positive

4 - positive

**Goals:**

The purpose of this project is to accurately classify movie reviews based on the sentiment ratings for each review. Achieving this goal requires building multiple classification algorithms, applying those to text processed in a variety of ways, and comparing the results to find the best algorithm and preprocessing technique. The outcome results from this analysis will include the best document grouping, the best featureset, and the best model type to accurately classify the movie reviews.

To achieve these goals, the movie review corpus went through three primary steps:

1. Text Processing
2. Feature Engineering
3. Experiments

**Step 1: Text Processing**

**Processing the Text:**

Processing the Kaggle Movie Reviews corpus and preparing it for tokenizing, filtering, and pre-processing involved splitting the reviews up into two separate documents. The first document included grouping the data by sentence ID and retaining just the complete reviews with sentiment ratings instead of having them broken down into smaller pieces. The second document uses the entire dataset with each of the broken out reviews and their associated sentiment ratings. The comparison between both documents is the foundation of this analysis and will drive the comparison between each of the featuresets built using the documents.

**Tokenizing the Text:**

Tokenization process for each of the documents is the primary analysis and will serve as the baseline for comparison to the filtered documents, the pre-processed documents, and a combination of filtering and pre-processing. The tokenization function (called process\_text) imports the document lists created in the processing section and performs four primary functions on each of the reviews that will remain constant throughout the filtering and pre-processing steps. After pulling in the document, the tokenizer uses the NLTK tokenize function, turns all words into lowercase, filters out a list of punctuations, and appends the resulting tokens with their sentiment ratings to a list. This is the baseline comparison to see if filtering the documents and pre-processing the documents holds any bearing for the classification models.

**Filtering the Text:**

Filtering text documents is the first comparison to only tokenizing the documents and includes all the same tokenization functions with additional features built in to filter the documents. The first function removes all non-alphabetic characters, relying on pattern matching using regular expressions with the following syntax:  
  
pattern = re.compile(r'^[^a-z]+$')

The next function in filtering the data includes removing words under two letters in length as a majority of the words would be filtered out during the next step. This removes basic words like “on” and “a” as part of preparing a smaller list of stopwords needed. The final filtering step includes removing all the base stopwords in the NLTK stopwords list, as well as additional stopwords defined by their common usage and lack of importance based on the frequency of showing up in the documents. The following is a list of the NLTK stopwords and the additional words filtered from the documents:

NLTK Stopwords: ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your','yours',

'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'hers',

'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',

'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are',

'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does',

'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until',

'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',

'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',

'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here',

'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',

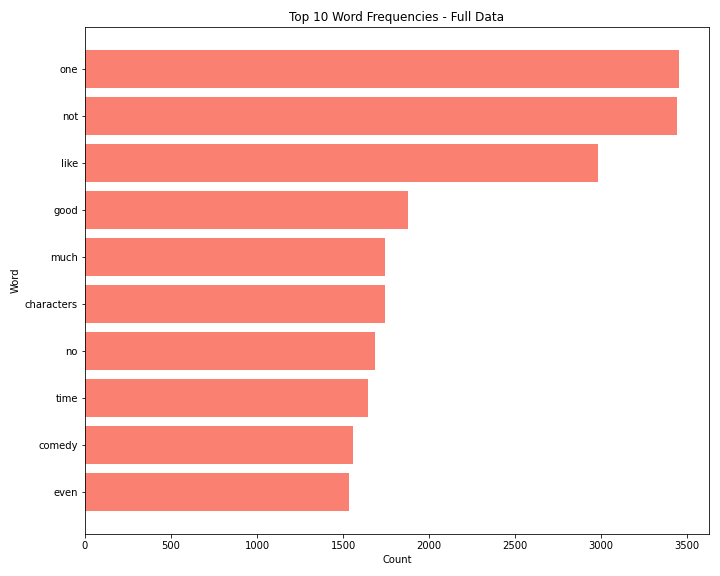
'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',

'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now']

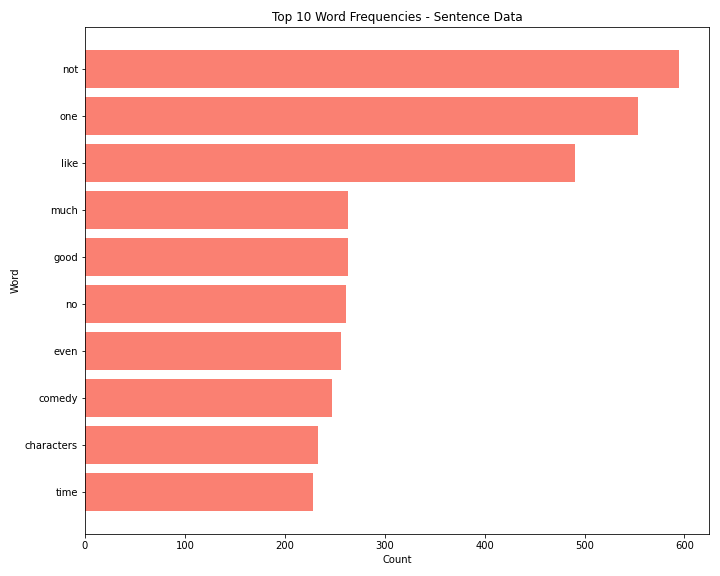
Additional Stopwords: ['can','has','have','had','must','shan','do', 'should','was', 'were','won','are','cannot','does','could','did','is','might','need','would', "'s", 'film', 'movie', 'story', '-rrb-', '-lrb-',"'re", "n't"]

