Date: 06/12/2017

Nirmal Budhathoki
A53212052
Kaggle Username: NBudhathoki

Private board: 14th

Report on '*Predicting helpful votes of Amazon Product Reviews*'

Problem Statement: Predict whether a user's review of an item in Amazon will be considered helpful or not.

Problem Definition: The e-commerce and online shopping is now taking over the world, and Amazon is leading the business. Millions of customers go online to purchase various items, and most of them provide reviews after using the product. Not all the reviews are always helpful, which brought up our problem statement.

Amazon provides an option to vote whether the review was helpful or not, and it ranks the reviews based on the helpfulness score (helpful votes/total votes). However, the problem is that most of the good reviews do not have enough helpful votes to be listed at top, and some of the bad reviews have higher votes. Can we resolve this issue by machine learning?

Machine learning task: Provided with review data and total votes, the task is to predict the number of helpful votes to solve the problem statement.

Process flow diagram/pipeline:

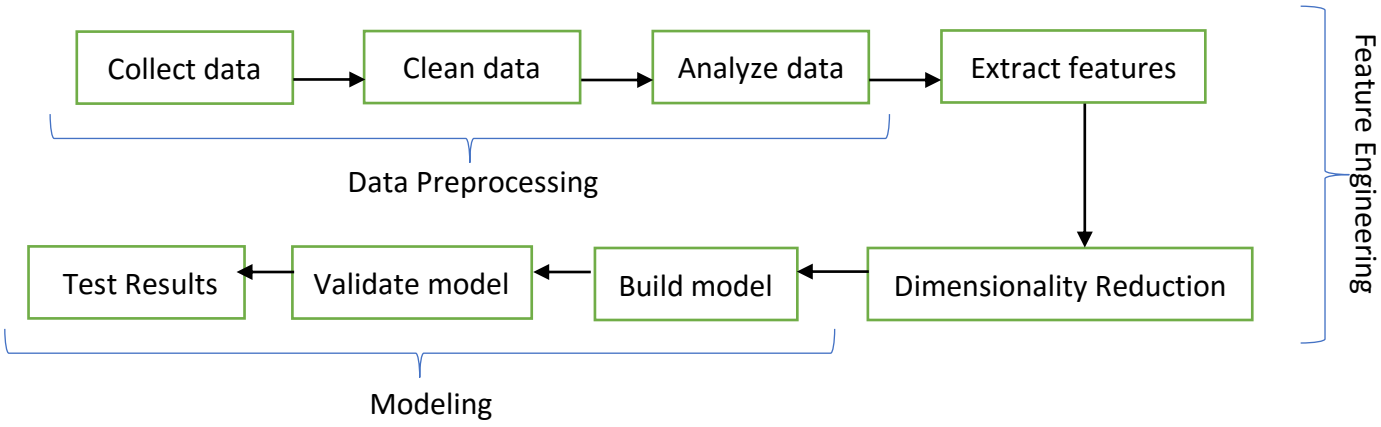

Figure 1. Pipeline set up for Machine learning task

Data preprocessing:

The data was collected from the given source:
https://inclass.kaggle.com/c/dse-220-final/data

We were provided with 200,000 amazon reviews as training set, and 14,000 as test samples to make prediction on number of helpful votes for the given number of total votes. I have learned from this project that data cleaning is very important. Most of the assignments that we did in class had pre-cleaned data, so our job was easier. The model is as good as the data is. I have spent significant amount of time on Exploratory Data Analysis (EDA), plotting and visualizing the data dispersion, and cleaning the data. Some of the things that I learned about the data from EDA are:

- About 68% of the training samples have total outOf votes equal to zero, which I decided to remove from training samples, because they do not provide any usefulness.
- About 62% of price data is missing. I did some research on data imputing, and I tried to add the average price for each category to impute missing values. However, since more than half of data was already missing, it did not make any sense to add the average value. And my model didn't show much improvement by adding the price data so I decided not to.
- Ratings was an important field, as I found out that most of the higher rated items have higher tendency to receive votes. But the catch is how relevant the reviews can be. The average rated products (ratings 3) are not found very helpful, while the low and high ratings can provide helpful feedback to collect more helpful votes.
- We can abstract some useful features out of the review text. Length of review, number of capital words used (CAPS signifies more importance in a sentence), special characters used, number of positive and negative words used can provide useful feature values.
- Since category data is not missing, I plotted the category ID from the reviews against total votes to see the pattern of data distribution for training and test set. The training set chart is attached below:

