Morgan Gere

NLP-IST664

06/22/2022

Final Project

Kindle Sentiment Analysis

**Intro**

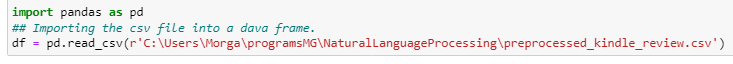
To gain a firm understanding of sentiment analysis using NLTK and Sklearn, both were put to the test against one another to determine how well they preform along with the pros and cons of each. To do that a Kaggle data set was acquired that contained reviews of books on the amazon kindle. (<https://www.kaggle.com/datasets/meetnagadia/amazon-kindle-book-review-for-sentiment-analysis?select=preprocessed_kindle_review+.csv>) The data consisted of 12,000 reviews that were comprised of text and a rating 1-5. The goal was to first try getting a vector with features obtain though NLTK and then use the SKlearns vectorization to replicate the same features and compare accuracy, precision, recall, and f1-scores. It was also prudent to discover the ease of feature creation when these options were used. After the initial NLTK vs Sklearn more vectorization options were tested using only the sklearn vectorization.

The next stage new feature set engineering was implemented to create multiple new features ranging from simple, to complex to see how they affected the accuracy of the models created with both the NLTK and sklearn feature generation. Multiple new features were then included into two models (one NLTK generated with additional features one Sklearn with additional features) accuracy, precision, recall and f1-score were obtained and then compared with the original models.

**Method**

Input and Preprocessing

The Data set was read in using pandas directly into a pandas data frame.



The sum of null values was found for each column in the data frame. There were no null values found.

Text, table

Description automatically generated

To see the number of each example for each label type value counts was used. To determine raw agreement from this the largest group was divided by the sum of all the examples. One divided by the number of labels gives the baseline.

Graphical user interface, text, application, email

Description automatically generated Graphical user interface, text, application

Description automatically generated

The review text and the rating were placed into lists.

Text

Description automatically generated with low confidence

Lastly, NLTK Vader sentiment was used to obtain a compound score for each review. A loop was used to get a prediction if the score was found to fall in predetermined ranges. The accuracy of Vader’s prediction was found to use as a higher standard of comparison.

Text

Description automatically generated

Since this is a review written by people to fix words with multiple character repeats like “whyyyyyyy” a regular expression was created that takes any word with multiple character repeats and replace the number of repeats with 2, so the “whyy”. This helped reduce the size of vocabulary and make them a new token as the individuals using them wanted to emphasis these words and add extra sentiment to them.

Text

Description automatically generated with low confidence

To use the NLTK feature extractor function given in lab the data needed to be in a specific format. The format was a list of tokenized words contained in a tuple with the other portion of the tuple being the label. A list containing lists of the tokens of each review was created using a loop. A function to merge each corresponding item from two sperate lists and place them into a tuple was created and used to combine the list containing the list of reviews in token form with its corresponding label obtained previously.

Graphical user interface, text, application, email

Description automatically generated

NLTK BOW

A list of all the tokens was obtained using code from the lab. This could have been obtained through other measures but since the list of tuples structure was needed later and created using the same method as the lab saved time. The list of tokens was passed through nlrk’s FreqDist to obtain a count of the tokens the top 2000 were selected and placed into a list along with that count. The words were extracted and placed into a list.

A screenshot of a computer

Description automatically generated with medium confidence

A function was created that takes two lists of strings as input. The function checks if the strings in the second list are present in the first list. The string has a “V\_” attached to the front and is placed as a key in a dictionary if the string is found in the first list the value is input as True otherwise the value is False.

Text

Description automatically generated

The function was used on each item in the list of lists containing a tuple of tokens and their label. It checked for the top 2000 word created previously on each. It extracted the tokens from the tuple and placed the functions output into a new list. The label portion was dropped, and the list of dictionaries was used to create a pandas data frame.

