The Sisyphean Task of Predictive Modeling: An Amazon Ratings Classifier

Author: Dimitriy Shchedrovitskiy
Kaggle Username: Dimitriy Shch
Date: 11/7/2023

Abstract:
Tackling this midterm project turned out to be a real mix of highs and lows. It was challenging, sure, but I came out the other side knowing a lot more. I poured a lot of effort into trying different ways to get the results I wanted, although I didn't always see the success I was hoping for. It's been a tough but informative ride. In this write-up, I'm going to walk you through the nitty-gritty of how I built a machine learning model to guess how many stars people give products on Amazon. You'll get the scoop on how I picked out the most important bits of data, choose the right tool for the job, fine-tuned all the settings, and made sure everything was working right. Along the way, I hit some snags and had to figure out how to deal with them. Plus, I'll share some cool stuff I figured out about the way people write their reviews on Amazon and how that helped me build my model.

Data Exploration and Preprocessing:

When it came to data exploring and preprocessing, the code given to us in the sample did a good job of highlighting some biases and issues with the code, those being the skew of data towards 5 star reviews, the fact that the significant chunk of data given wasn't very reliable since a large population of reviewers and products had a single review. This would be likely chaotic and hard to categorize, my first thought when it came to preprocessing is what if I drop these review and just leave the less skewed ones in/ This latter proved to be the wrong approach even tho achieving high accuracy locally it was non-submittable due to size mismatch, I would also reason it wouldn't be great at the set of all reviews since it would be unfamiliar with a large chunk of the data. See Appendix A.

It was clear I needed a change of strategy and I decided to work with simple, apparent and clear metrics. I decided to look at things like review length, length of words in review, character count and such. This proved to be an efficient method of approaching this and graphing some of these correlations resulted in some interesting correlations see Appendix. B - E. There were clear patterns for each rating in these metrics, so I settled on using these factors as part of my model.

Feature Engineering:

This was rather straightforward. After doing enough data exploration and processing I just decided to add the code that I used to generate these graphs to my feature processing and add a new column for them. Sadly I didn't see very good results, and needed to think about some other ways to improve the performance of this model. I decided that a sentiment analyzer could be very useful since identifying positive from negative reviews would be a task that needs to be approached with more complex features that I was using till that point. I decided to look at word clouds, see Appendix F and G. The result wasn't perfect since even after some text cleaning and removal of the words “Film” and “Movie” I was still left with some overlap, however the important part is the clouds, in general, were different which meant that sentiment analysis could work. After implementation of it utilizing TextBlob I generated some graphs and the result speak for themselves, See Appendix H and I, as you can see there is a clear correlation between score and Sentiment polarity, subjectivity wasn’t as clear but it felt useful to keep to give context to the polarity within the model, so I decided to move onto model engineering with my main feature being sentiment, with a bunch of supporting ones which I described previously.

Model Engineering:

My initial foray into model selection began quite traditionally with the K-Nearest Neighbors (KNN) algorithm given to us in the initial sample. It wasn't long before its limitations prompted a switch to XGBoost, a decision fueled by familiarity and past successes. Unfortunately after my failure with my first approach to drop less reliable reviews the switch to XGBoost turned into a mess of feature engineering and hyperparameter tuning that didn't yield the improvements I had hoped for. It also took obnoxious amounts of time. I decided to add GridSearchCV to try to find the ebay model parameters, however this turned into a huge mistake in both my time and accuracy (see appendix J). The results were underwhelming, and the accuracy stubbornly unsatisfactory. I was thinking of giving up at this point since the amount of work I have sunk into this so far to get no result started feeling like a Sisyphean task.

As I got the energy to return to this assignment I decided to delve deeper, experimenting with Random Trees and the intricacies of pipelines. I appreciated the clarity pipelines brought to the process, especially when integrating the TFIdf Vectorizer. This improved the model's comprehension of text data significantly, and resulted in a small increase in my scores. Yet, XGBoost continued to disappoint, especially when the confusion matrix exposed its weakness in classifying non-extreme ratings. Implementing SMOTE provided some improved results, especially for those middle-ground predictions, but success on Kaggle eluded me. I also had to drop TFidf since smote cannot work with text fields. In the End I decided to get rid of SMOTE.

LightGBM was my turning point. The efficiency it brought to handling large datasets was transformative, cutting down training times from hours to minutes and offering encouraging confusion matrix insights. I felt a rush of excitement as my local RMSE scores dipped below 1.0—a personal victory. But Kaggle's feedback was a reality check, the scores did not mirror my local success. However I did not give up and continued working.

Previously I saw what an improvement SMOTE could yield and decided to look for a solution which would do something similar but not ruin the integrity of my whole model utilizing text data. I found a parameter for LGBM called class weight, this greatly increased the ability of my model to identify reviews that were not on the extremes, while sacrificing some accuracy of 5 star and 1 star reviews my confusion matrix finally became more logical (Appendix K) and signaled an improvement in model performance, a dip below 0.9 locally. For some reason the kaggle submissions kept not matching up to my results tho and were honestly a huge discouragement, since every breakthrough I felt like I made wasn’t validated.