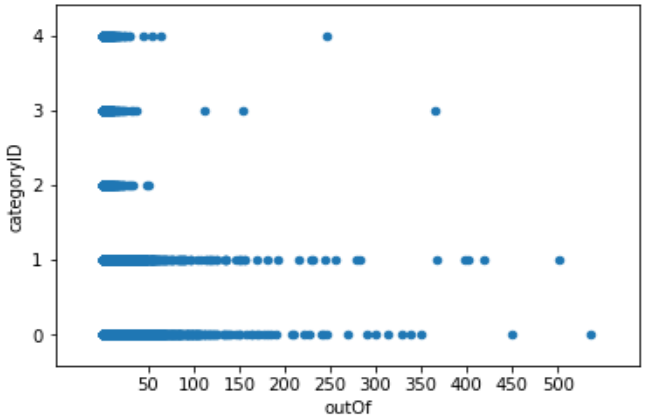

Figure 2. Category ID Vs. Total votes

I learned that there are few outliers, and we have a distinct pattern of high and low votes data. I plotted several other charts to visualize the data better, however I have not included them for the readability of the report.

Feature Engineering:

As much as I think of feature engineering now, I cannot disagree that it is one of the most important steps in any data science project. I have started with higher number of features and then used ablation technique to drop the features that I found not useful. I am still not sure if my feature selection was good enough. Based on the performance in leaderboard, I may have missed some good information in my feature engineering step. The features that I ended up using are:

Ratings, total votes, review text word counts, review text caps count, review text special character count, review text readability index, and review text positive and negative words difference.

I think I added too many of review text features unnecessarily. Finding the optimal and right number of features was the most challenging step for me in this project.

Modeling:

I started with simple linear regressor model, which was not producing satisfying results. Then I shuffled around with features to re-capture more features like performing one-hot encoding of category and ratings variable, and reducing the ones that are highly correlated with others. I also tried PCA to perform dimensionality reduction, but I did not see much significant performance gain by PCA. I have also tried polynomial feature interaction for the linear regressor to know more about polynomial feature weights.

After settling down on the features as I mentioned in above section, I tried various other models like Elastic net, Random Forest and Gradient Boosting. However, my models were still not performing to the best. Then I thought of finding the breakdown of dataset based on the outOf votes. After performing some tests, I divided the data into two sets as:

Training set with high votes: I found the number 15 to 150 performed well. This makes sense because it captures the variance in data as well as avoids some of the outliers at upper bound.

Training set with low votes: I used the number 1 to 40 votes for this training set to capture enough variance in data to learn the model. However, I do think that I should have tried to limit the lower boundary on this one somewhere around 5, and increase the upper boundary to 80 to capture more variance for model to learn better. I have used this boundary on one of my submissions which performed best at private board and I could have got 7^{th} standing based on that score. Unfortunately, I missed to select that model for grading.

Ensemble Models:

I used ElasticNetCV model to train the high votes training set, and gradient boosting learning algorithm for low votes training set. Random Forest (RF) was also giving me about the same scores as gradient boosting, but the performance was slower. However, now I think I could have used RF because of its robustness for overfitting as compared to gradient boosting which was not easy to tune the hyper parameters for its best performance. One of my challenge was to get mean absolute error (MAE) for the combined result of ensemble, one model had less data sets so it was giving me lower MAE, while another model had more data sets and it was giving me higher MAE. I had limited time for research and study on getting the accurate value of combined MAE.

Lessons Learned:

This was my first kaggle competition, so it was a great learning experience for me. Some of the key takeaways are:

- Data cleaning is very important
- Feature engineering can be challenging
- Be cautious about overfitting problems
- Try out various models
- Cross Validation and Parameter tuning are very important
- Ensemble models can be helpful and challenging at the same time