Graphical user interface, text, application

Description automatically generated

The modeling was done using Sklearn. To be able to compare easily multinomial naïve bayes was used for these models. The model was initialized, and an accuracy score was obtained using 10-fold cross validation. The precision, recall, and f1-score were obtained using the cross validation predict and a confusion matrix followed by a classification report from sklearn. This was much simpler but will provide less accurate results for the precision, recall, and f1 score.

Text, letter

Description automatically generated

Text

Description automatically generated

The f1-scores were captured by selecting portions of the confusion matrix and converted back to numbers using float. These were placed into a dictionary to be graphed for visual understanding of the f1-scores.

Graphical user interface, text, application, chat or text message

Description automatically generated

NLTK Bigrams

Bigram assoc measures was used to collect all the bigrams from the all words list created previously. Using the chi squared measure the top 500 were placed into a list.

Text, application

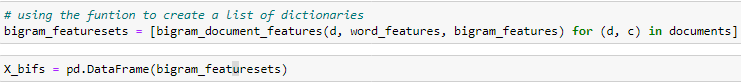
Description automatically generated

A function was used from lab that takes 3 lists of strings as input. The function checks if the strings in the second and third list are present in the first list. The strings from the second list have a “V\_” attached to the front and is placed as a key in a dictionary if the string is found in the first list the value is input as True otherwise the value is False. The strings from the third list have a “B\_” attached and follow the same process of being placed into a dictionary.

Text

Description automatically generated

The function was used on the lists of lists of tuples by extracting just the tokens and checked if either the 2000 most common words were present or if the top 500 bigrams were present and placed the results in a dictionary. This dictionary was then placed into a pandas data frame.



The modeling was done using Sklearn. The model was initialized and an accuracy score was obtained using 10-fold cross validation. The precision, recall, and f1-score were obtained using the cross validation predict and a confusion matrix followed by a classification report from sklearn.

Text, letter

Description automatically generated

Text

Description automatically generated

The f1-scores were obtained again using the same method and stored.

Text, chat or text message

Description automatically generated

Sklearn Unigram MNB

For vectorization the min document frequency was selected to be 5 and the number of features was manually tuned to find the best at 2450. To see a fair comparison a count vectorizer using Boolean was used and this was followed by tf-idf (Term frequency inverse document frequency) vectorization. The data was fit and transformed using the vectorization.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

The same 10-fold cross validation method was used along with the 10-fold cross validation predict to obtain precision, recall, and f1 score.

Text

Description automatically generated

Text

Description automatically generated

The f1-scores were stored using the same method as previous models.

Text

Description automatically generated

Since the accuracy was found to be so close only the tfidf vectorizer will be used in further comparisons. The change in accuracy was found to be 0.02%

Sklearn Ngram (Unigrams + Bigrams) MNB

For vectorization the min document frequency was selected to be 5 and the number of features was manually tuned to find the best at 3000. To see if there was a difference using bigrams for vectorization again the count vectorizer using Boolean was used and this was followed by tf-idf. The data was fit and transformed.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

The same 10-fold cross validation method was used along with the 10-fold cross validation predict to obtain precision, recall, and f1 score.

Text, letter

Description automatically generated

The f1-scores were stored using the same method as previous models.

Text

Description automatically generated

The accuracy was found to be 0.18% difference so tfidf was used for the comparison.

Since it was found not to have a very large difference in the model output NLTK features vs the Sklearn vectorization and the ease of creating and implementing features on the Sklearn the remaining vectorization feature tests were conducted in sklearn (adding stemming, adding trigrams, using other algorithms). The vectorization was also tfidf. This will be shown in the results section and discussed in the conclusion.

Sklearn Ngram (Unigram + stemming

The Porter stemmer from nltk was imported and an analyzer was created using this stemmer. A function was created that allowed the stemmer to be used inside the tf-idf vectorization.

Text

Description automatically generated

For vectorization the min document frequency was selected to be 5 and the number of features was manually tuned to find the best at 2450. The analyzer was selected using stemmed-words function created. The data was fit and transformed using the vectorization.

