Nicholas McFadden

9/10/2022

Natural Language Processing

Classification of Kaggle Movie Review Sentiment Analysis

The dataset is a corpus of Rotten Tomato movie reviews from a Kaggle competition. A group of crowd sourced individuals annotated all the subphrases of sentences with the sentiment labels: “negative”, “somewhat negative”, “neutral”, “somewhat positive”, “positive”. In this project I will train a Naïve Bayes classifier on various feature sets using cross-validation to obtain precision, recall and F-measures scores to determine the best set for the task. There are 156,060 phrases in corpus of which I will begin with a subset of 1000. I chose 1000 as this sample size because without knowing anything about the data I can assume a standard deviation of .5 is often a fair value to ensure the sample is representing the population. Therefore, this gives me approximately a 95% confidence interval and a 3% margin of error or 99% confidence and a 4% margin of error depending on how you want to look at it. I believe this to be sufficient for the task at hand as it is a movie review classification and something needing higher accuracy such as heart attack detection.

To ensure the model doesn’t overfit I will be applying cross-validation of 10 folds to the Naïve Bayes classifier. I will use precision, recall, and F1 scores to evaluate the differences in feature sets on the performance of the model. However, I will include an overall accuracy value, not cross-validated, just as a high-level view to base the inclusion of a confusion matrix in the beginning. The confusion matrix will be viewing how the data is being skewed and how new bins can assist. The reason being that I do not intend to do a hard balancing of the data set. As seen later in the analysis, I do complete some pre-procession to mitigate risks that could arise from this approach, and I will focus more on the micro averaging of the scores which is a weighted average of the labels.

To begin I will need to import some packages:

Text

Description automatically generated

Then I needed to get the path of the training corpus with the following code:

Text

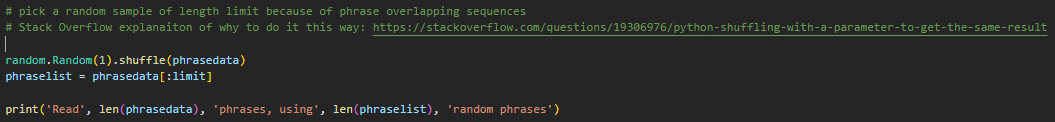
Description automatically generated

Then I needed to create a function(processkaggle) that is going to be the main function from which the feature functions will be executed through. This function pulls the train.tsv file provided, removes the header, strips any end characters, and splits the 4 tabs to use only the phrase and its sentiment:

Text

Description automatically generated

I picked a random sample of 1000 phrases but to make this analysis repeatable I needed to set a seed for the randomization:



Of the phrases collected a phrase document needs to be create as a list of words and a label. To accomplish this, the phrases need to be tokenized by NLTK’s word tokenizer and combined with its sentiment score. The sentiment labels are between 0-4 representing:

* 0: “negative
* 1: ”somewhat negative”
* 2: “neutral”
* 3: “somewhat positive”
* 4: “positive”

As mentioned before, I am not going to do a strict balance, but I am going to combine some of these labels. I am mainly interested whether the movie review was negative, neutral, or positive and not on the magnitude of negativity or positivity. Therefore, binning these reviews into 3 different categories: negative, neutral, positive seemed more appropriate. Reviews that had sentiment scores of:

* 0 or 1: binned as “negative”
* 2: binned as “neutral”
* 3 or 4: binned as “positive”

Jumping ahead slightly, I computed the baseline (Unigram/Bag-of-Words(BOW)) with and without the binning and that the model performed better with the binning.

|  |  |
| --- | --- |
| Original Scoring Matrix | New Bins |
| Text  Description automatically generated |  |
|  |  |

I see that the new binning method improves the model across all metrics. Obviously by having more data points for a specific category to train on and it being less complex it improves the model. However, reviewing the Confusion Matrix I can see that the correctly identified values are still skewed to the neutral values. Although improved from the 5-bin model, this will still require the usage of the “Micro Average”, a weighted average, when making decisions about performance. Thus, providing more “balance” in the decision-making process without manually balancing the data.

Now continuing with preprocessing to create the baseline featureset, a Unigram/Bag-of-Words (BOW), that ran the above bin check. I made all the phrases lowercase to circumvent any case differences and then parsed the token list of the phrase docs so I could extract the 1500 most common words out of the 2620 identified to create the word\_features.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Now I needed to define a feature definition function to define the keywords of a document for a BOW/unigram baseline, so each feature(keyword) is ‘V\_(keyword)’ and is true or false depending on whether that keyword is in the document.

A screenshot of a computer

Description automatically generated with medium confidence

Then this gets utilized back in the ‘processkaggle’ function to create the first featureset.



Now as mentioned previously I need to train my classifier and show performance in cross-validation. Therefore, I create a function to complete the cross-validation and feed it with the number of folds, which is 10 so 100 values per fold, the featureset, and the unique labels.

Text

Description automatically generated

The cross-validation function will take the defined number of folds and divide that by the featureset length and take that many samples of that size. Each fold is separated into a train and test set and the Naïve Bayes classifier is run to make the predictions.

Text

Description automatically generated

These predictions are run through an evaluation function to calculate the precision, recall, and F1 measures by tallying the True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN).

