Amazon Review Score Classification utilizing the PySpark Pipeline

Ciaran Grant and James Hooper

School of Mathematics, Computer Science and Engineering at City, University of London
Ciaran.Grant@city.ac.uk and James.Hooper@city.ac.uk

Introduction

We are predicting Amazon Fine Food review scores using text classification via Random Forest; varying the number of trees, the size of hashed vectors and the size of the training set to optimize performance. Text classification of reviews is particularly useful in automated review analysis due to the huge volume of and reliance on reviews for marketing such as in product suggestion, or for classification of texts on social medias such as Twitter [6]. We expect that the more trees, the larger sized hash vectors and the larger the training data set, the more accurate our classification will be.

Initial Analysis of the Dataset

The Amazon Reviews dataset consists of roughly 550000 rows of data. Initially the columns that appeared to be of use were the Review Score, the Review Summary, the Review Text, the Review Helpfulness and Review Unhelpfulness. The pipeline was built to predict Review Score from Review Text.
The score column was found to have values that were outside the normal $1-5$ stars and so any erroneous values were removed before starting the analysis.
As can be seen in Figure 1, the data is hugely weighted to 5 star reviews. If a reviewer feels strongly enough to make a review it is probably in praise, hence the top two scores by count are 5 and 4 followed by 1 as it is likely a terrible product would cause a reviewer to review as well. This skew towards 5 will affect the analysis if it is not dealt with. Some potential solutions are:

1. Randomly sample reviews so that there is an equal amount for each score. The problem here is that the data could no longer be considered big, which is the point of this analysis [1].
2. Include Class weights to the Classifier so classes with less reviews will hold a higher weight in the classifier. The problem here is that there is no way to do this in a Spark ML Pipeline with Multinomial classification as of yet [1].

Since there are issues with both of these potential solutions, nothing will be done to mitigate the skew to 5 star reviews. It is important to note that this will cause our final classifier to classify test reviews as 5 stars too often.