Text, application

Description automatically generated

The same 10-fold cross validation method was used along with the 10-fold cross validation predict to obtain precision, recall, and f1 score.

Text

Description automatically generated

**Text

Description automatically generated**

The f1-scores were stored using the same method as previous models.

Text, chat or text message

Description automatically generated

Sklearn Ngram (Unigrams + Bigrams + Trigrams) MNB

For vectorization the min document frequency was selected to be 5 and the number of features was manually tuned to find the best at 2450. The N-gram range was increase to include trigrams. The data was fit and transformed using the vectorization.

Text

Description automatically generated

The same 10-fold cross validation method was used along with the 10-fold cross validation predict to obtain precision, recall, and f1 score.

**Text

Description automatically generated**

**Text

Description automatically generated**

The f1-scores were stored using the same method as previous models.

Text, chat or text message

Description automatically generated

Sklearn Ngram (Unigrams + Bigrams) Linear SVM

Since SVM works well with numeric data such as provided by tfidf and it is easy to select features with sklearn’s vectorizer SVM was implemented with to test against the MNB models used earlier. This will also be implemented using SVC with kernel tricks to operate in higher dimensional space.

The vectorization options selected were unigrams and bigrams, max document frequency was set to .5 and tfidf was used. Finally, the max features were set to 10,000 this feature was found that the higher it was the better the performance, but the training took much longer. It was also found to take to much computing power and thus was selected to be 10,000 to be considered usable. The change in accuracy beyond 10,000 was run once and found to increase accuracy by around 0.2% but take around 4 hours to train. The data was fit and transformed using these vectorization options.

Graphical user interface, text

Description automatically generated

The same 10-fold cross validation method was used along with the 10-fold cross validation predict to obtain precision, recall, and f1 score.

Text, letter

Description automatically generated

Text

Description automatically generated

The f1-scores were stored using the same method as previous models.

Text, chat or text message

Description automatically generated

Gridsearch

Gridsearchcv was used to tune the model checking the C parameter on multiple levels (0,1, 1, 10, 100) The same features for vectorization were chosen as in the linear svm model.

Text

Description automatically generated

Sklearn Ngram (Unigrams + Bigrams) SVC

One vector was created for multiple SVC models using different kernels (kernel trick allows the model to operate using planes in higher dimensional space). The model used bigrams and unigrams along with tfidf and the min document frequency was selected to be 5. The max features were set to 3000 as these models took a long time (12hr+) to run. The data was fit and transformed.

Graphical user interface, text, application, email

Description automatically generated

For each model the same 10-fold cross validation accuracy was obtained, and 10-fold cross validation predict was used to generate precision. Recall, and f1-scores. The f1 scores were stored using the same method. The code for each is shown below.

RBF

Text, letter

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Poly

Text, letter

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Sigmoid

Text, letter

Description automatically generated

A screenshot of a computer

Description automatically generated

Text

Description automatically generated

Feature Sets, Set up discussions

To test the feature sets being created against the original accuracy of a model some of the base models needed to be selected. Since there will be many models run using the feature sets (one for each being tested for each feature set being generated) MNB was selected because of time needed to run the models and computing power required for the models. Both the NLTK and SKlearn that were comprised of unigrams and bigrams were selected to implement the feature set on to see how new feature sets change the output. Below is the explanation of the feature sets creation followed by a short example of the testing processes followed by the creation of the final two models with multiple feature sets one for NLTK and one for Sklearn.

Vader Sentiment feature set creation

NLTK’s vader sentiment analysis was used to loop through and get a compound score for each review. The compound review ranged from -1 to 1 and was normalized in a range from 0-1.

Graphical user interface, text, application, email

Description automatically generated

Subjectivity feature set creation

The subjectivity feature function was obtained from lab and used create a dictionary of the subjectivity of select words given from lab. The function was used, and the corresponding words were added into a list of lists. New code was created that took the weak positive scores and 2x the strong positive scores and add them into a list, this was the positive count. The code also did the same with weak negative and strong negative scores and added them into a list called the negative count. These scores were then normalized to range from 0 to 1.