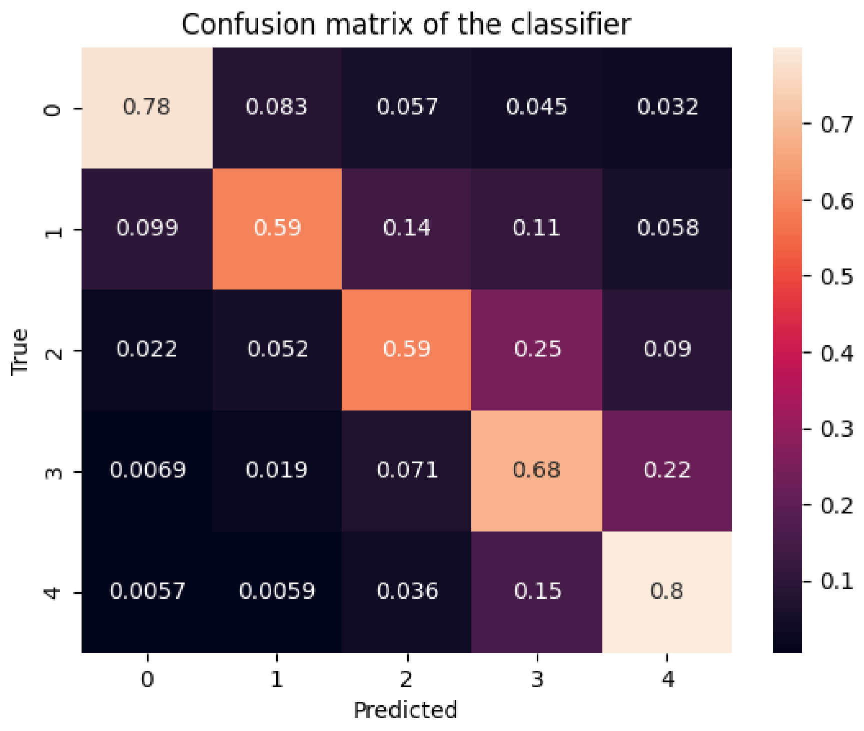
I spent a long time trying to walk through the logic of my code and attempting to gain a deeper understanding of what could have been causing this seemingly impossible gap between my results and the kaggle submissions see appendix L. The issues I uncovered made me quite embarrassed, I completely missed the fact that all of my confusion matrices show results as an array starting from 0 meanwhile kaggle expects a start from 1. This made me feel both exhilarated and completely clueless since I spent a completely unbelievable amount of time trying to fix errors in my code and reimplementing models over and over spending hours on this to in the end be enlightened by a difference of +1. When I finally discovered this I got a score of 0.92 on my best local model, a respectable and much needed victory.

I got a new burst of energy and decided to make my final improvement to the model by stacking models, a surprisingly easy process in the pipeline which greatly increased my runtime at first since I was using things like random trees which did not yield great times. However later I switched it to LinearSVR with its promises of great performance and previous discussion of Support Vector machines in class I decided to give it a shot. However my stacked model also needed a final_estimator to produce meaningful results by taking the results of the input estimators, the choice of LogisticRegression was more due to a lack of time and my familiarity with its quick performance rather than the most robust. However even with some shortcuts taken in the final model with some parameter adjustments, mostly having to do with the estimators and learning rate for the LGBM classifier, which at first I thought was improving from a smaller estimator pool and a larger learning_rate was an incorrect assumption since when I took the time to run it with all the new improvements and a larger number of estimators and a slower learning rate I was able to achieve an absolutely amazing result of 0.82 locally and 0.83 on the kaggle (Appendix M). At this point I was quite satisfied with my model and was ready to submit.