Note: any stopwords that fell into the negation words list were removed from the filtering to allow for comparing negative featuresets to the rest of the data.

Top 10 word frequencies for all data:



Top 10 word frequencies for just sentence data:



Although the sentence data uses complete reviews and the full data has the split out reviews, it makes sense for the frequencies to align between the two. The sentence data has significantly less frequencies than the full data and actually flips the first and second most common words as well as a couple other words throughout the text. This signals the model performance between datasets could align in terms of rankings but the sentence-only data should have much lower confidence levels due to the reduced word count.

**Pre-processing the Text:**

Pre-processing text documents is the second comparison to only tokenizing the documents and includes all the same tokenization functions with additional features built in to filter the documents. The first function includes stemming the words using the Porter stemmer from NLTK. The Lancaster stemmer was also tested but removed from the pre-processing due to the aggressive stemming observed in the documents. As a more gentle stemming function, the Porter algorithm was also preferred due to being the standard stemming algorithm used for most NLP tasks. Stemming involves chopping off the ends of the words and was used first in this analysis to prepare the data for lemmatization. Using the NLTK lemmatizer, the words were lemmatized to return only the base form of a word.

**Pre-processing and Filtering the Text:**

The final point of comparison to the tokenized documents is a combination of both filtering and pre-processing. This section repeats the same functions observed in the filtering and pre-processing sections, while ensuring that words filtered out do not end up in the end documents.

**Final Text Processing Experiments:**

Experiments will run against two primary datasets: all data available and only the first line of each review containing the full sentence.

The following is a breakdown of the planned experiments based on all the data:

* All tokenized data
* All filtered and tokenized data
* All pre-processed and tokenized data
* All pre-processed, filtered, and tokenized data

The following is a breakdown of the planned experiments based on only full sentences:

* Full tokenized sentences
* Full filtered and tokenized sentences
* Full pre-processed data and tokenized sentences
* Full pre-processed, filtered, and tokenized sentences

The eight final text processing documents listed above will go through each of the nine feature engineering stages, resulting in a total of 72 varying featuresets compared during the experiments stage. Although there are quite a bit of featuresets to compare, the end result should meet the goals listed within the introduction section with adequate comparison between the different text processing techniques. Each of the documents were randomized using the same seed number to reproduce the experiments so all processing started with the same documents. To save on performance, only the first 1,000 reviews of the randomized documents will go through the feature engineering and modeling.

Note: this method leaves room for introducing additional documents for comparison. Each of the functions within both the filtering and the pre-processing could be split out to identify the best document processing for this dataset. The individual filtering and pre-processing steps outlined above and their combinations would result in hundreds of featureset to test and model.

**Step 2: Feature Engineering**

The goal in this stage is to produce the featuresets by passing the processed documents through functions that identified word features, bigram features, and additional word similarities captured in various lexicons and outputting the result into dictionaries. Each dictionary consists of a word or bigram with a true or false rating and the sentiment number for the tokenized review. As mentioned previously, the 8 documents from the processing stage all go through the feature engineering stage for a total of 72 featuresets. Each of the featuresets will pass through the modeling stage to find the best combination of processing and feature engineering for the data.

The feature engineering functions all rely on the same word features function to ensure the same word count is used for each of the featuresets. The function defines a list of all the words and sentiments in each document, finds the frequency of the words in the reviews, and uses the top 1,000 most common words to produce the word features.