Text

Description automatically generated

Text

Description automatically generated

Sentence Count feature set creation

The NLTK sentence tokenizer was used on the preprocessed raw text and appending into a list. A count was created for each token in each review and normalized.

Graphical user interface, text, application, email

Description automatically generated

Word Count feature set creation

The NLTK word tokenizer was used on the preprocessed raw text and appending into a list. A count was created for each token in each review and normalized.

Graphical user interface, text, application

Description automatically generated

POS Tag Count creation

The POS function from lab was taken. This function took all the words in the document and adds a part of speech tag to each word. It then checks for the specific words contained in the top 2000 words previously created and if the word appears it takes a count of the word’s part of speech and adds this to a new dictionary.

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

The feature set created was transformed into a pandas data frame and only the POS count columns were converted to list form and normalized.

Graphical user interface, text, application, email

Description automatically generated

Negation Count creation

A negation function was obtained from IST736 which uses regular expressions to search though each review and find a match for words “not”, “no”, and “never” or any word ending with “less”. If a match is found a 1 is returned if no matches are found a 0 is returned. This function was used on each review and its output was saved to a list.

Graphical user interface, text, application, email

Description automatically generated

Testing feature sets

The NLTK output which was already in a pandas data frame was ready to be used. The sklearn vectorized data was placed into a pandas data frame using .toarray() and the get feature names out to label the columns.

Text

Description automatically generated

A dictionary was created to store the accuracies of each 10-fold cross validation for comparison of the feature set and decide which features would be added for the combination set/ final model. The original accuracy of the models (Sklearn unigram + bigram mnb and NLTK 2000 top words and 500 top bigram mnb) were added to the dictionary

Graphical user interface, text, application

Description automatically generated

A feature set was added to both pandas data frames and each data frame was used to find a 10-fold cross validation accuracy using Multinomial naïve bayes algorithm. The accuracies were then added to the dictionary. The columns created using the feature sets, were then dropped and the process repeated for each feature set. (only one is shown refer to the attached html Morgan\_Gere\_IST664-Final\_Feature\_sets to see full code)

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

The dictionaries were then graphed. One graph code is provided below.

Text

Description automatically generated

Final Model creation (multiple feature sets)

The 10-fold cross validation predict was used to generate confusion matrix and classification reports for the best preforming model with feature sets added. (one for NLTK 2000 top words and 500 top bigrams and one for Sklearn unigram and bigram vector both multinomial naïve bayes) The accuracy and F1-scores were saved and placed into a dictionary and graphed for visual inspection.

Text

Description automatically generated with low confidence

Text

Description automatically generated

Text, chat or text message

Description automatically generated

Text, letter

Description automatically generated

Text

Description automatically generated

**Results**

Accuracy raw agreement, baseline and Vader

The accuracies found to tell how the models are doing are 0.2, 0.25 and 0.3. 30% is the accuracy needed to beat random guessing out of the box sentiment analysis.

NLTK BOW vs Sklearn Unigram

10-fold cross validation accuracy

NLTK Sklearn

** **

10-fold cross validation predict confusion matrix and classification reports

NLTK

Table

Description automatically generated

Sklearn

Table

Description automatically generated

NLTK Bigrams vs Sklearn Unigram + Bigrams

10-fold cross validation accuracy

NLTK Sklearn

** **

10-fold cross validation predict confusion matrix and classification reports

NLTK

Table

Description automatically generated

Sklearn

Table

Description automatically generated

Sklearn stemming

10-fold cross validation accuracy: Sklearn 

10-fold cross validation predict confusion matrix and classification reports

Sklearn

Table

Description automatically generated

Sklearn Trigrams

10-fold cross validation accuracy: Sklearn 

10-fold cross validation predict confusion matrix and classification reports

Sklearn

Table

Description automatically generated

Sklearn Linear SVM and Gridsearch

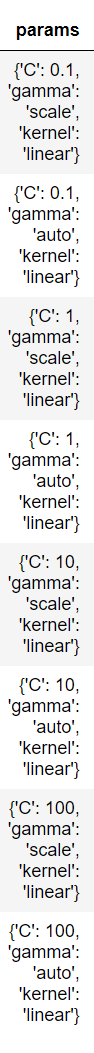
10-fold cross validation accuracy: Sklearn 