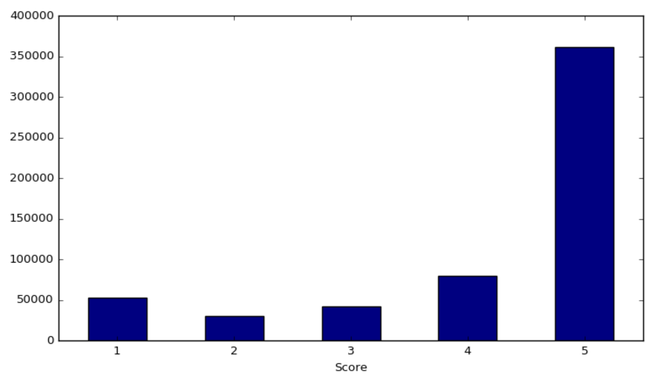

Figure 1: Score Distribution

Pipeline

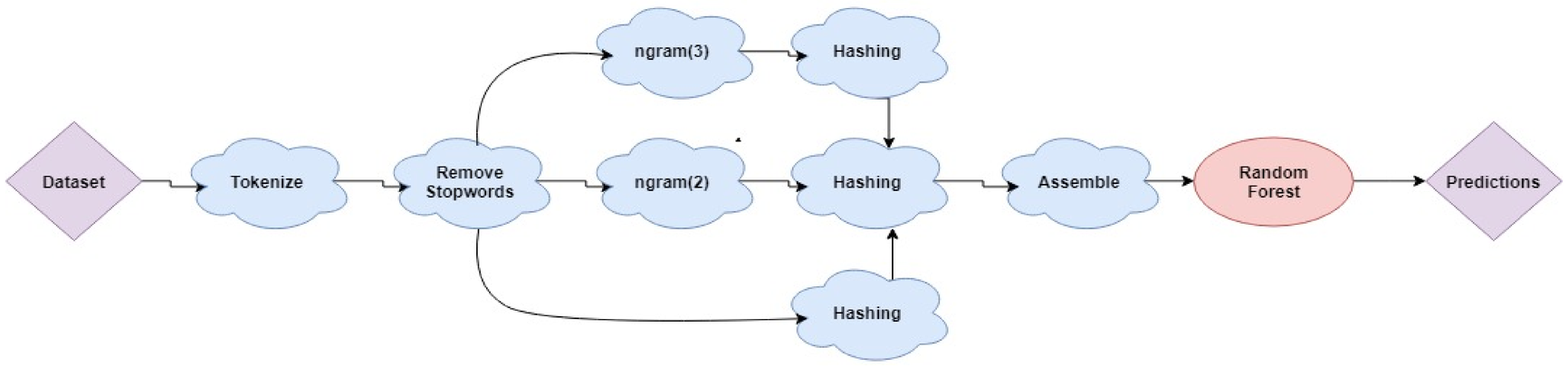

Figure 2: Pipeline Process Map

1. **Tokenize**: The first step is to take the text from the reviews, which are a couple of sentences maximum, and break them up into words. It's hard to understand whole sentences, it's much easier to deal with words individually to get an understanding of the reviews' sentiment.
2. **Remove Stopwords**: For each review, we have all the individual words that make it up. Not all of these words are helpful, we don't want to waste computational time on words that won't help us to classify the review. So we remove common words that won't add anything meaningful such as 'it', 'is' or 'the'.
3. **N-Grams**: Sometimes words need to be part of a phrase to provide meaning, this next step acknowledges that and connects consecutive words to form phrases of length N. We chose N in $\{2, 3\}$, so we now have vectors of meaningful words, pairs of words and triplets of words [3].
4. **Hashing**: This step enables the data that has been produced by Remove Stopwords and both N-Grams, which are still all vectors of words, to be transformed into a numerical vector, which is better suited for classification [2]. The length of the hashed vector is a hyperparameter that was adjusted in the model selection phase.
5. **Assemble**: This step appends the three hashed vectors end to end so if each original hashed vector had a length of 1000 then the assembled vector would have length 3000.
6. **Random Forest**: This step is the classifier that will be trained to make predictions on the test data. Random Forest was a good choice for this task as it performs well in multi-class classification [5]. A hyperparameter that will be changed in the model selection phase is the number of trees in the random forest .

Model Selection and Training

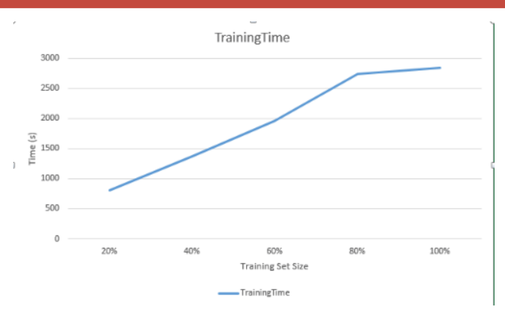

Figure 3: Training Time compared to Training Set Size.

The raw data was split into 80% training and 20% testing datasets. The training data was randomly sampled 4 times (20%,40%,60%,80%) to give different sized training sets. Including the original training set we had 5 training sets so that we could vary the training set size to see its effect on training time and testing accuracy. The testing set was set aside to be used for testing only, all training sets were tested on the same testing set.
We varied the size of the vector used in Hashing from 500, 1000, 1500 and the number of trees used in the Random Forest from 5, 10, 15 to see how each of these affected the training and testing times as well as testing accuracy.
For model selection, the Hashing vector sizes and number of trees in the Random Forest were built into a parameter grid. This grid was used in TrainValidationSplit with an 80% training set and 20% validation set split. We performed this model selection on each of the 5 different sized training sets, producing 5 models.
The best parameters for each model are below in Table 1. We were unable to extract the size of the Hashing vectors, however there is no clear optimal number of trees for the Random Forest [4] but generally a smaller number of trees and a simpler model worked best. At the extreme ends of training data size, 20% and 100%, only 5 trees were chosen whilst at 40%, 60% and 80% 10 trees were chosen.
It was clear that the training time increased as the size of the size of the training set increased as we can see from Figure 3. This is a linear trend up until 80%, which is roughly 400,000 reviews, and further increase up to 100% doesn't see such a high increase in training time.