**Bag of Words/Unigrams Features:**

The BoW approach consists in collecting all the words in the corpus and asserting two things: the vocabulary of the known words and a measure of the presence of the known words. This approach allows us to get a count of words as well as their frequency. The resulting featureset contains the word and a true or false rating to capture whether the word was in the word features list. The featureset also contains the sentiment rating for the review to serve as the classification point for modeling. The following output is the first 10 words of the first review within the bag of words featureset using token processing only:

[{'V\_the': False,

'V\_a': True,

'V\_and': True,

'V\_of': False,

'V\_to': False,

"V\_'s": False,

'V\_that': False,

'V\_in': False,

'V\_is': False,

'V\_it': False},0]

**Bigrams Features:**

Similar to the unigram count, the bigram feature allows us to get a count and frequency of two word combinations that occur in the corpus. The resulting featureset contains all the bigrams within each review and a true or false rating for each bigram of the review to capture whether the bigrams within the document were in the bigram word features list with the top 1,000 most common bigrams. The featureset also contains the sentiment rating for the review to serve as the classification point for modeling. The following output is the first 10 words of the first review within the bigram featureset using token processing only:

[{'B\_1\_drips': False,

'B\_102-minute\_infomercial': False,

'B\_94-minute\_travesty': False,

'B\_a.s.\_byatt': False,

'B\_accurately\_reflects': False,

'B\_across\_america': False,

'B\_act\_abroad': False,

'B\_acted\_skateboards': False,

'B\_added\_clout': False,

'B\_adding\_flourishes': False},0]

**Bigrams and Bag of Words Features:**

The resulting featureset contains both the bigrams and bag of words unigrams within each review and a true or false rating for each bigram and unigram of the review. This captures whether the bigrams and unigrams within the document were in the bigram and unigram word features list with the top 1,000 most common bigrams and unigrams. The featureset also contains the sentiment rating for the review to serve as the classification point for modeling. The following output is the first 10 words of the first review within the bigram and bag of words featureset using token processing only (note: bigrams are also part of the featureset, they are applied at the end):

[{'V\_the': False,

'V\_a': True,

'V\_and': True,

'V\_of': False,

'V\_to': False,

"V\_'s": False,

'V\_that': False,

'V\_in': False,

'V\_is': False,

'V\_it': False},0]

**Part-of-speech Features:**

The PoS process consists in categorizing words of the corpus in correspondence with a particular part of speech, depending on the definition of the word and its context. The various elements of the corpus get tagged as nouns, verbs, adjectives, adverbs, etc. The resulting featureset contains all the words within each review and a true or false rating for whether the word is in the part-of-speech tagging word features list with the top 1,000 most common words. The featureset also contains the sentiment rating for the review to serve as the classification point for modeling. The following output is the first 10 words of the first review within the part-of-speech featureset using token processing only:

[{'contains(the)': False,

'contains(a)': True,

'contains(and)': True,

'contains(of)': False,

'contains(to)': False,

"contains('s)": False,

'contains(that)': False,

'contains(in)': False,

'contains(is)': False,

'contains(it)': False},0]

**Negative Features:**

This feature lists words with a negative connotation (such as “no”, “not”, ”never”) but also approximate negators like “hardly” and “rarely”. The negative feature involves two approaches: one is to negate the word following the negation word and the other is to negate all words following the negation word up to the next punctuation mark.The resulting featureset contains all the words within each review and a true or false rating for whether the word is in the negative word features list with the top 1,000 most common words. This feature set also contains the unigrams and their associated true or false rating based on the word features. The featureset also contains the sentiment rating for the review to serve as the classification point for modeling. The following output is the first 10 words of the first review within the negative featureset using token processing only:

[{'V\_the': False,

'V\_NOTthe': False,

'V\_a': True,

'V\_NOTa': False,

'V\_and': True,

'V\_NOTand': False,

'V\_of': False,

'V\_NOTof': False,

'V\_to': False,

'V\_NOTto': False},0]

**Subjectivity Lexicon:**

Subjectivity refers to expression of opinions, evaluations, feelings and speculations (thus incorporating sentiment). The subjectivity lexicon therefore includes clues relating to these perceptions usually grouped as predefined lists of words associated with the emotional context (such as positive/negativeO

[{'V\_the': False,

'V\_a': True,

'V\_and': True,

'V\_of': False,

'V\_to': False,

"V\_'s": False,

'V\_that': False,

'V\_in': False,

'V\_is': False,

'V\_it': False},0]

**Sentiment Lexicon (LIWC):**

Linguistic Inquiry and Word Count is a text analysis program that calculates the degree to which various categories of words (such as positive or negative emotions, self-references or causal words) are used in a text. The resulting featureset contains all the words within each review and a true or false rating for whether the word is in the LIWC lexicon positive or negative word features list with the top 1,000 most common words. The featureset also contains the sentiment rating for the review to serve as the classification point for modeling. The following output is the first 10 words of the first review within the LIWC featureset using token processing only:

[{'contains(the)': False,

'contains(a)': False,

'contains(and)': False,

'contains(of)': False,

'contains(to)': False,

"contains('s)": False,

'contains(that)': False,

'contains(in)': False,

'contains(is)': False,

'contains(it)': False},0]

**Subjectivity and Sentiment Lexicon (LIWC):**

The resulting featureset contains both the words that were in the LIWC lexicon and the subjectivity lexicon with a true or false rating for each word of the review. This captures whether the words within the document were in the top 1,000 most common word features built using the respective lexicons. The featureset also contains the sentiment rating for the review to serve as the classification point for modeling. The following output is the first 10 words of the first review within the Subjectivity and LIWC featureset using token processing only:

[{'contains(the)': False,

'contains(a)': False,

'contains(and)': False,

'contains(of)': False,

'contains(to)': False,

"contains('s)": False,

'contains(that)': False,

'contains(in)': False,

'contains(is)': False,

'contains(it)': False},0]

**Additional Lexicon:**

The Harvard IV (4) Inquirer was also used, it consists of various categories of words expressing specific concepts, some of which are feelings (pleasure and pain), social categories (race, kinship), objects (tools, food, vehicle), and cognitive abilities like knowing, thinking or problem solving.The resulting featureset contains all the words within each review and a true or false rating for whether the word is in the Harvard IV-4 lexicon with positive or negative and strengths for each word in the word features list within the lexicon. The featureset also contains the sentiment rating for the review to serve as the classification point for modeling. The following output is the first 10 words of the first review within the Harvard IV-4 Inquirer featureset using token processing only:  
  
[{'V\_the': False,

'V\_a': True,

'V\_and': True,

'V\_of': False,

'V\_to': False,

"V\_'s": False,

'V\_that': False,

'V\_in': False,

'V\_is': False,

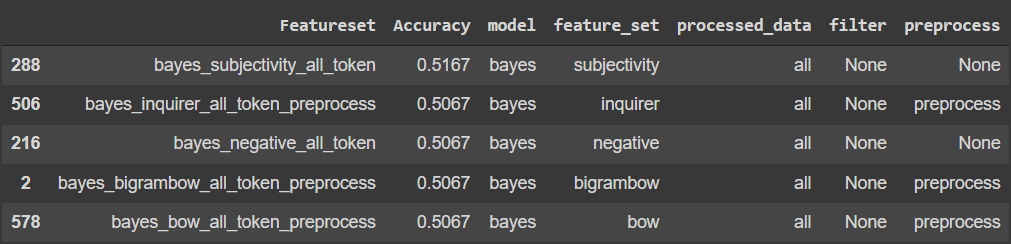
'V\_it': False},0]

**Step 3: Experiments**

**Naive Bayes Classifier:**

Based on Bayes theorem, the Naive Bayes classifier builds a model by establishing relationships between features in a very general way; it works in a supervised manner by predicting a test outcome based on a training dataset. This classifier is a probabilistic classifier which means that given an input, it predicts the probability of the input being classified for all the classes. It is also called conditional probability.

Top 5 Results:

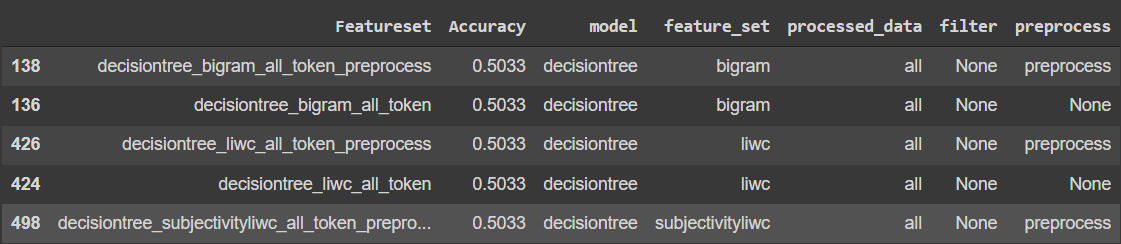


With the Naive Bayes classifier, we had a 0.5167 accuracy using the subjectivity featureset with all data and no filtering or pre-processing.

**SKLearn Classifiers:**

**Decision Tree:** it functions by breaking down a dataset into smaller and smaller subsets based on different criteria. Many decision trees linked together result in a Random Forest.

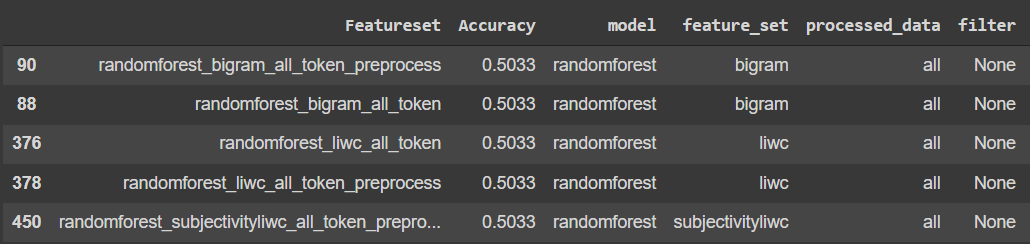
Top 5 Results:



The Decision Tree model gave us a 0.5033 accuracy using the bigram featureset with all data and no filtering or pre-processing.