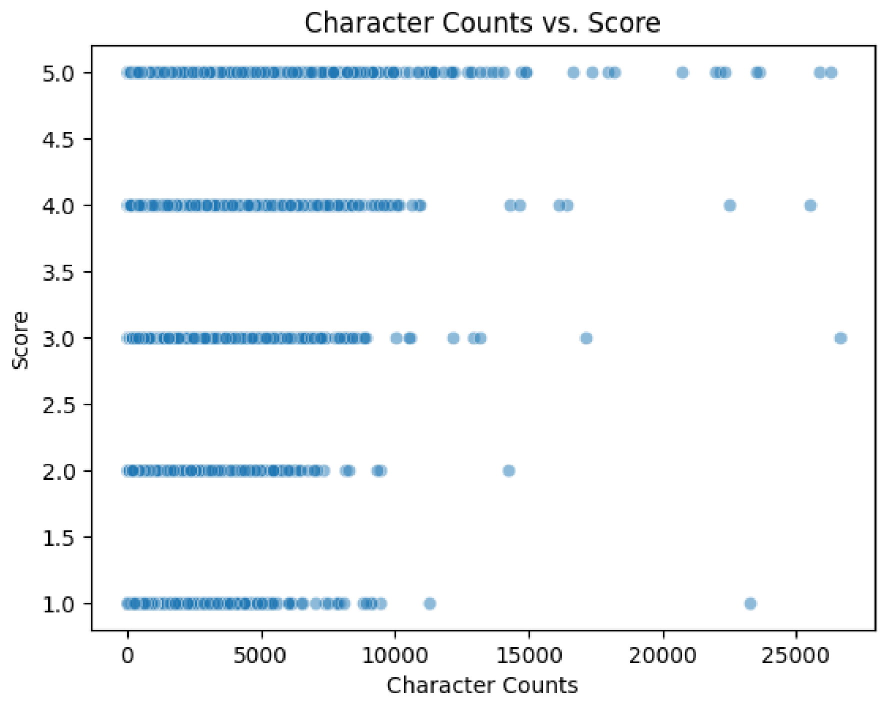
Conclusion:
This was a very difficult and at times very frustrating assignment however I am glad to say in the end through all my trial and error and different theories on how to approach it I am ecstatic to report that I came out victorious(in my own way) with learning not only about the model creation process but asl how my habits as a coder need to adapt to assignments like this going further. It was an invaluable journey of understanding, refining, then repeating that process. And even though in the beginning it felt more like a Sisyphean task, I am glad I was able to crawl out of that infinite hole of hopeless attempts and managed to create a model I am somewhat proud of with complex training and feature selection. And even if I didn't with the competition the insight I learned from this will be invaluable further in my machine learning career.

Appendix:

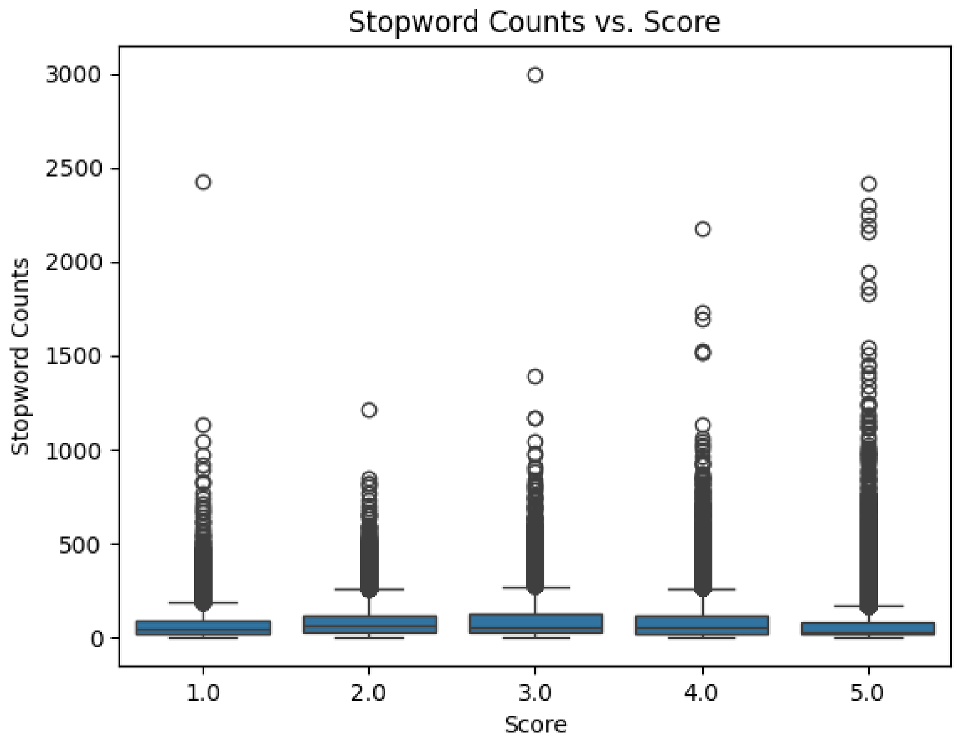
A: a graph displaying great results achieved by removing a large chunk of unreliable reviews