10-fold cross validation predict confusion matrix and classification reports

Sklearn

Table

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Sklearn Linear SVC

RBF

10-fold cross validation accuracy: Sklearn 

10-fold cross validation predict confusion matrix and classification reports

Sklearn

Table

Description automatically generated

Poly

10-fold cross validation accuracy: Sklearn 

10-fold cross validation predict confusion matrix and classification reports

Sklearn

Table

Description automatically generated

Sigmoid

10-fold cross validation accuracy: Sklearn 

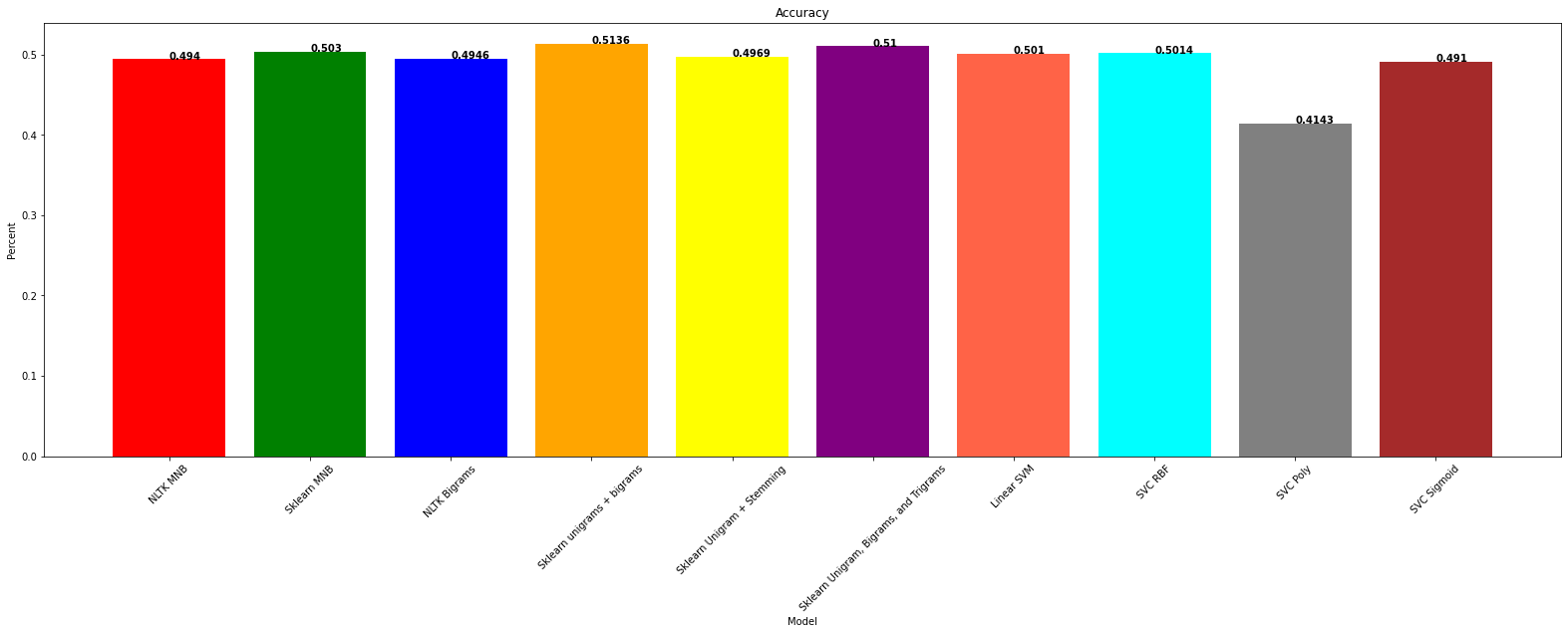
10-fold cross validation predict confusion matrix and classification reports