**Random Forest:** as mentioned above, it is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of the dataset.

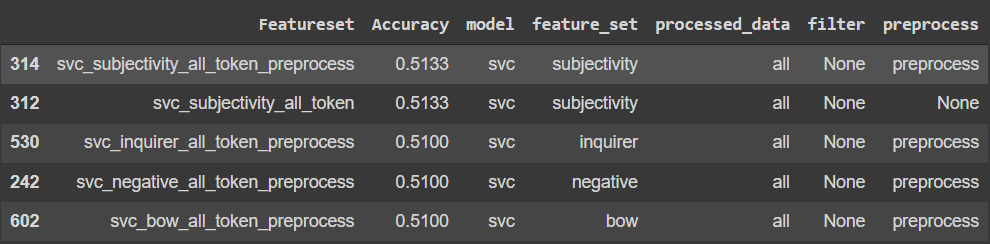
Top 5 Results:



Using the Random Forest classifier with a bigram featureset, all data and no filtering, the accuracy we obtained was 0.5033.

**SVC:** the Support Vector Classifier (SVC) works by drawing a line between the different clusters of data points to group them into classes. To increase confidence in the class allocation, the classifier will try to maximize the distance between the line it draws and the points on either side of it.

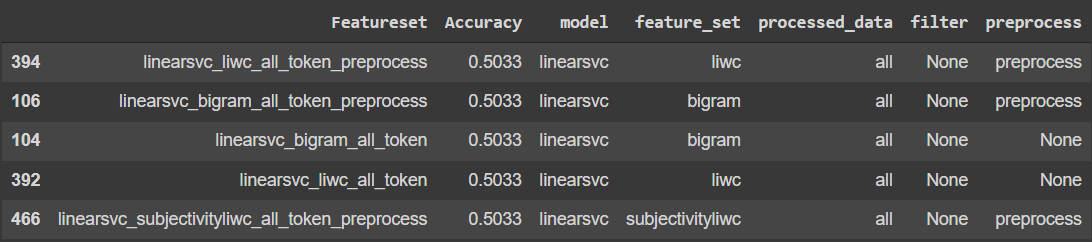
Top 5 Results:



With the SVC model, using the subjectivity featureset, all data, no filtering and pre-processed data, we got an accuracy of 0.5133.

**Linear SVC:** this method applies a linear kernel function to perform classification and it performs well with a large number of samples. If we compare it with the SVC model, the Linear SVC has additional parameters such as penalty normalization which applies 'L1' or 'L2' and loss function. The kernel method can not be changed in linear SVC, because it is based on the kernel linear method.

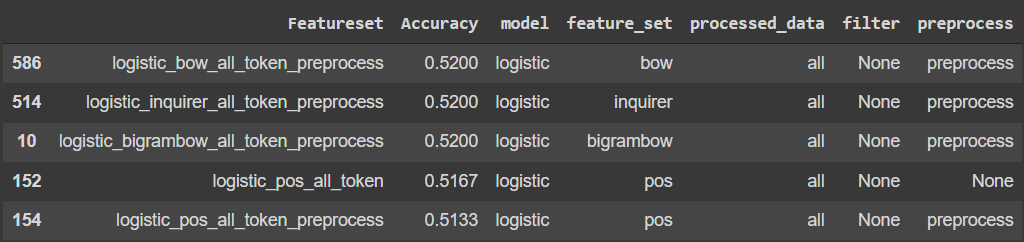
Top 5 Results:



With pre-processed data, no filter and the LIWC featureset, the linear SVC model gave us a 0.5033 accuracy.

**Logistic Regression:** used for binary classification tasks (yes/no, spam/not spam) the logistic regression picks a threshold and decides where the selected value falls on either side of it.

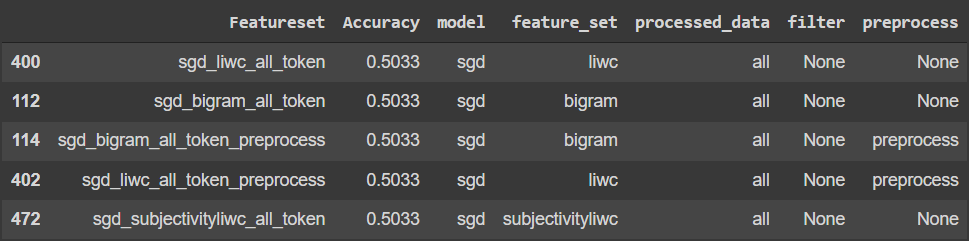
Top 5 Results:



With the Logistic Regression classifier, we had a 0.52 accuracy using the BoW featureset with all data and no filtering but with pre-processing.

**SGD:** the Stochastic Gradient Descent approach consists in fitting linear classifiers and regressors under convex loss functions by implementing a plain stochastic gradient descent learning routine.