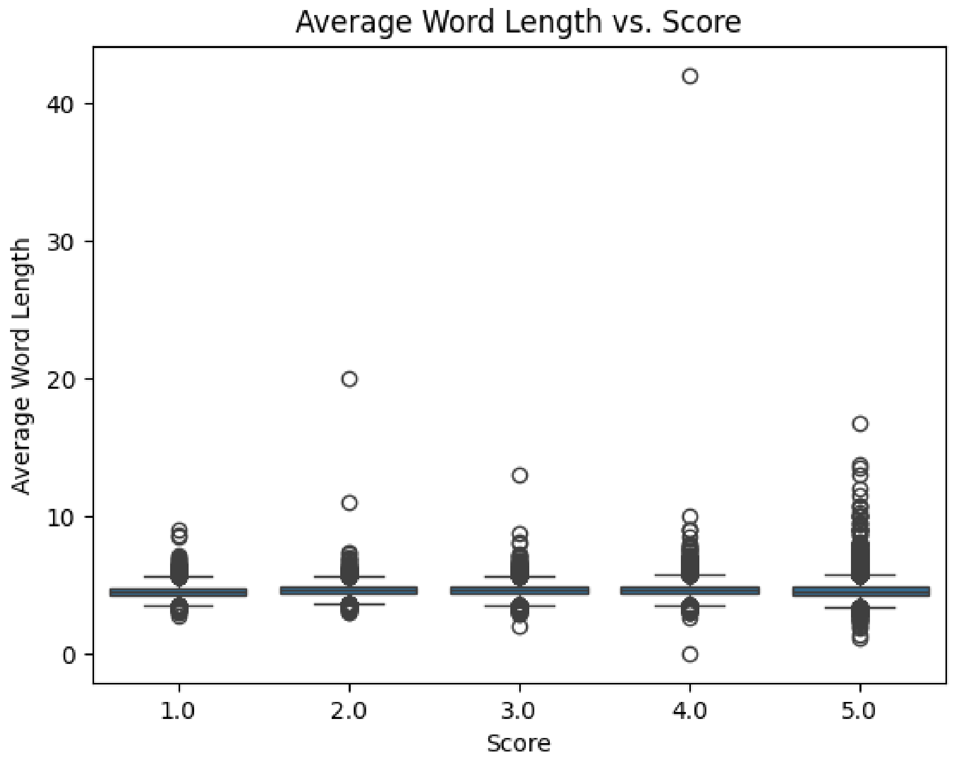
B:



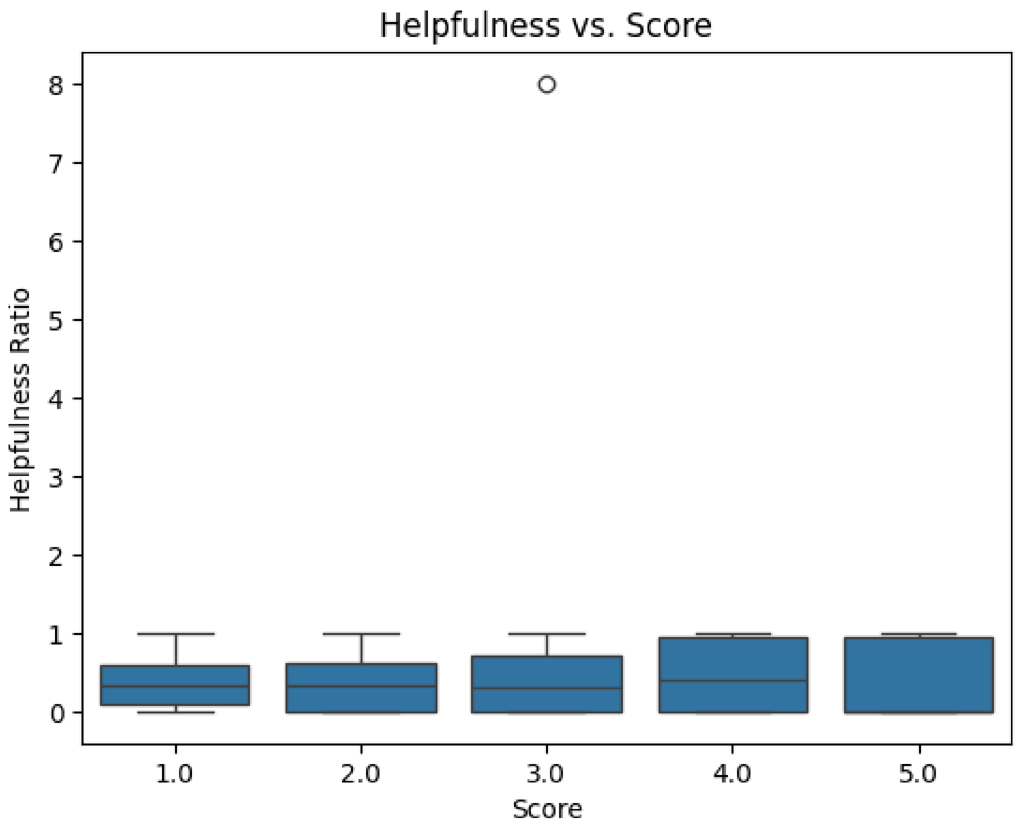
C:


Stopword Counts vs. Score
Stopword Counts
3000
2500
2000
1500
1000
500
0
1.0
2.0
3.0
4.0
5.0
Score

