How Helpful is This Amazon Review?

Cristian Caraballo Richard Addo Hanna Kim

BACKGROUND

Reviews have completely transformed the experience of online shopping. In the past, there was much risk involved for the consumer because of uncertainties in the product's quality, durability, efficiency, and etc. Now, reviews serve as a tool to minimize the distance between the product and the consumer. In theory, having a public forum on these shopping websites where previous buyers can enter in text reviews and star-based ratings should solve the problem. However, such a method does not account for the fact that undetailed textual reviews do exist and do little to inform the decision making process of a potential buyer. There have been many modifications made on shopping websites to work towards this goal of providing a potential buyer with enough information to help decide whether or not to purchase an item.

Amazon.com, for instance, poses the question, "Was this review helpful to you? Yes/No" to a potential buyer reading a review, and automatically sorts reviews based on "helpfulness". However, not everyone reading a review responds to this question, and hence inundation of unhelpful reviews still persists. Through the application of machine learning algorithms, we hope to develop a tool that can automatically measure the helpfulness of any online review to assist the consumer with the weeding out of unhelpful reviews and to promote the ones which provide substantial information.

MILESTONE GOAL UPDATE

In our project proposal, we mentioned that we expected to have preliminary implementations of Naive Bayes (baseline), Maximum Entropy, and Random Forests.

However, we decided to use Logistic Regression for the baseline instead. This is because we have reformulated the problem as a regression problem instead of a classification problem. We had originally wanted to measure the helpfulness of a review under 4 discrete categories, but upon discussions within the group and with advice from Professor Torresani, we realized a regression formulation would be a much better solution for our

problem. So now, instead of predicting the class of a review, we predict the actual helpfulness score for the review.

Additionally, we decided to implement only Random Forests as our main method, because after more literature review, we believe that with more time and focus on the Random Forest method, we would be able to fine tune our Random Forests implementation and achieve very good results, as opposed to focusing on improving the performance of both Maximum Entropy and Random Forests within the limited timeframe. However, should we obtain very satisfactory performance with our Random Forests implementation before the final project deadline, we would implement the Maximum Entropy model solely for comparison purposes.

As far as reaching milestone goals are concerned, although we have yet to implement our baseline method, we believe we are well on track as we have already achieved very good results with our implementation of Random Forests.

DATA

We successfully obtained links to the entire Amazon reviews dataset compiled by the Stanford Network Analysis Platform (SNAP) Group as part of the Stanford Large Network Dataset Collection. The dataset consists of almost 35 million consumer product reviews from Amazon.com, spanning 18 years. Almost 2.5 million products are reviewed by over 6.5 million users, and over 50,000 of these users have reviewed more than 50 products.

The original dataset comprises reviews for 28 different consumer product categories. However, all training and testing so far has been performed on a subset of reviews for the "Electronics" category.

Each review in the dataset comes with the user's name and ID, the user's rating of the product, the total number of votes on the review with the corresponding number of "helpful" votes, and other information in addition to the actual review text. Additionally, the dataset provides a separate file containing descriptions of all the products. Figure 1 is a snapshot of the format of each review in the dataset [1, 2].

```
product/productId: B00006HAXW
product/title: Rock Rhythm & Doo Wop: Greatest Early Rock
product/price: unknown
review/userId: A1RSDE90N6RSZF
review/profileName: Joseph M. Kotow
review/helpfulness: 9/9
review/score: 5.0
review/time: 1042502400
review/summary: Pittsburgh - Home of the OLDIES
review/text: I have all of the doo wop DVD's and this one is as good or better than the
1st ones. Remember once these performers are gone, we'll never get to see them again.
Rhino did an excellent job and if you like or love doo wop and Rock n Roll you'll LOVE
this DVD !!
```

Figure 1: Snapshot of each review in the dataset.

FEATURE EXTRACTION

So far, we have experimented with a word presence representation of the words in the review and the user's rating of the product as the features.

First, we set a filter on the kinds of reviews in the dataset that should be considered. We have been only considering reviews that have at least 15 words in the text and at least 10 total votes.

Out of the selected reviews, we build a vocabulary out of the subset randomly selected as the training set. To achieve this, we first iterate through the review text of each review in the training set and apply Porter Stemming on all words. We then use this set of stemmed words to create a mapping of words to their frequencies, and select the *n* most frequent words that would become the vocabulary.