Sklearn

Table

Description automatically generated

All Models no feature sets Accuracy



All Models no feature sets F1-scores

Chart, bar chart

Description automatically generated

Feature sets Accuracy NLTK

Chart, bar chart

Description automatically generated

Feature sets Accuracy Sklearn

Chart, bar chart

Description automatically generated

Original Model vs Final Model NLTK

Original 10-fold cross validation confusion matrix and classification report:

Table

Description automatically generated

Final 10-fold cross validation confusion matrix and classification report:

Table

Description automatically generated

Chart, bar chart

Description automatically generated

Original Model vs Final Model Sklearn

Original 10-fold cross validation confusion matrix and classification report:

Table

Description automatically generated

Final 10-fold cross validation confusion matrix and classification report:

Table

Description automatically generated

Chart, bar chart

Description automatically generated

**Conclusion**

NLTK vs Sklearn Features

Sklearn’s process of vector creation is very simple and easily repeatable. If you want to select different vectorization options, it only involves changing portions of one line of code and an entire model can be created differently. There are many options such as including stop word removal, adding stemming, adding bigrams or trigrams, only having bigrams, selecting the number of features, looking at how often the features appear in the documents, and more. For NLTK the code is much more complex. While there are other ways rather than what was done in the lab code to accomplish similar results, it still will take a lot of code, and to alter the code is much more time consuming and troublesome.

Sklearn also runs faster for model creation with more metrics to allow comparison of the model’s performance. The output of the vectorization is easily used to accomplish other things. With NLTK it was difficult to run the models with the same level of complexity. They took much longer to run, and thus it was decided to change the output into a pandas data frame and allow the models to be created in Sklearn. While this still allows for a look at NLTK’s feature extraction vs Sklearn vectorization it was felt it would have been a better experiment overall to run the models in NLTK but the computing time was so high to be able to accomplish many different experiments and run many models for each experiment it almost seemed like an impossible task.

NLTK vs Sklearn model output

The first models looked at Sklearn vectorization vs NLTK feature extraction using MNB (Multinomial naïve bayes). The accuracy of only using unigram tokens of the sklean’s vectorization was slightly higher than that of the NLTK’s feature extraction, around 0.91%. While looking at the different metrics precision and recall, the unigram sklearn vectorization accomplishes a model with higher precision and NLTK feature extraction has higher recall. The F1 scores are higher in general for NLTK. This all tells that Sklean’s vectorization has more of its predictions correct but gets less of the total number of predictions that are possible. While NLTK gets more prediction wrong but captures more the overall possible predictions. There isn’t a clear reason with this data to prefer one metric over the other without a direct purpose of the classification being clear. If we look at the F1 scores the Sklearn and NLTK, overall preform similar for the 1 and 5 rating, while NLTK does a much better job at classifying 3, and sklearn does better at 2 and 4. If there was a direct reason to prefer precision (capturing as many correct predictions) Sklearn is a better choice if you wanted to select recall (making sure the model misses the least classifications) NLTK is the better choice. Looking at the F1-scores they are look even. Overall, for the unigram model simply because the accuracy is higher the Sklean model pushes ahead.

Looking at the models that added bigrams the accuracy difference increased to 1.9% with sklearn over NLTK. The other metrics remain at similar levels, meaning the same conclusion could be drawn.

Additional Sklearn model output

To take a deeper look at other possible vectorization options and because of the ease of implementation of the vectorization options stemming and adding trigrams were tested using Sklean vectorization only. This was followed by implementing linear svm and non-linear svm (support vector machines) on the bigram model (the best preforming vectorization option).

With Stemming there is a reduction in the accuracy and the f1-score of classification 3 is reduced which makes stemming not a good option for this data. While looking at trigrams the accuracy is slightly worse but the f1-score of classification 3 is slightly better this could have been selected as the best model.

