CS506 Midterm Write Up

Timothy McCorry

Summary

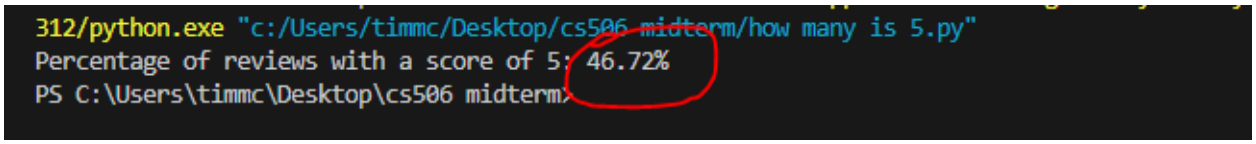
My review of this midterm was that I feel like I learned a lot but that I was disappointed in my final score. In this report I work through my workflow and the strategies used to achieve my score – while highlighting strategies that did not work and areas for improvement.

Final Score

My best Kaggle submission had a score of 0.54622 which was only 8 percentage points above an algorithm that would have guessed 5 with every prediction.

Best Kaggle Submission

Script run on train.csv to show what percentage of the data was 5 stars

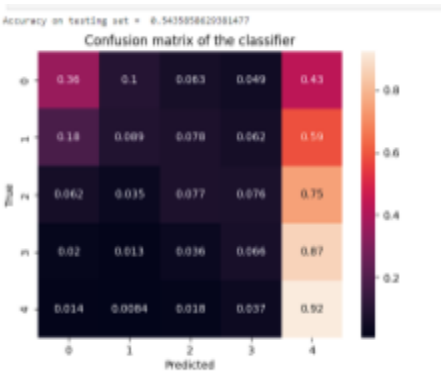


Algorithm Overview

The best performing submission used the VADER (Valence Aware Dictionary and sEntiment Reasoner) sentiment analyzer which is a lexicon based tool that judges text. The two scores that I found to be the most effective were the negativity and positivity score of the review summaries. I also tried using the compound scores and testing on the 'Text' component but surprisingly that did not help. I would assume that this is probably revealing of a problem elsewhere because I would assume that the 'Text' component has more valuable data.

```python
def get_sentiment_scores(text):
    if isinstance(text, str):  # Check if the text is a string
        scores = analyzer.polarity_scores(text)
        return scores['pos'], scores['neg']
    else:
        return 0, 0  # Return default scores for non-string inputs
```

The confusion matrix of the best run that I had is displayed on the right. You can see, as expected, that a 5 star rating is predicted the most. The darker areas is the part where the model is performing the least which is basically everywhere but 5 and it performs a comparatively better at 1 star predicted.



Methodology

My research ahead of beginning the project brought up two main approaches to solving this issues which were:

1) Tokenizing the 'Text' component of the review or making it a vector like TF-IDF (which was used in the highest scores seen at the lecture discussion) or CountVectorizer from SK Learn which I used
2) Using sentiment analysis either through a library like VADER (which had the best results for me) or otherwise

Both of these strategies it seemed like a great option for classification was Random Forest algorithm which uses several decision trees (cause for the forest name) combining them into one single output

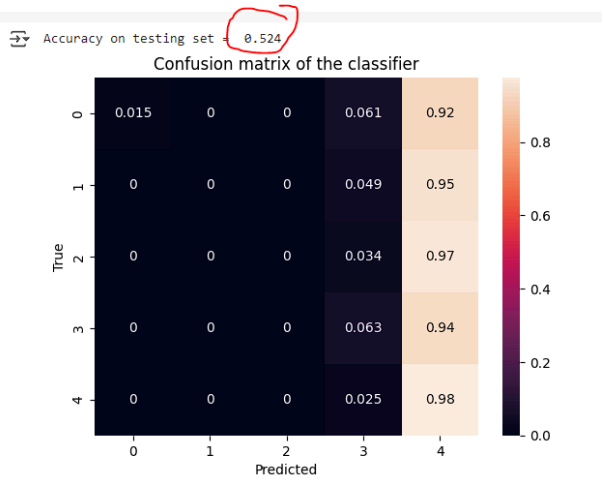
[12]:
```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, random_state=40)

model.fit(X_train_select,Y_train)

Y_test_predictions2 = model.predict(X_test_select)
```

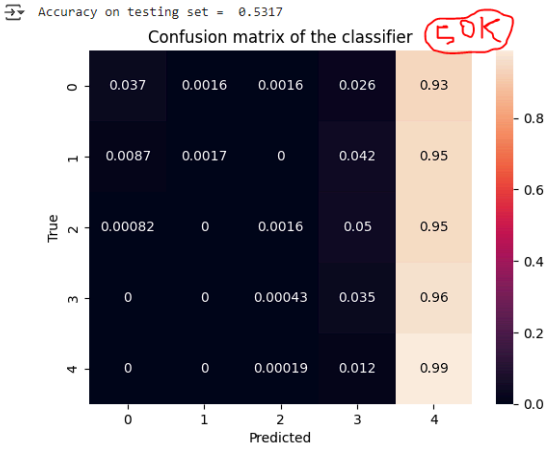
I first implemented VADER with KNN classification and as soon as I switched to Random Forest I saw a several point percentage jump – it also ran the notebook significantly faster.

In Practice

My plan which proved unsuccessful was to optimize the sentiment scores the best I could and then use that in combination with text tokenization to run Random Forest classification.

To do the analysis of the ‘Text’ section I first partitioned the data into 5000 row CSV and a 50,000 row CSV and used SK Learn CountVectorizer to build functional models and then increase them as everything is running smoothly. The confusion matrix below shows that this method was very 5 star dependent – but that increasing training gained a point.





I also tried using a parameter of Random Forest called class_weight which uses the unproportional distribution of the star ratings (Ex: there were way more 5 star ratings), but this weight caused accuracy to decrease.

```
# rf = RandomForestClassifier(random_state=2)
# rf.fit(X_train_vec, y_train)

rf_weighted = RandomForestClassifier(random_state=2, class_weight=updated_class_weight_dict)
rf_weighted.fit(X_train_vec, y_train)
```

```
RandomForestClassifier(class_weight={1: 3.134796238244514, 2: 3.389830508474576,
                                     3: 1.5267175572519085,
                                     4: 0.9259259259259259,
                                     5: 0.37721614485099964},
                       random_state=2)
```

Conclusion

At the competition's end my best performing algorithm was a pretty novel implementation of a Random Forest algorithm using sentiment scores from VADER on the movie review summaries with manual tweaks done. The plan of combining the sentiment score of summary with tokenization of ‘Text’ component was on the right track to achieve a better performing algorithm and could have resulted in a better performance but it did not fully come together. I had a lot of trouble with the kernel crashing and this was the first time where I have had notebooks that you have to let run for hours – toward the end of the competition I saw improvement as I ran the text analysis on more data.