Next, we extract features for each of the training and testing examples. For each review, we apply the same Porter Stemmer on each word in the review text. We then check for the presence of each stemmed word in the vocabulary in this stemmed review text and then add the product rating (discrete values from 1-5) to create a feature vector for the example.

METHODS: RANDOM FOREST IMPLEMENTATION

We grow each tree using CART methodology, and we do not prune any of the trees. Additionally, our Random Forest implementation employs two layers of randomness:

- **Bootstrapping**: We randomly sample with replacement a subset of examples to build each tree in the forest.

- **Random features***:* To split a node, we evaluate a random selection of features with different thresholds to identify the best feature to split on. So far, we have tested with $m_{try} = \frac{numFeatures}{3}$, where m_{try} represents the size of the randomized subset of features being considered for splitting.

We employ these two layers of randomness because they have been shown to produce high levels of accuracy [3].

Termination Criteria

In our current implementation, the only termination criterion is the size of the node. In our code, we set a variable `minSplittableNodeSize`, and during the building of a tree, any node that has fewer than `minSplittableNodeSize` examples is not split any further. So far, we have only tested with `minSplittableNodeSize` *= 5*.

Prediction

We take the mean over values predicted by the regression trees in the forest. This final number is the predicted helpfulness score for the given review.

RESULTS

The accuracy of a random forest model is expected to increase with increasing number of trees in the forest. Since our current implementation runs very slowly, we first measured the performance of our implementation with different number of trees before proceeding to perform cross-validation with other parameters.

Figure 2 shows the mean squared error (MSE) obtained when using a training set of 4,000 examples and a test set of 1,000 examples. From Figure 2, it appears that the MSE does not

reduce significantly for the small range of numbers of trees we tested (1 - 21). However, we observed that the plots of the cross validation error and test errors assume the same shape. Additionally, the errors for both cross-validation and testing were very close to each other. The maximum difference between the error values is about 0.0055, when the forest is composed of 11 trees.

Overall, the preliminary implementation of the algorithm appears to be performing quite well, as observed from the ~7% error in both cross-validation and testing (Figure 2).

Table 1 is a snapshot of the actual values and the predicted values when the forest constructed from the 4,000 training examples was applied on the 1,000 test examples. Noticeably, the algorithm performed very well with examples such as example 9, where it predicted a helpfulness rating of 0.9266 instead of 0.9355. For other examples, such as example 3 where the algorithm predicted 0.3704 instead of 0.5966, the algorithm did not perform as well.