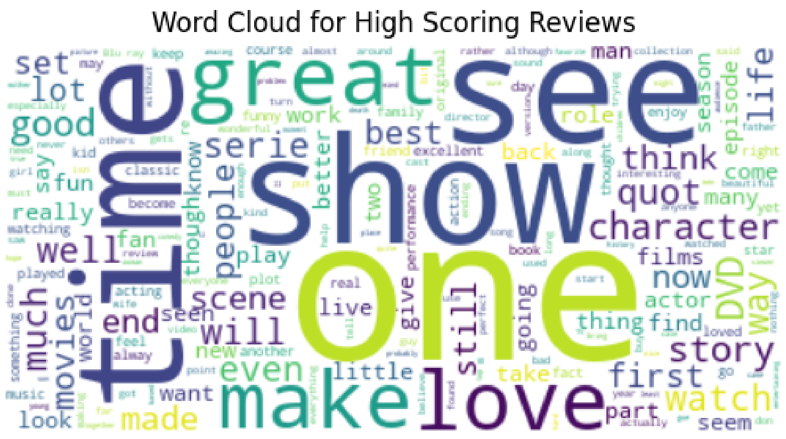
D:

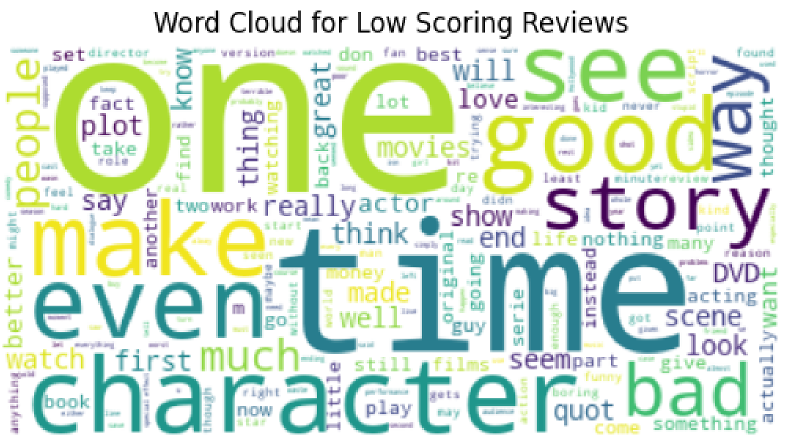

Average Word Length vs. Score
Average Word Length
40
30
20
10
0
1.0
2.0
3.0
4.0
5.0
Score

E:

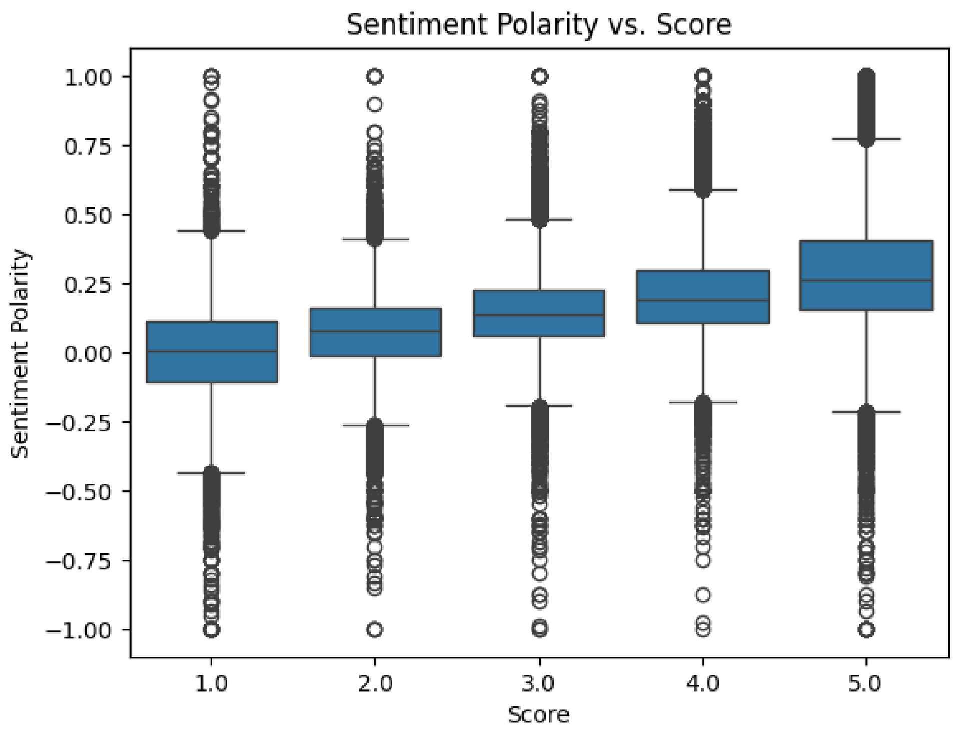


F and G: Word clouds high is 4 and 5 and low is 2 and 1

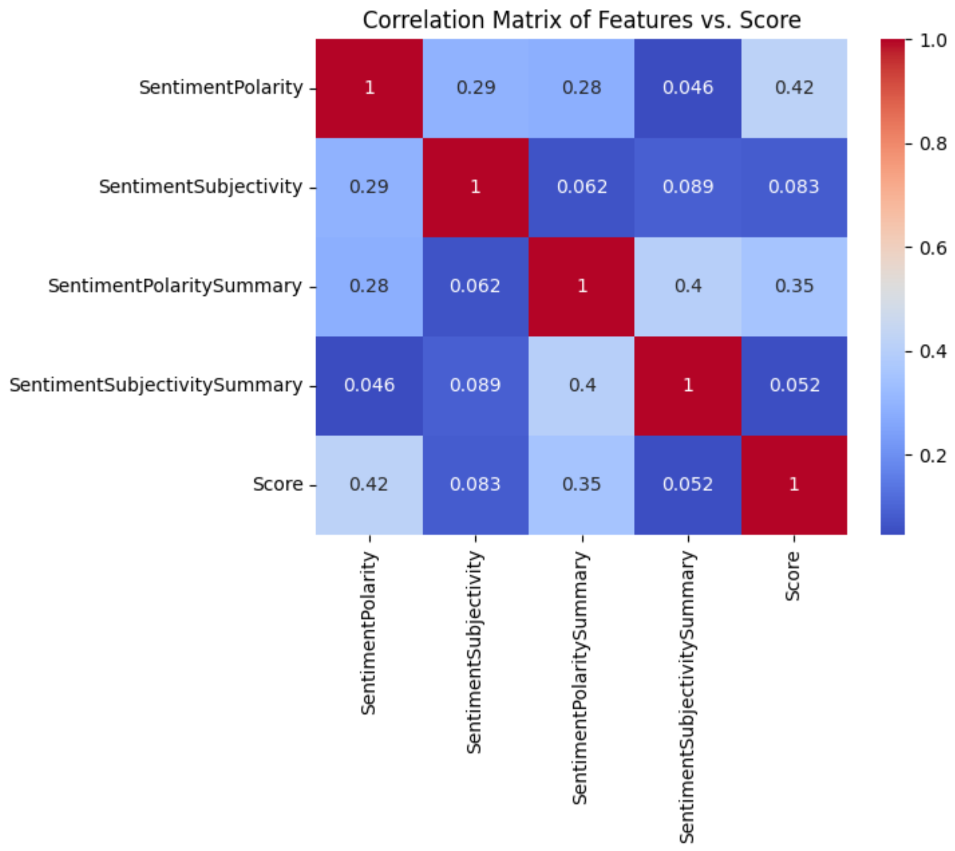




H:

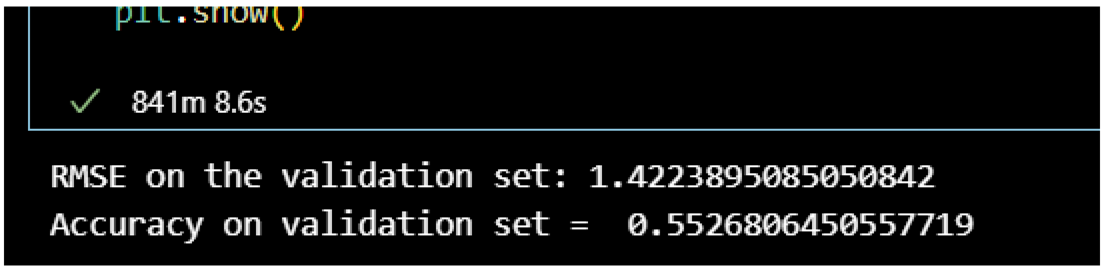


I:



	SentimentPolarity	SentimentSubjectivity	SentimentPolaritySummary	SentimentSubjectivitySummary	Score
SentimentPolarity	1	0.29	0.28	0.046	0.42
SentimentSubjectivity	0.29	1	0.062	0.089	0.083
SentimentPolaritySummary	0.28	0.062	1	0.4	0.35
SentimentSubjectivitySummary	0.046	0.089	0.4	1	0.052
Score	0.42	0.083	0.35	0.052	1