Training Set Size	20%	40%	60%	80%	100%
Hashing Vector Size	N/A	N/A	N/A	N/A	N/A
Number of Trees	5	10	10	10	5

Table 1: Chosen Hyperparameters for each Training Set Size

Model Testing

For each training set size, the hyperparameter with the best Test Accuracy were chosen, as can be seen in Table 1 (unfortunately no way was found to extract the Hashing Vector Size).
In Figure 4 (Left) the Test and Training Accuracy is represented for each Training Set Size. The Training Set Size with the highest Test and Training Accuracy was 80% of the maximum size, so it is not necessarily true that a larger test set will provide better Test Accuracy.
It is important to note that the Test and Train Accuracy for each Training Set Size are all within a very small range between 63.7% and 64.1%. Therefore, it seems that any attempt to increase the accuracy by changing the size of the Hashing Vector of the Number of Tress in the Random Forest has had an insignificant effect. This is likely due to the large skew towards 5 star reviews. The Random Forest Classifier is very likely to predict a review as 5 stars since the majority of the training data is of 5 star reviews.
Figure 4 (Right) represents that the Testing Time marginally decreases as the training size gets bigger.

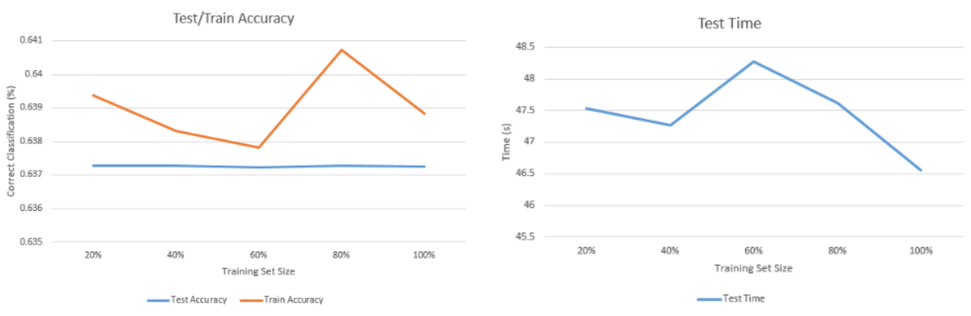

Figure 4: (Left) - Training/Test Accuracy for each Training Set Size. (Right) - Test Time for each Training Set size.

Conclusion

The main conclusion that can be drawn from the results is that an unbalanced dataset largely effects final predictions if nothing is put in place to mitigate that. Another equally clear conclusion is that an increase in Training Set size yields a roughly linear increase in Training Time. To improve the quality of this analysis some future work could be considered: 1. Apply oversampling to solve the problem of unbalanced classes that should reduce the number of false 5 star predictions. 2. extract a confusion matrix from the test set and predictions to better understand what was causing false predictions.

References

[1] Ganganwar, V. (2012). An overview of classification algorithms for imbalanced datasets. International Journal of Emerging Technology and Advanced Engineering, 2(4), pp.42-27.
[2] Joulin, A., Grave, E., Bojanowski, P. and Mikolov, T. (2018). Bag of Tricks for Efficient Text Classification. [online] Arxiv.org. Available at: https://arxiv.org/abs/1607.01759 [Accessed 24 Apr. 2018].
[3] Moore, Gareth / Young, Steve (2000): "Class-based language model adaptation using mixtures of word-class weights", In ICSLP-2000, vol.4, 512-515.
[4] Oshiro, T. (2012). How Many Trees in a Random Forest?, Machine learning and data mining in pattern recognition, MLDM 2012, Berlin, Germany
[5] Prinzie, A. and Van den Poel, D. (2008). Random Forests for multiclass classification: Random MultiNomial Logit. Expert Systems with Applications, 34(3), pp.1721-1732.
[6] Sriram, B, and Fuhry, D. (2010). Short Text Classification in Twitter to Improve Information Filtering. Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval, pp.841-842