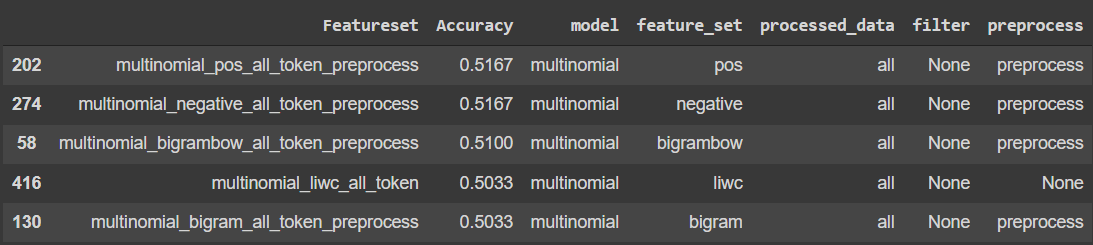
Top 5 Results:



With no pre-processed data, no filter and the LIWC featureset, the SGD model gave us a 0.5033 accuracy.

**Multinomial Naive Bayes:** this classifier guesses the tag of a text, such as an email or a newspaper story, using the Bayes theorem. It calculates each tag's likelihood for a given sample and outputs the tag with the greatest chance.

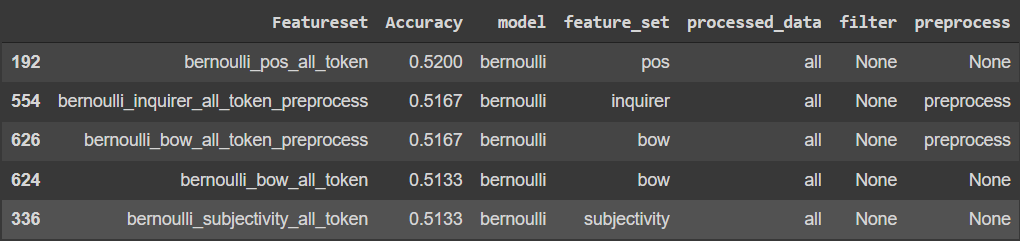
Top 5 Results:



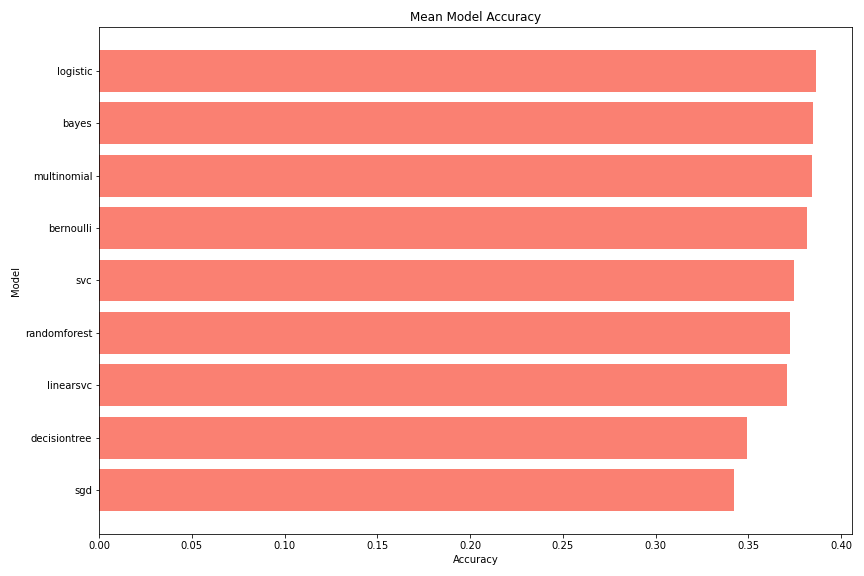
With the Multinomial Naive Bayes classifier, we had a 0.5167 accuracy using the PoS featureset with all data and no filtering or pre-processing.

**BernoulliNB:** a Naive Bayes classifier, based on the Bernoulli distribution, that’s used for discrete data, where features are only in binary form.

Top 5 Results:

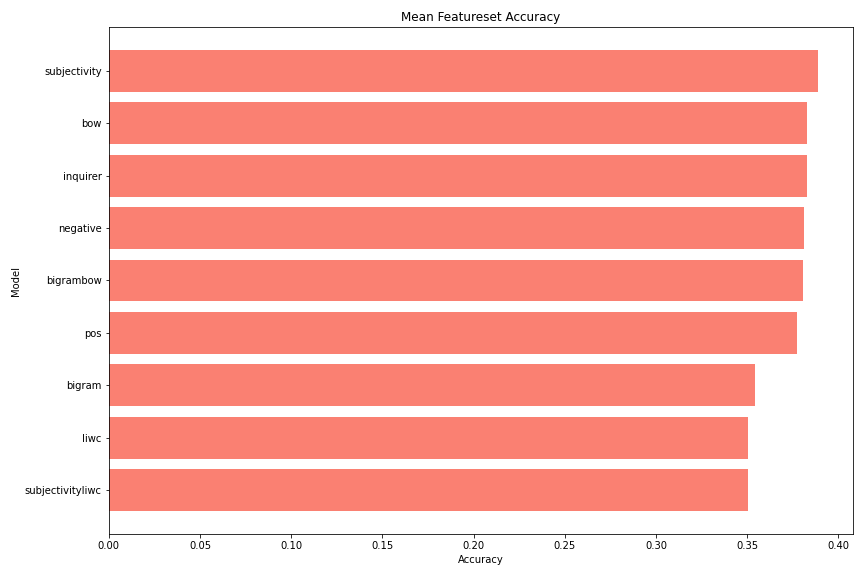


With the Bernoulli Naive Bayes classifier, we had a 0.52 accuracy using the PoS featureset with all data and no filtering or pre-processing.

**Model Comparison:**

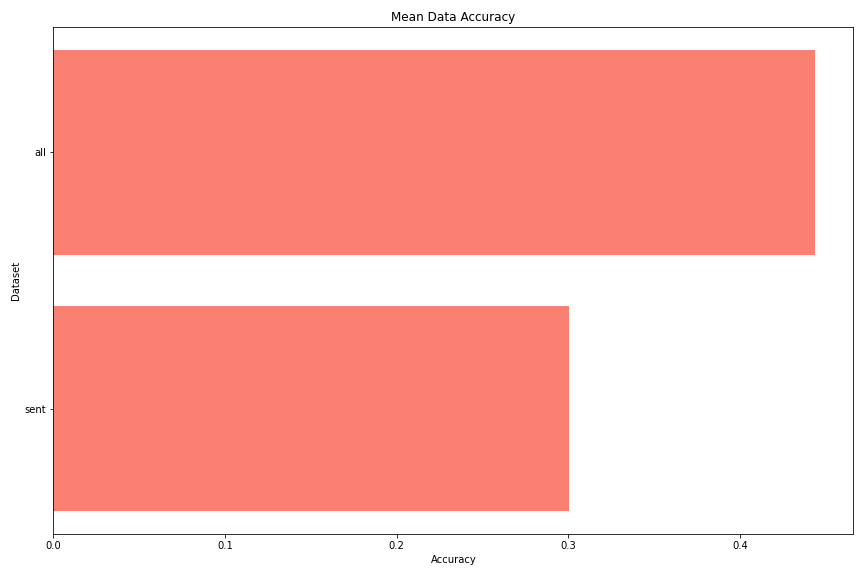
In a model comparison, the logistic regression model (0.386) barely edged the three Bayesian models - Naive Bayes (0.385), Bernoulli NB (0.384) and multinomial NB(0.382).

**Featureset Comparison:**

****

In featureset comparison, the subjectivity model has an accuracy of 0.389, better than the 0.383 accuracies of the BoW and Harvard inquirer.

**Dataset Comparison:**

****

In a comparison of the datasets, the ‘all’ dataset had a far better accuracy (0.44) than the ‘sent’ dataset (0.3).

**Cross-Validation for Top 3 Models and Featuresets:**

Cross-validation: it is a technique used to protect against overfitting in a predictive model, particularly in a case where the amount of data may be limited. In cross-validation, you make a fixed number of folds (or partitions) of the data, run the analysis on each fold, and then average the overall error estimate.

Precision is a measure of how many of the positive predictions made are correct (true positives), accuracy describes the number of correct predictions over all predictions, recall is a measure of how many of the positive cases the classifier correctly predicted over all the positive cases in the data and the F1-score is a measure combining both precision and recall that basically weighs the two rations in a balanced way.

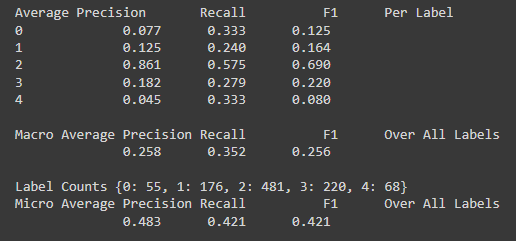
**The best model and dataset:**

Model: Logistic regression

Featureset: Bigram BOW

Data: All data

The Cross Validation results:



The cross validation results indicated a macro average precision of 0.258, a recall of 0.352, and a F1 score of 0.256 for the best model (logistic regression) and featureset (BoW bigram) combination. Although this model and featureset combination was the best in accuracy, the precision and F1 scores were not the best out of the three.