J: I was very close to giving up after this run

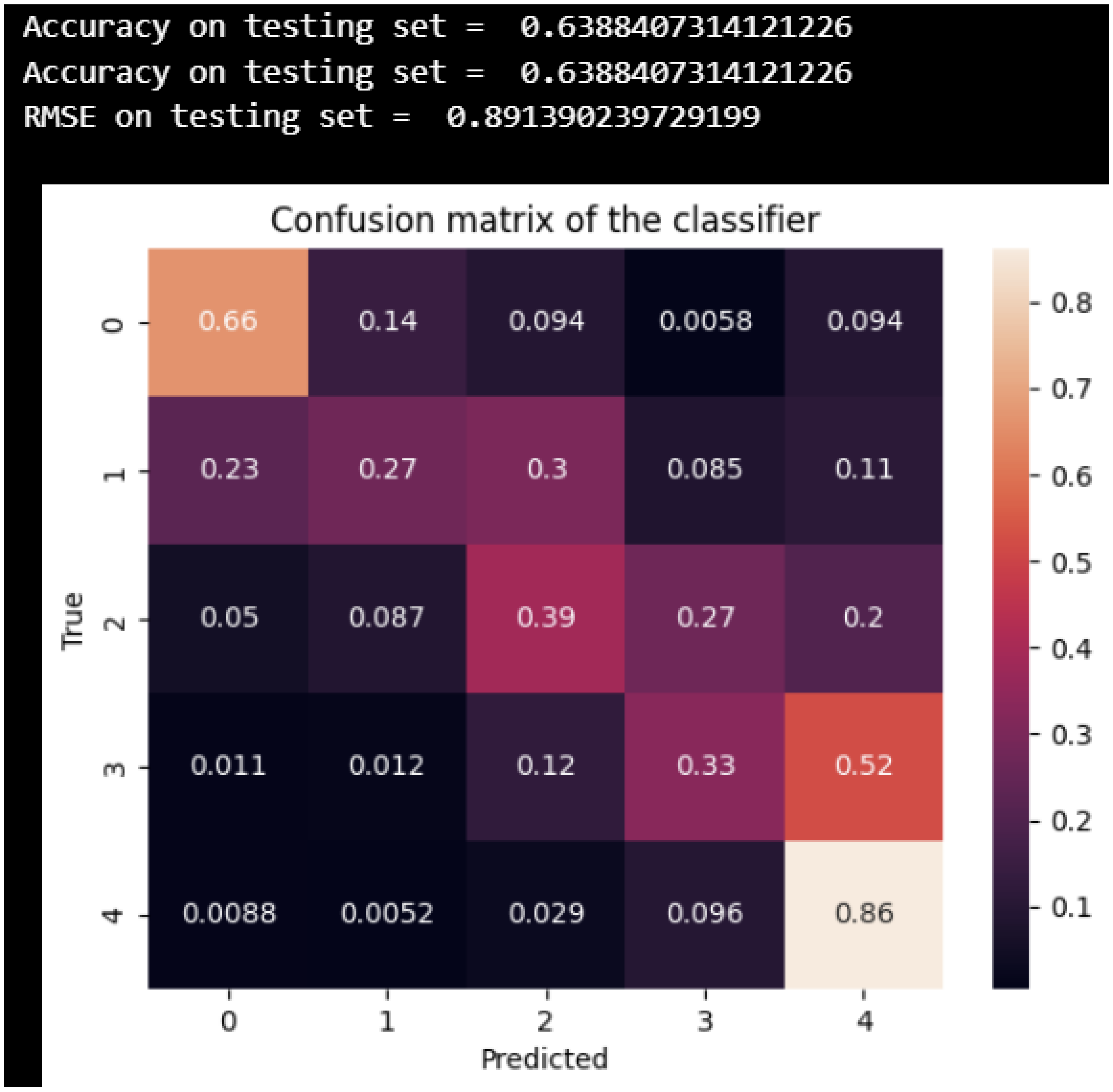


```
    plt.show()

  ✓  841m 8.6s

RMSE on the validation set: 1.4223895085050842
Accuracy on validation set =  0.5526806450557719
```

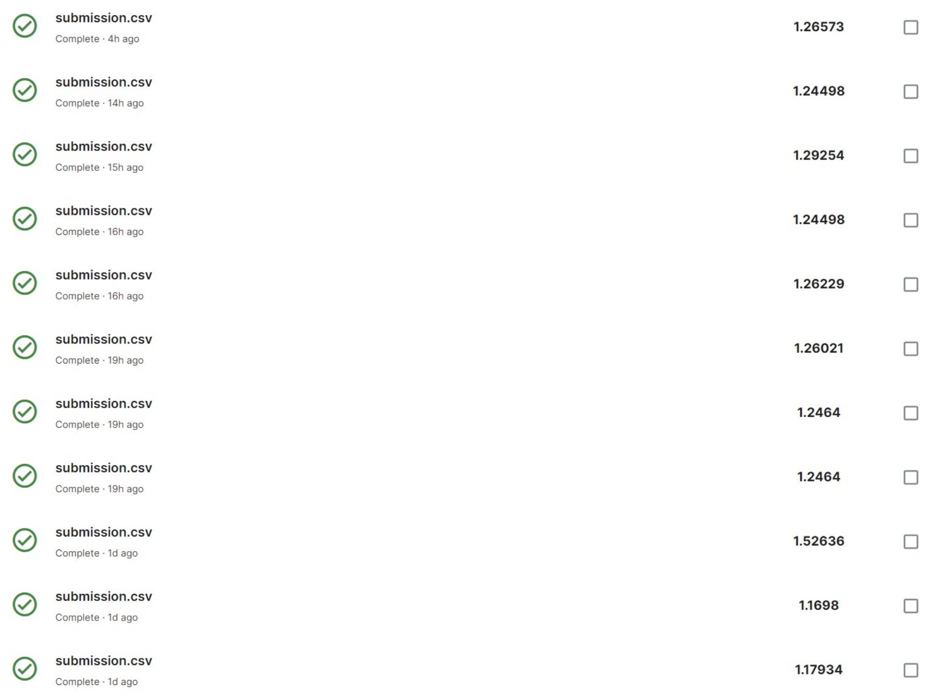
K: Model became capable of recognizing 2,3 and 4 star review with some confidence



```
Accuracy on testing set =  0.6388407314121226
Accuracy on testing set =  0.6388407314121226
RMSE on testing set =  0.891390239729199
```

