Ryan Inghilterra - DSE 220 Final Report - Spring 2018

Kaggle Competition Overview

Helpfulness prediction - Predict whether a user's amazon review of an item will be considered helpful. The file 'pairs Helpful.txt' contains (user,item) pairs, with a third column containing the number of votes the user's review of the item received. You must predict how many of them were helpful. Accuracy will be measured in terms of the mean absolute error, i.e., you are penalized one according to the absolute difference of nHelpful and prediction, where 'nHelpful' is the number of of helpful votes the review actually received, and 'prediction' is your prediction of this quantity.

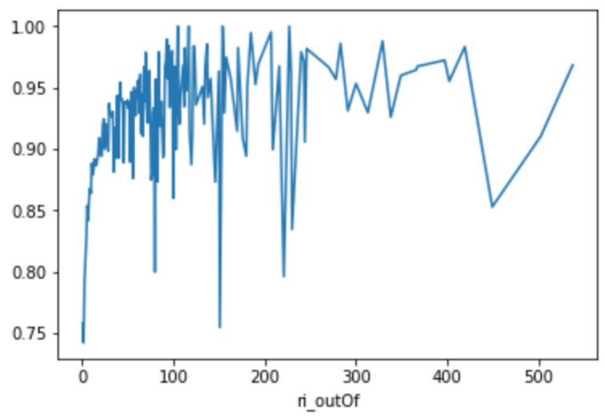
I broke this report into the main steps in my kaggle competition journey.

1. Exploratory data analysis and brainstorming
2. Feature engineering and extraction
3. Model experimentation
4. Building a data transformation pipeline
5. My final models
6. Final thoughts, lessons learned
7. Conclusion / Private Score results

Exploratory data analysis

The first thing I wanted to do is looking at the raw data. I wanted to view the distribution of the different raw feature values and most importantly the outOf and helpful variables (which represents what really we are trying to predict). What immediately stood out was that around 60% of the data was reviews that had outOf = 0 and num_helpful = 0. I knew immediately that I could throw this data away and would not need to train my model with it. For any outOf = 0 for a review, I could always just predict 0. This was nice because it reduced the training data set from 200,000 reviews to ~60,000.

With my reduced training set, I looked at the distribution of my data for outOf vs numHelpful again. I then calculated percent helpful (num_helpful / outOf) and took the mean for each outOf value and viewed the distribution. This showed a nice trend, but there some large outliers (5 total) that I decided to remove from my training data set.



Above shows the mean percent helpful (y-axis) vs outOf for the train data. (After removing 5 outliers). Notice that the mean is always between 0.75 - 1.

Feature engineering and extraction

Now I was ready to try and clean up my training data and create the features which I could try training some different models on.

I applied some basic text manipulation to the pandas dataframe of training data, such as pulling out the year, month, and day from the data into their own columns. I also noticed the price was missing from a lot of reviews, so I imputed the missing values with the mean price from the reviews that had a price. The initial helpful column of the training data dataframe was in json format where it contained {helpful: , outOf:} and so I pulled each of these into their own columns. So there was an outOf column and a separate helpful column. At this point I had a clean dataset I could work with, but next I wanted to figure out which features I could create from the data given which could help our model.

I initially thought the summary length and review length could be good features, so I added those. Next, I combined the reviewText with the summaryText into a single column. From this, I took the log of the counts of different punctuation characters in the text for each review (such as periods, commas, colons, dollar signs, etc..) . I had read that punctuation character counts can be good features for text based models. It also makes sense, for example colons might indicate a list format which can correlate to helpful reviews, since they present information nicely to people, etc… At this point, even before doing any real text transformation, I had a feature set that I was ready to trying some modeling on.

Model experimentation

With my feature set, I was ready to try training a model. This was a regression problem, as I was trying to predict a number, being the number helpful for each review. After playing around with Linear Regressor, Random Forest Regressor, and XgboostRegressor, I decided to use XgboostRegressor(Gradient Boosted Trees). Xgboost is quick and gives good scores. One thing that is also nice is you can view feature importance. Also, Xgboost does its own feature selection, so I did not have to focus much on manually selecting important features myself.