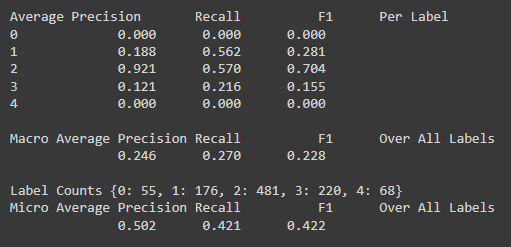
**Second best model and dataset:**

Model: Bernoulli

Featureset: PoS

Data: All data

The Cross Validation results:



The cross validation results indicated a macro average precision of 0.246, a recall of 0.270, and a F1 score of 0.228 for the second best model and featureset combination. The precision and recall scores came in the worst out of the three models and featuresets, with a significantly worse recall score than the other two models and featuresets.

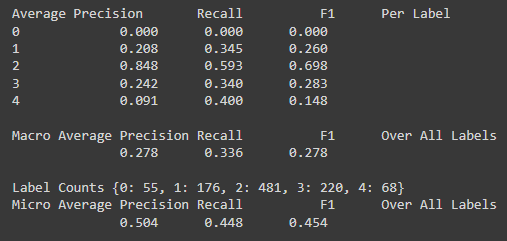
**Third best model and dataset:**

Model: Logistic regression

Featureset: Harvard Inquirer IV-4

Data: All data

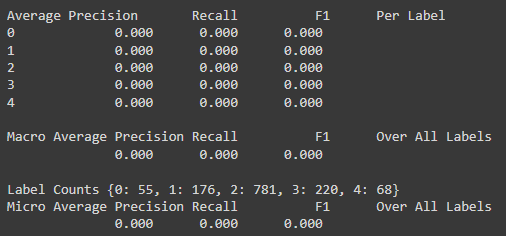
The Cross Validation results:



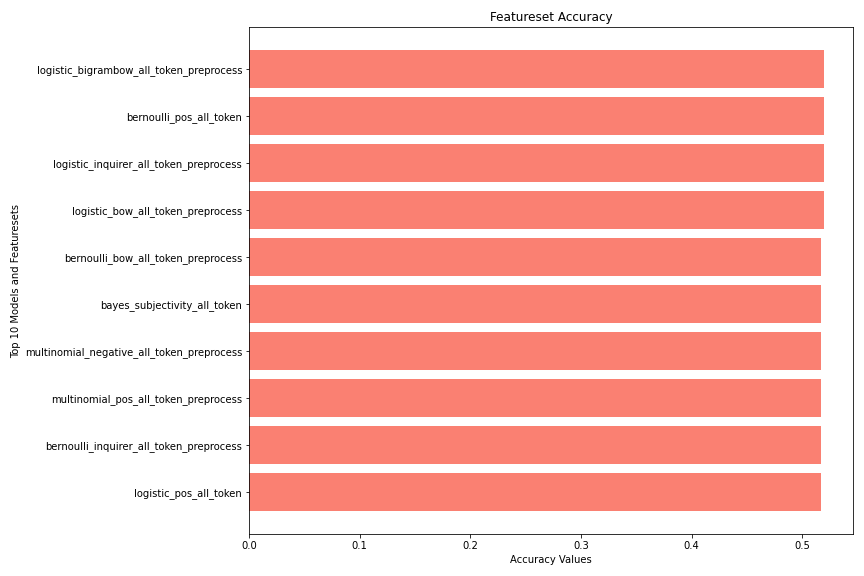
The cross validation results indicated a macro average precision of 0.278, a recall of 0.336, and a F1 score of 0.278 for the third best model and featureset combination. Although this model and featureset was the third best in accuracy, the precision and F1 scores were the best out of the three.

**Cross-Validation for Testing Data:**

The cross validation scores resulted in 0 for each of the three scores when training the logistic model with the best featureset and trying to predict the testing featureset. With every rating having a sentiment of 2 for the testing data, the model trained on a variety of sentiment scores using the training data could not accurately predict the sentiment score of 2 for the testing data given the bigram and BoW featureset built using the testing data.



**Conclusion**



The findings from this analysis highlight the difficulties in classifying movie reviews based solely on the text alone. With the model results, the highest accuracy was 52 percent and resulted in a four-way tie. Each of the top four models relied on processing the full data and three out of the four benefited from the use of pre-processing with the fourth not requiring any pre-processing or filtering. The top featuresets for the model outputs were the bigram BoW, PoS, Inquirer, and the BoW. Logistic regression came out as the top model, holding three out of the top four spots with Bernoulli being the last. As hypothesized in the introduction, the full dataset resulted in higher accuracy scores with only using the primary sentence with each review showing abysmal results nearly 20 percent lower.The top three featuresets by average model accuracies resulted in subjectivity being the best, followed by BoW and Inquirer.

Both of us worked together to produce each of the results in this report, leaning on Google Colab for the Python portion and Google Docs for drafting the report.