CSC2515 Project Part 1 Kaggle Competition

Neel Nitin Shah (1002241852), Team ID: 5933454

December 9th, 2020

1 Data Processing and Feature Selection

To process the data it was first read in from the json file. I noticed some of the features would likely not be useful (determined intuitively), like UnixReviewTime and reviewTime and reviewHash. Although, for some attempts with my first method, UnixReviewTime was included as a feature because it was already an integer and therefore easy to incorporate. Since, at first I wanted to reuse my least squares code from homework 2, I primarily focussed on converting the data to numbers. This involved converting the price to an integer and using a one-hot encoding approach to handle the categories. Since some of the price data fields did not contain a price, the price for those entries was set to the average price of the dataset. Further work was done using NLTK vader to convert the review summaries to numerical sentiment values. Also, there were attempts made to featurize the userID and item ID but these attempts were unsuccessful (discussed further in the next section). These feature extraction and data processing steps were mainly done to facilitate my first attempt at the project. Once those efforts proved futile I shifted my focus on using exclusively NLP techniques to make predictions. This involved extracting the review text data and passing it to a term frequency, inverse document frequency (TF-IDF) package. This returned the feature vectors for the final model. The final model did not include any of the price or category information, only the TF-IDF information was used as features. This was chosen because it greatly simplified the problem and it was able to provide the desired performance so any additional features were deemed unnecessary. Since some data was missing the review text, that was replaced with a placeholder NaN.

2 Model Selection

2.1 Locally Re-weighted Least Squares

My first attempt was to adapt the locally re-weighted least squares model from homework 2 for this project. Even though the problem is a classification problem I suspected a regression model would work because mean squared error was being used as the error metric. For this approach I experimented with many different feature vectors. At first, I tested the performance of the model using only the numerical data provided: userID, unixReviewTime, category, price and item ID. Category was converted to a feature by assigning a number to each category (e.g. pop = 0, rock = 2 etc.). The performance of this model was tuned by selecting the best τ and λ. The τ was selected using the semilogx graph which was implemented in homework 2. λ was chosen by trial and error. This yielded a mean squared error of 1.00120 on the test set which is only enough to beat the naive baseline. The main issues with my approach, aside from not using any of the language data, was how the category, userID and Item ID were featurized. userID and item ID were featurized by simply taking the number of the id. This is problematic because the LRLS model makes use of distance metrics extensively but the distances between the number in the ids do not really have a meaning. For example, just because one user has the

id u045678 and another has u045679 does not mean they are close in anyway but the distance metrics would consider these users closer than a user with the id u045670. A similar problem occurs for the item id. Additionally, the way the categories were converted to numbers was problematic as well. Since they were simply assigned numbers (e.g. pop =1, rock = 2, electronic = 3 etc) this gave a misleading notion of distance. It implies rock is 'closer' to pop than electronic when in reality that may not be the case. To properly implement this feature it would have been more productive to use a one-hot encoding scheme. So after the first attempt I tried to featurize the user ID and item ID in a similar one-hot encoding method. At first, I attempted to use sparse vectors where the element corresponding to the user Id number is set to 1 to identify the user associated with the review. A similar approach was done for the item ID. This implementation ran into issues because the LRLS method was not able to handle sparse vectors in its distance metric calculations. So next I tried using a dense vector. This involved initializing two 1x99999999 vectors of zeros and setting the user ID and item ID numbered elements to 1. Although this effort would have worked with the distance metric calculations it proved futile because now the feature vectors were too large to handle. Due to these troubles, one last attempt was made to see how well the model could perform without user or item ID but while using the review text. I incorporated NLTK Vader into my LRLS program to return the sentiment of the summary. Although this was promising as the results lent themselves to featurization, ultimately this approach was not fruitful. It should be noted that due to resource limitations this model was only trained on the first 1000 training cases from the json file. Of these 1000, 70% were used as the training set and 30% were used as the validation set. Due to these roadblocks I did not think it was worth putting in more effort into this model and switched focus to an exclusively NLP approach.