Of all the different SVM models produced, linear and RBF preformed the highest. All the SVM models took almost 20x the amount of time to produce any of the non-linear SVM models taking much more time than that. Because the results were similar between them a gridsearch was preformed on linear svm to find the best C value and the results showed an accuracy of 52.48% which is the highest accuracy. Some attempts were made to use the feature sets on this model but because it used all possible features somewhere in the 400,000 range when new features were added it was too much computing power. The max features were moved to 80,000 features and gave no changes when new features were being added. This was not included in the code because how many times the code was rerun to try and increase the accuracies, this model just took to long to remain a viable option and was scrapped.

To remain with the comparison of NLTK and Sklearn the model’s using unigram and bigram along with MNB were selected to have additional feature sets used in addition to their original features. This was done for two reasons, 1 to see how feature set addition would change both sklearn vs NLTKs features and 2 because these models were both the best performing models for each type of feature extraction.

Feature sets creation

The creation of additional feature sets was the same for both NLTK and Sklearn. Since the models were being made in Sklearn because of simplicity and computation time adding the new features to a pandas data frame was simple for both. If however the process was to be done all with NLTK adding new feature sets would require getting the output into a specific type and placing it in a specific format which would have been much more difficult overall. This pushes the use of Sklearn vectorization far ahead of NLTK.

There were simple engineered feature sets tested, followed by more complex ones. The simple ones were word count, and sentence count. This could have shown if people writing the reviews had more to say about books they loved or hated. Or if they were using more in-depth sentence to discuss preferred books vs non preferred books. This was followed by adding an out of the box sentiment analyzer (Vader) to give a compound sentiment score for each review. This would allow to see people sentiment behind the words they used. This was also preformed using the subjectivity from lab to have a different type of score. A negation count which states if the review was using negation or not. Finally, a Part of speech tag was used to get a count of nouns, verbs, adjectives, and adverbs.

Feature sets accuracy

To see which feature sets increase the productivity of the model only accuracy was used when comparing the different feature sets. Then the multiple feature sets that increased the accuracy were used together the model with the highest overall accuracy was used to create a “final” model and capture the accuracy, precision, recall, and f1-score to compare against the original values captured from the original model without additional feature sets.

For NLTK the accuracy was increased when the sentiment score from Vader was applied, the subjectivity was included and both the word count and sentence count. These were tested and ran in many combinations finding the best combination to be Vader plus subjectivity. Yet this combination was still lower accuracy than simply using the Vader sentiment by itself but not by much. This combination model was chosen because the accuracy was so close.

For Sklearn there is a accuracy increase in the Vader sentiment. A combination was attempted with multiple ones and found for the greatest accuracy of possible combinations to include Vader, subjectivity, and negation. The feature sets chosen for the final model was that of just the Vader sentiment. This was because there seemed to have significant difference in accuracy.

Final Model

The addition of engineered feature sets had a greater change in the Sklearns model over the NLTK’s model (change of 0.5% vs 0.9%). This change is very small with both. While it is possible to change the model by adding feature sets. I believe it would work better with other types of data other than book reviews.

Final Thoughts

Sklearn is much easier and faster form of vectorization to achieve very similar results. It allows for many more options to be considered in a much shorter amount of time. The feature extraction is very good and allows for extra features such as tf-idf to be implemented with very little work. This is a tool I would choose over NLTK’s simply for these reasons.

As for feature engineering it can help with sentiment analysis and for this data set it did very little but with more time a better feature set could have been engineered. If more time allow the next feature set added would have been a addition of sentiment/ subjectivity with negation included. Each review could be broken down into sentence tokens and a sentiment of subjectivity of that sentence acquired then if the sentence had a negation present it could have appended the sentiment/ subjectivity score from that sentence after being multiplied by -1. Then all the scores could have been summed and the score for each review could have been normalized. There are countless approaches and feature sets that can be added. This is an important tool for NLP classification.

The overall thought of this is creating a good classification using text takes time and many different trial and error experiments. This is not something that can be done simply, and preprocessing of the data can be very important to the end result.