Helpfulness Prediction

Rishabh Misra | r1misra@ucsd.edu

Helpfulness Prediction

- Leaderboard Standing: 52nd
- Team Name: r_misra
- Profile: kaggle.com/rmisra

MODEL AND FEATURE DESIGN

- **General approach**: I started with a very simple linear model with rating and length of the review text as features. Then, I did a systematic analysis of all the features that could possibly help in predicting the number of helpful votes. I used ablation to gauge the significance of features individually as well as in a group (do they have a synergic effect?). After identifying the appropriate features, I further analyzed their form in which they could be used to produce better results (like one-hot encoding, categorical representation, scalar representation, binary representation etc.) After identifying all the good features, I tested various regression models (ridge regression, gradient boosting, random forest, MLP regressor), though in the end, linear regression performed the best. I also tried PCA to select only the most informative features, but that did not improve the performance. For testing purposes, I considered various random settings of shuffling the data (using seed) and holding 15% of data for validation. I considered a particular approach good if only it improved the accuracy on all the sets, thus keeping me from overfitting on the public test data.

- **Specific details**: There were several eureka moments when I improved my previous solution by a large margin. First one was when I used only the training examples which had total votes (outOf) greater than a certain number (after tuning on validation sets, it came to be 16). The improvement made sense because the reviews with less number of total votes restrict the model from learning the true association between the label and the features as they have less variance. The second one was when I used the features related to the text of the reviews (bag-of-words representation and TFIDF vector). Though, I ended up not including them into my final solution as they didn't fit well with the model I used later son (and which was giving better performance). The third one was when I tried PolynomialFeatures preprocessing from scikit-learn. This allowed me to produce combinations of features (I used degree 2) efficiently. Having those as additional features removed the assumption that features are independently predicting the labels. Though, this eliminated the scope of using one-hot feature representations as producing features with them would not exactly capture the interactions (as 1 column of one-hot representation only conveys a fraction of information about that feature). Last moment was when I realized that instead of ignoring the training examples with less total votes, I could train a separate model on those examples. That way I won't ruin my first model by fitting it for examples having less total votes. Thus, I'd have best from both models to give the final prediction (first one predicting for less total votes and the second one predicting for more total votes).

- **Final solution's features** were 2-degree polynomial interactions of following attributes
  - outOf
  - number of words in review text after stemming and removing punctuation
  - rating
  - price -> if price of item is not in dataset, then take category's average

- month -> values 1-12
- category id

Note: 2-degree polynomial interaction (with interaction only and bias parameter true) of attributes [a, b, c] would produce [1, a, b, c, ab, ac, bc] features.

- **Final solution's model** is described following:
  - Linear Regressor 1: trained on examples with outOf > 16
  - Linear Regressor 2: trained on examples with outOf <= 16 and outOf > 5.

  I still ignored the examples with outOf <=5, as they had too little variance to learn a decent predictor. Though, I used the second regressor to predict labels for them.

RESULTS AND LEARNING

- For the final submissions, I chose one solution which had only one regressor and gave a score of 0.16071 on the public leaderboard, and the other which had two regressors and gave a score of 0.159 on the public leaderboard. Though I had better submissions on the public leaderboard, I chose not to select them as though they improved my score on public leaderboard, their performance was not consistent across multiple validation sets I was testing my models on. Now having the scores on whole data, I know that I made the right choice. My final score on the private leaderboard is 0.17114.
- The most important things I learned from this task are one, sometimes more data is less useful and two, being mindful of overfitting.