2.2 Linear Regression with TF-IDF

To handle the issues discussed above, I attempted to use only the language data to make predictions. I also switched to using a linear regression library from sklearn. This was so I could focus on implementing the new features while not having to worry about whether my code was correct for the training and testing components. Additionally, this allowed me to focus on optimizing one design variable, the number of features extracted. If I had to also set an ideal τ and λ it would have likely been infeasible to find suitable values for each given my limited resources. To process the review text, I used an NLP method called TF-IDF (term frequency-inverse document frequency). This method was chosen because it is straightforward to implement using a sklearn library [1] and it allowed a tuning parameter, the number of features extracted from the reviews, which could be adjusted to obtain the desired accuracy. TF-IDF is composed to two factors, the term frequency and the inverse document frequency. The term frequency is simply the total number of times a word occurs in a document divided by the total number of words in that document. Inverse document frequency is the log of the number of documents divided by the number of documents that contain the word of interest. Finally, the TF-IDF is calculated by multiplying those two terms together. This is a productive approach because it provides weights for how important words are while also giving less weights to common words like 'the', which is used often but does not provide any information. Several models were trained with varying numbers of features extracted. The primary factor limiting the number of features used was the computing power. At first, 1000 and 2000 features was attempted which yielded test accuracy of about 0.7 which is not enough to beat the strong baseline. So the features was increased to 3000 to study how that affected the test accuracy. Since these models take upwards of 25 minutes to train on my computer this number of features selection process was done heuristically. This is to say, it was not performed similar to how τ was selected for the LRLS scheme. Had I had enough computing power I would have trained the model for each number of features from 1000-5000 and chosen the result which gave the best result with low risk of overfitting. But with limited computing resources I had to perform a coarser search. Since this

study is my final submission, this model will be discussed further in the next section.

3 Results

The final model submitted is a TF-IDF linear regression model with 3000 features used. The model has an additional restriction whereby if the predicted label is greater than 5, the recorded prediction is 5. Since the true label can be at most 5 any prediction above that is just contributing to the error. Similarly, if the prediction is less than zero, the recorded prediction is 0. This produced a training error of 0.604 and a test error on the public leaderboard of 0.61389 which is substantially better than the strong baseline on the public leaderboard. This approach was significantly better than my first attempt which used the price category and basic sentiment analysis. This discrepancy in performance was likely due to the fact that the actual words people used in their reviews were more indicative of their rating than the price, category or general sentiment of the review summary. Additionally, since we can tune the number of features provided this allowed for greater flexibility in the training (i.e. we could provide information about more words). Furthermore, being able to interpret the text likely allowed the model to understand subtle nuances as opposed to something so detached from the user as category or price. A heuristic test was performed for the number of features extracted to see how increasing it would affect the final training error. Therefore, an additional model was trained with 3500 features, which yielded a training error of 0.601 which is only marginally better than 3000 features so the 3000 feature model was the model used to make the final predictions. Additionally, a cross-validation method was used to determine the training error where the training set was split 4 times, into 4 different train and validation sets. The resulting error was averaged. This was to ensure we were not only getting good error values for a fixed training set (i.e. overfitting to a training set).

3.1 Weaknesses and Possible Improvements

The primary weakness of this model is the time it takes to train. Due to this limitation I could not thoroughly investigate how number of features used affects the error. I also could not verify that by increasing the number of features it did not result in overfitting. Some attempts were made to train the model on minibatches of data and using those results to choose the ideal number of features but this was not suitable because the ideal parameter for a minibatch is not necessarily the same as the full batch. To remedy this, a cross validation method was used as described in the previous section. Additionally, this model did not take into account user or item information. Incorporating this would likely reduce the error as similar users will likely provide similar ratings. Lastly, some preprocessing could have been done to the review text to remove stop words, punctuation and casing. Again, since I was able to attain the desired performance without these extensions they were not investigated in this report.

4 Conclusion

To conclude, a TF-IDF linear regression model was trained on review text to predict the rating out of five stars for a user-item pair in amazon reviews. This feature approach worked much better than using the category, price and sentiment analysis.

References

[1] Cory Maklin. Tf idf— tfidf python example, 2019.