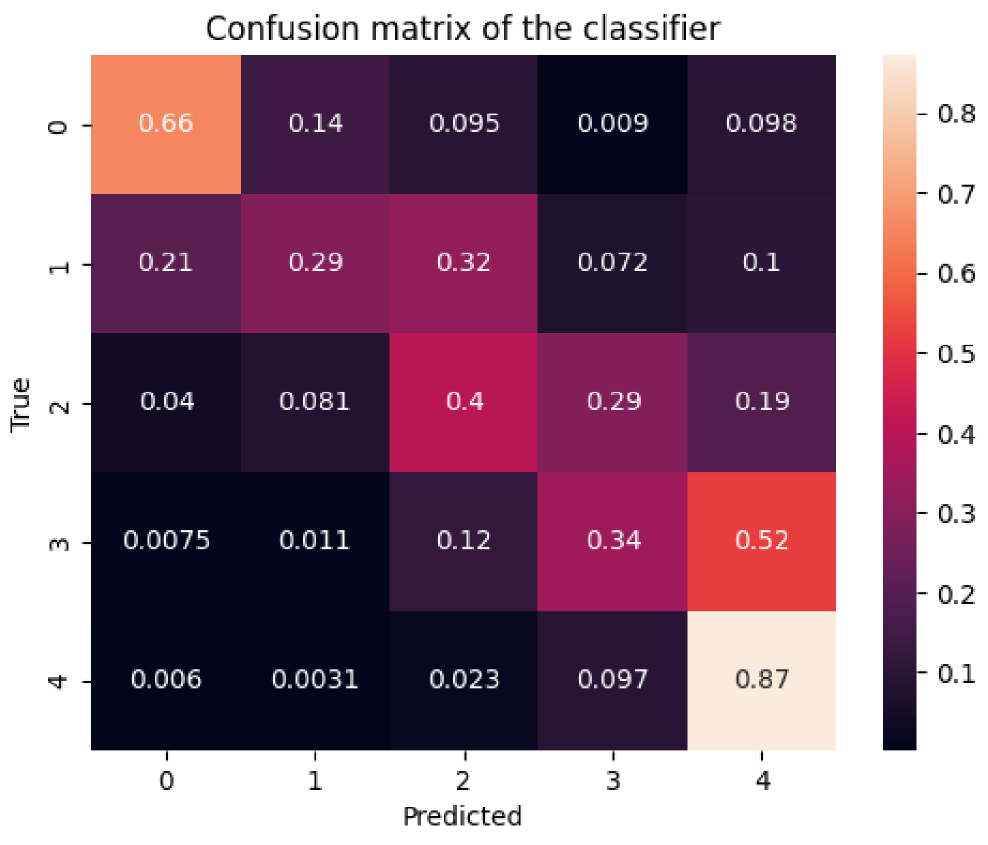
True \ Predicted	0	1	2	3	4
0	0.66	0.14	0.094	0.0058	0.094
1	0.23	0.27	0.3	0.085	0.11
2	0.05	0.087	0.39	0.27	0.2
3	0.011	0.012	0.12	0.33	0.52
4	0.0088	0.0052	0.029	0.096	0.86

L: Iterative changes made to my model, which in no way portrayed any improvement in actual results



submission.csv Complete · 4h ago	1.26573	☐
submission.csv Complete · 14h ago	1.24498	☐
submission.csv Complete · 15h ago	1.29254	☐
submission.csv Complete · 16h ago	1.24498	☐
submission.csv Complete · 16h ago	1.26229	☐
submission.csv Complete · 19h ago	1.26021	☐
submission.csv Complete · 19h ago	1.2464	☐
submission.csv Complete · 19h ago	1.2464	☐
submission.csv Complete · 1d ago	1.52636	☐
submission.csv Complete · 1d ago	1.1698	☐
submission.csv Complete · 1d ago	1.17934	☐

M:


Confusion matrix of the classifier
True
Predicted