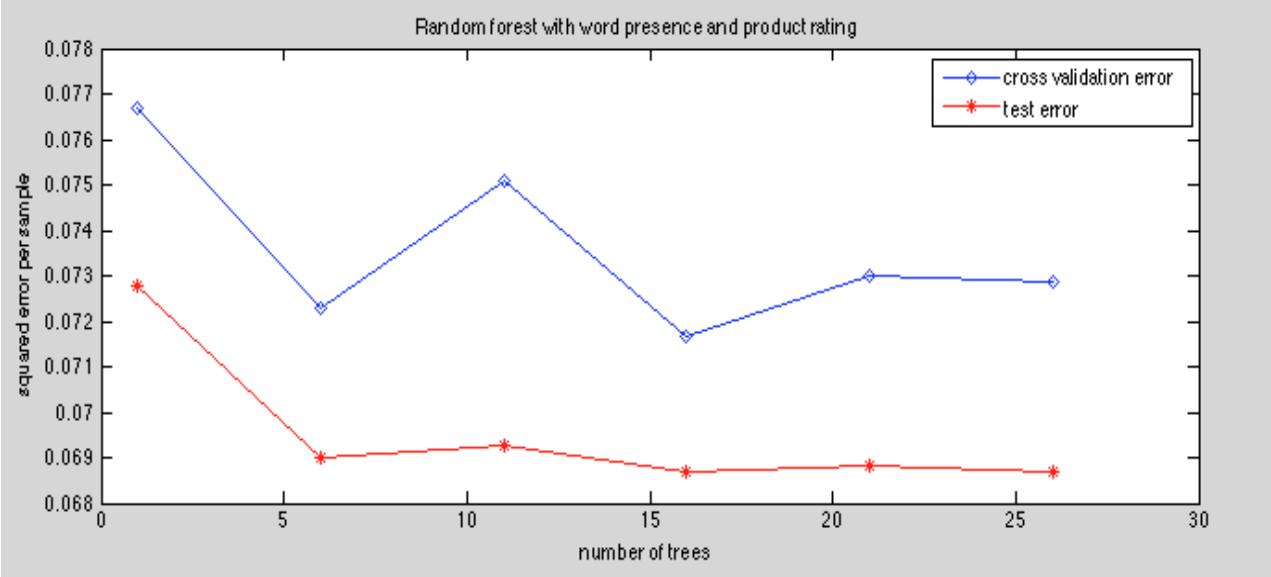
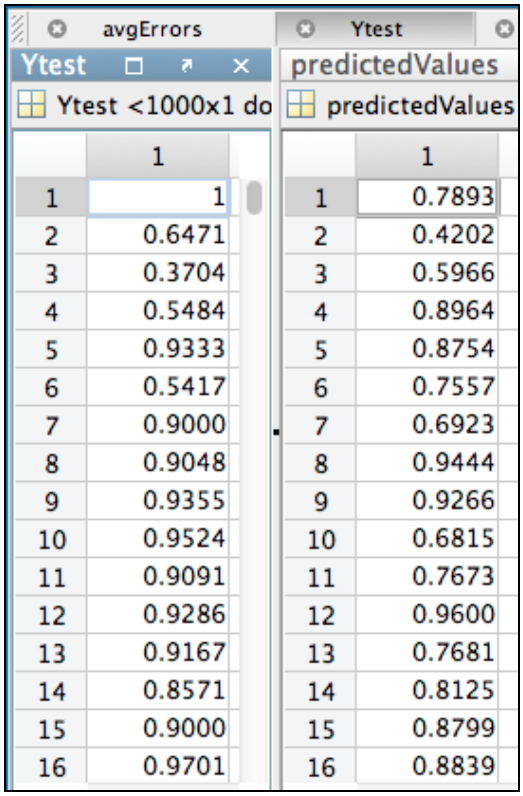


Figure 2: Mean squared error for different number of trees in the forest.



	1
1	1
2	0.6471
3	0.3704
4	0.5484
5	0.9333
6	0.5417
7	0.9000
8	0.9048
9	0.9355
10	0.9524
11	0.9091
12	0.9286
13	0.9167
14	0.8571
15	0.9000
16	0.9701

	1
1	0.7893
2	0.4202
3	0.5966
4	0.8964
5	0.8754
6	0.7557
7	0.6923
8	0.9444
9	0.9266
10	0.6815
11	0.7673
12	0.9600
13	0.7681
14	0.8125
15	0.8799
16	0.8839

Table 1. Comparison of the actual values and the predicted values with a training set comprising 1000 examples.

NEXT STEPS

- Implement logistic regression baseline
- Fine-tune our Random Forest implementation
  - ❏ So far, we have only experimented with the review text and product rating as features. We intend to explore the incorporation of additional features such as: *review length, number of reviews by same user, product cost, length of product description,* and *words in review summary*.
  - ❏ Evaluate performance of the implementation with a bag of words representation.
  - ❏ Evaluate performance of the implementation with different values of other parameters such as `minSplittableNodeSize` and m_{try} and different termination criteria.

- Parallelize forest construction:
  A key to exploring different features and parameters would be optimizing the running time of our implementation so that we will be able to perform several tests and cross-validations within the limited time. The current implementation took over 15 hours for the cross validation and testing in Figure 2 and Table 1, and we hope to improve this by parallelizing the construction of the trees.
- If time allows, implement Maximum Entropy and compare its results with those from our implementations of Logistic Regression and Random Forests.
- Run SciKit's Random Forest implementation on the same selection of examples. It would be interesting to see how our implementation compares to the library-provided implementation as an additional point of reference.

REFERENCES

---

[1] http://snap.stanford.edu/data/web-Amazon.html.

[2] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. ACM Recommender Systems conference (RecSys), 2013.

[3]. Andy Liaw and Matthew Wiener. Classification and Regression by randomForest. *R News*, Vol. 2, No. 3. (2002), pp. 18-22.