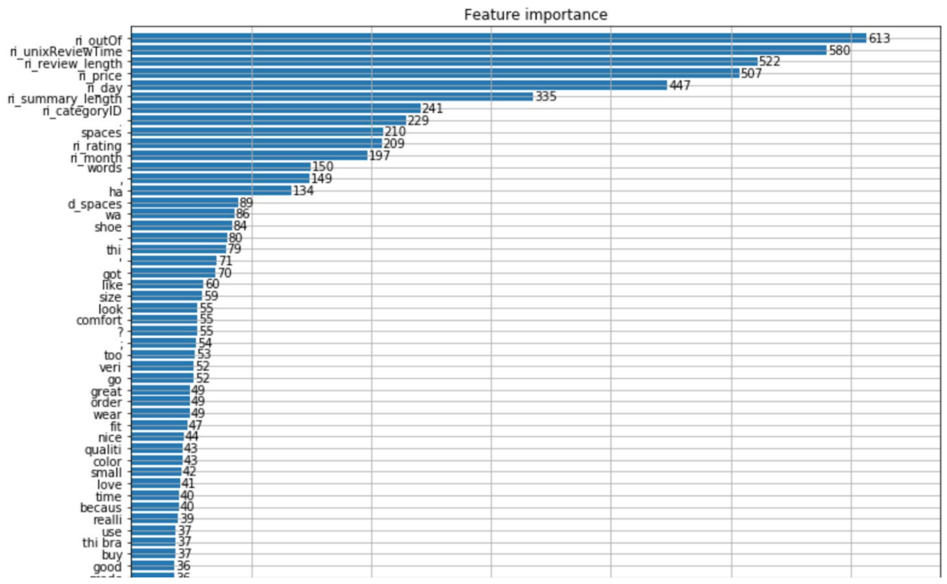
I broke my train data into a train, test split of 90%-10%. And then I broke my train data set into a train validation set split 90%-10%. One thing that is nice about xgboost, there is an early stopping flag you can set where it stops training once the validation set score has not improved after a certain number of rounds (I used 100). This helps in preventing overfitting. I received mediocre score results on my test set. **What I quickly realized was that it was best to round my predictions to the nearest round number**, since the num_helpful results will also be a round number. This helped my scores a lot!

At this point I was thinking more about what could improve my model. Specifically what additional features could I create that would be useful? Since I was not actually using the review+summary text at all as part my model yet, I thought it would be best to try and do some NLP. First I did not some basic preprocessing of removing any weird formatting on the review+summary text, as well as lower casing everything . I then used sklearn's TFIDFVectorizer to transform the text into inverse document frequency counts . As parameters included into this transformer, I included a PorterStemmer which transformed the tokenized words into their base representation, and then removed stopwords (using NLTK python package). After some different experimentation I decided I would first extract the to top 50 unigram word bases, as well as the top 50 bigrams. Looking at the results the unigrams, and bigrams all looked to be like words, phrases that could be important to include in my model. So I decided to take the log of the counts of these top 100 unigram / bigrams as 100 new featues for my model.

This helped slightly improve my results for xgbRegressor() locally as well as the public scoreboard!

Finally, I decided to try and use percent_helpful instead of predicting num_helpful, as it made a lot more sense intuitively and I wanted to see how the results would be affected. It gave similar results, so overall I decided to use the one of each model.

Here is a snapshot of some of my top importance features from xgboost.



Building a data transformation pipeline

Trying different variations of added features with my model started to make my work disorganized, with jupyter notebooks scattered everywhere with duplicate code. Then once I wanted to apply my new models to the test data including all the different feature extraction, data cleaning and finally spitting out final results to a .txt file, this was getting very time consuming! So I decided to a create a python helper files with functions that were generic enough where I could pass in different models and it would do all the data transformation and test result file creation for me. This helped me immensely in time saved in the long run, so I could focus more on trying different modeling variations and techniques, and less on duplicate data cleaning work, etc..

My final model(s)

Overall I submitted 2 models, once more conservative and one more aggressive.

For my conservative model, I ended up using XgbRegressor() with max_depth 5 and n_estimators 150 and all other parameters default as it gave best local test score. Features used were outlined in this report including punctuation counts, review/summary lengths, top 100 unigram/bigram counts. I used percent_helpful as the predictor. And of course, always rounded my predictions to the nearest whole number

For my aggressive model, it gave a very high score on the public score, but could possibly be overfitting. What I did was an interesting approach I read about where I essentially took the model above, and took the prediction results and fed all the same features from the first model that into another XgbRegressor() model, but now included the first model predictions as a new feature. I did these then again, for the final model including all the initial features + model 1 predictions + model2 predictions as features. For each model I did a different random train / validation split. Overall I saw large improvements in local scores as well as public scoreboard scores, though I suspected I might be overfitting. I predicted num_helpful for this model.

For all my final models I used a local test set as well as GridSearchCV for hyper parameter tuning and cross-validation.

Lessons learned

- First off most importantly, I spent way too much time trying to slightly improve my public score, which was not beneficial! I should have just focused and trusted my local CV and test data scores.
- Building a general pipeline for data cleaning, feature extraction, and test data prediction was very beneficial and something I should have done earlier
- Spend more time on feature engineering and less on hyperparameter tuning!
- More complex is not always better. A simpler model is a lot easier to interpret, use, and tune. And the improvements seen in score by added complexity can be from overfitting

Conclusion / Final Scores

The more aggressive model which gave a MAE of .16042 public score gave my best private score of .16657. I was happy that my model was not overfitting too badly! My more conservative model increased from .167 to a little over .170.

In conclusion, I learned a ton from this kaggle competition and really enjoyed it!
I am excited to try some real kaggle competitions in the future!