Text

Description automatically generated

Finally, the cross-validation function averages them together at the end. This helps ensure that the model is not overfitting to this one featureset by trying to introduce some variability. The precision metric is the measure of how many of the positive predictions made are correct (true positives). Recall is a measure of how many of the positive cases the classifier correctly predicted, over all the positive cases in the data. The F1 score is a measure of combining precision and recall but the idea is to provide a single metric that weights the two ratios (precision and recall) in a balanced way, requiring both to have a higher value for the F1-score value to rise. In a normal average (mean) if your recall was low (.01) and the precision was high (1.0) you would get .505 which is misleading when the F1 would be closer to .02. Also, F1 scores tend to be better for imbalance data sets which I do have. At the same time between Precision and Recall, Recall is not as important in the context of movie reviews. If your decision to watch a movie if you miss a positive result (aka predicting false negatives) it is not as impactful as say missing someone who had cancer. I may just end up watching or not watching something because of what the model predicted for the sentiment of the review.

Text

Description automatically generated

This is what was completed for the original featureset(unigrams/BOW). However, no filtering of stopwords occurred and when reviewing some of the top words there was additional punctuation being accounted for and the words ‘movie’ and ‘film’ that was not providing value. That is why I add this to the NLTK stopwords corpus to remove from the featuresets. One of the featuresets is negation so I did not want the stopwords to include this negation and overlap so they were listed for later use and removed from the stopwords list.

Text

Description automatically generated with medium confidence

When I compare the original featureset to the original featureset but without the stop words I was surprised to see that the stop words were actually worse than the original featureset across all metrics. The only party being that the Average Precision for neutral was higher for the stopwords. Seeing as the neutral category included not only had more records but it had more of the kinds of stopwords being removed so removing them reduced some of the noise inherent in the “neutral” category but not sufficient to improve the results. It helped remove noise for neutrals but reduced it for the positive and negative sections.

|  |  |
| --- | --- |
| Original | Original w/o stopwords |
|  | Text  Description automatically generated |

This defines a negation function that will go through every word in the word features and negate the word that follows a negation word or “n’t”. Calls the Not\_features function that was defined above. The NOT\_featuresets is generating an array. Each item in the array has both an object of features and the sentiment. The object of features contains every word in word\_features so V\_word : TRUE, V\_NOTword : FALSE. It states whether that word follows a negation word and if the word is in the phrase.

Text

Description automatically generated

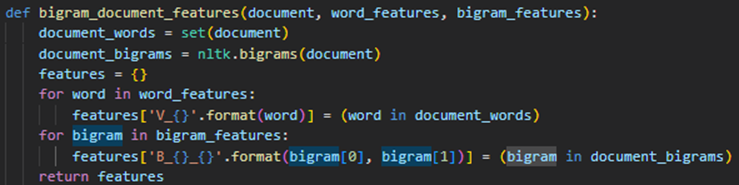
On average Negation performed worse than the original but it did improve in the precision of the positive sentiments improving its F1 score accordingly. The negation may not have help identify more negative sentiments, but it at least knew that the negation was not positive.

|  |  |
| --- | --- |
| Original | Negation |
|  |  |

Next is bigrams, this imports the nltk.collocations.BigramAssocMeasures from the nltk.collocations and saves it in bigram\_measures.



This defines a bigram function that contains both word features and bigram features. There are two loops in this function. The first loop goes through every word in the phrases and creates a sparse matrix with V\_word and states True or False depending on if the word is in that specific phrase. The second loop creates a sparse matrix of bigrams V\_word\_word and states true or false depending on if that bigram is in the phrase.



This line goes through all of the words in the all\_words\_list and creates bigrams and stores them in an array called finder.



It is important to note that when creating bigrams, you cannot remove stopwords as then the bigrams would not be accurate. This uses the chi\_sq measure to define the top 500 bigrams.



Calls the bigram\_document\_features function that was defined above. The bigram\_document\_features is generating an array. Each item in the array has both an object of features and the sentiment. The object of features contains every word in the phrases followed by true or false, depending on if the word is represented in that specific phrase, it also contains the top 500 bigrams with a true or false, depending on if the bigram appears in that phrase followed by the sentiment.



Overall, they had the same scores, but the bigrams did improve on neutral recall enough to improve the F1 score. However, it took a hit in the recall of positive sentiments. As mentioned previously, lower recall is not completely detrimental to this analysis. Therefore, I would say that I would go with the Bigram feature set in this case. I personally consider the F1 stat to be more beneficial in this case because an F1 is better for skewed data. Also, if it can still account for enough to improve the F1 of the neutrals but reduction in Recall didn’t move the F1 for the positives, I think that is significant.

|  |  |
| --- | --- |
| Original | Bigrams |
|  |  |

The POS feature applies NLTK’s pos\_tag to the documents and the counts each instance o a noun, verb, adjective, and adverb. Then based on these counts makes a prediction.

Text

Description automatically generated

This was very interesting to see that according to the POS featureset had a higher overall F1 when accounting for weights. The Bigrams had a higher precision and recall but the average of their F1 was lower. I attributed this to more volatility in the folds for the bigrams than in the POS. Subsequently, POS had a higher neutral recall and F1; higher negative precision, recall, and F1.

|  |  |
| --- | --- |
| Bigrams | POS |
|  |  |

In this instance I would stay with Bigrams as the final model. The reason being the Bigrams preformed mainly better in the precision and F1 for positive sentiments. I would rather that the Bigrams featureset identifies positive sentiments better than either the negative or neutral ones. In practicality I think people, myself included, tend to trust positive sentiments more and would choose to watch a movie associated with a positive review. The more precise the model is in identifying them the less likelihood I am to watch a potentially bad movie. Again, this is not the end of the world if it was a false positive and have to watch the occasional bad movie. Therefore, the model’s ability to correctly identify the neutral and negative measures do need to be as robust. The other comparisons were either completely obvious or so minute I don’t believe we would have been particularly worse off. However, this final comparison was more of a mixed bag. Ultimately, I believe the goal of focusing on the “positive” sentiment identification was more important to a practical use of the analysis.