Report: Predicting Star Ratings from User Reviews

Keerthi Krishnan, Pranav Nampoothiri, Ayush Arora
Cruz ID: kvkrishn, pnampoot, ayarora
Student ID: 1526350, 1530280, 1585167

December 6, 2019

Tools Used: list of libraries used are collections, csv, json, pandas, sklearn. Csv and json were used to translate the json content into csv format for pre-processing. Sklearn was used to train the model as well as make predictions on the testing set. Pandas was used to create the predictions file appropriately. **Diversity**: Our group consists of 3 south Asian students who are all males. Some of us came into the Computer Science field, unaware of what the field would bring us. One of the members, Pranav, came into college with no background or knowledge of the Computer Science discipline. Though he struggled early on to develop an interest for the topics in CS, he has learned how to approach and think about problems more natural to the study, and really appreciate the growing field and its contributions to the world thus far. One of our other members, Keerthi, also came into college with no computer science experience, but had experience in mathematical analysis of computational concepts. Ayush, on the other hand, came into college with some Computer Science knowledge through an AP course and a boot-camp for Swift programming. As he has progressed through his undergraduate program, his passion for statistics led into an an affinity for prediction models. His interests lie in working in the research and developmental field of bio-mechanics studies.

1 Abstract

The problem statement involved using text classification to predict ratings that would arise from certain reviews. Due to the fact that text is almost always linearly separable, we implemented a Linear SVM model to learn our training and testing set. Through our results, we see that Linear SVM proves to have a high prediction accuracy for the data.

2 Data Pre-Processing and Feature Extraction

In order to minimize bias, our team decided to pre-process the data and perform feature extraction to appropriately train our model to predict accurate ratings. We pre-processed the data by balancing the input data. More specifically, we calculated the frequency of each star rating and we used the minimum frequency as the universal frequency. We discovered that the 2 star ratings had the least amount of reviews, therefore our universal frequency was based on the number of 2 star rating reviews.

In order to separate our features from our labels, we had to load our features and our labels into separate datagrams. Therefore, we loaded the text into one datagram and the ratings into another datagram. This created a separation of features and labels, where our feature was the text and our label corresponded to the ratings.

For feature extraction, one of the salient features we wanted to extract was the text from the reviews. In order to pre-process the text, we implemented a vectorization technique called Term Frequency Inverse Document Frequency(TF-IDF) Vectorization, where we neglect common words that occur and focus more on uncommon words present. This creates a normalisation across the number of times the word is present

in the text for a review by the number of times the word occurs in all of the texts in the review.

Another pre-processing technique we used was the concept of n-grams, where it considers n words as one word, where n determines how many words you would like to consider. By using n-grams, the machine learning model can learn fine-grained meanings within the text, and can provide a more accurate prediction rating for the dataset.

3 Approach and Experimental Set-Up

In order to tackle this problem, we first researched different models ranging from the ones we learned in class to other techniques found online. From what we researched about text classification, we learned that text is almost always linearly separable, therefore we wanted to use a model that represented the data given. We decided to use Linear SVM, as it not only creates a linearly separable dataset, but also makes sure that the accuracy of the separable classes stays intact. Linear SVM is also efficient in runtime. Mathematically, we are trying to minimize an objective function given our features. In essence, we are trying to solve this optimization problem:

$$min_{w,b} \|w\| + C \sum \xi_i$$

subject to $y^i(w^T x^i + b) \geq 1 - \xi_i$ for $i = 1, 2, .., m$ and $\xi_i \geq 0$ for $i = 1, 2, ..., m$

Since text is linearly separable, Linear SVM will create a hyperplane such that the distance between the data points and the hyperplane will be maximized. This also brings into play the concept of support vectors, which in turn will be the features of the text that are the most important for classification and creation of the hyperplane. In order to do that, Linear SVM uses the dual form and utilizes the shape of the optimization problem through the support vectors present in the dataset. The dual form is shown below:

$$f(\vec{x}) = sign(\sum_i \alpha_i y_i \vec{x}_i^T \vec{x} + b)$$

where if α_i is not equal to 0, then $\vec{x}_i$ is a support vector.

By using the dual form, we extract salient features of the text as support vectors to form the classification boundary(hyperplane) and in turn, work to minimize the objective function, which will in turn solve our optimization problem by giving us the maximum margin hyperplane for the dataset.

Another important reason that we used Linear SVM is because of the way we pre-process the data. In our pre-processing method, we use a vectorization method to highlight uncommon words in order to dictate connotation of the reviews and we also use an n-gram method for the machine model to better understand fine-grained meaning in the text. Both these pre-processing techniques lead into the idea of having a small cost value as it leads to a lower chance of instances being misclassified. Therefore, if C is lower, we get a larger margin, eventually giving us the max-margin hyperplane that can linearly separate the data. We want to be able to take into account the fact that there might be misclassified instances as well, therefore Linear SVM seems to be the best model to match the data too.

In our experimental set-up, we first pre-processed our data and then trained our Linear SVM model on our training dataset, where the feature was the text and the label was our ratings. We made sure in the pre-processing that our dataset was balanced out and we would have an equal number of instances for each rating. We then ran a Linear SVM model and trained our model on this dataset.

4 Results and Conclusions

We split the training set into train and test subsections using the 67%-33% split. Predicting on the test set resulted in a 60.2% accuracy, 40% better than random assignment. After receiving the official testing

json file, we trained the SVM model on the entirety of the training set. We found that on the testing set, for a 5-4-3-2-1 rating model, the respective percentage of instances were 39%-22%-15%-12%-10%. When we trained and tested our model on the training set, for a 5-4-3-2-1 rating model, the respective percentage of instances were 44%-22%-14%-11%-8%. As you can observe, the percentages between the training and the official testing set do not differ that much and it also shows that good accuracy was achieved using our model.

In conclusion, Linear SVM proved to be a robust model for predicting the ratings based on the text reviews. We chose SVM due to the fact that a text data is almost always linearly separable. Therefore, we wanted to use a model that would calculate the max margin hyperplane as well as take into account misclassified instances. As a result, we see that the SVM model performs well when trained on the training data, with an accuracy of 60.2%. Hence, using Linear SVM for text classification and rating prediction proved to be fruitful.

5 Ideas For Future Work

Although our model outputs fruitful results, there are a few modifications we can implement in order to make the accuracy better. One of the implementations we can do is implement K-Cross validation in order to train the model on the training set more extensively. This will provide a higher accurate model in order to match our dataset better. Another implementation that could be done is increasing our n-gram count during the pre-processing stage. By increasing n, which is the number of words we want to consider as one word, the machine gets a better idea on connotation and fine grained text meaning, which will lead to more accurate results. We can also look into other classification algorithms, such as XGBoost, in order to see whether other models improve the prediction in the dataset.