Recommending Products without Purchase History

Ran Tao A11557029
Richard Tran A09774761
Harvey Yu A98118071

Goal

Amazon online shopping services suggests many products to the user in order to promote sales by helping the user find other items relevant to the user. These suggestions appear in a few forms: "Frequently Bought Together", "Customers Who Bought This Item Also Bought", and "What Other Items Do Customers Buy After Viewing This Item?". We chose the predictive task of simulating "Also Bought" predictions.

Applications of our predictions could be possible improvements to Amazon's also bought list, or a model used to help predict purchases based on other items. This can be used for advertising purposes especially where if a customer buys an item, we know which item they will most likely buy next or be thinking of buying.

Dataset

We chose to make predictions on Amazon product data[1], specifically for the Video Games category.

Overview

The data is a good fit for our task since the number of items are relatively low, but large enough to meet the requirements. There are 74,342 products, and 50,210 products have reviews, with 1,327,825 total reviews. The relatively low number of items allows for faster iterations of experiments, given the restriction of the low-budget consumer-level hardware available to us.

This data provides the following information for each review:

`reviewerID` - ID of the reviewer, e.g. A4QYHKWW8ICPC
`asin` - ID of the product, e.g. B00EFFW0HC
`reviewerName` - name of the reviewer
`helpful` - helpfulness rating of the review, e.g. 2/3
`reviewText` - text of the review
`overall` - rating of the product
`summary` - summary of the review
`unixReviewTime` - time of the review (unix time)
`reviewTime` - time of the review (raw)

The data also provides metadata for each item:

`asin` - ID of the product, e.g. B00EFFW0HC
`title` - name of the product
`price` - price in US dollars (at time of crawl)

[1] http://jmcauley.ucsd.edu/data/amazon/

imUrl - URL of the image associated with the product.
related - related products

- **also_bought** - List of product IDs that appear in “Also Bought” recommendations.
- **also_viewed** - List of product IDs that appear in “Also Viewed” recommendations
- **bought_together** - List of product IDs that appear in “Frequently Bought Together”.
- **buy_after_viewing** - List of product IDs that appear in “Buy After Viewing”

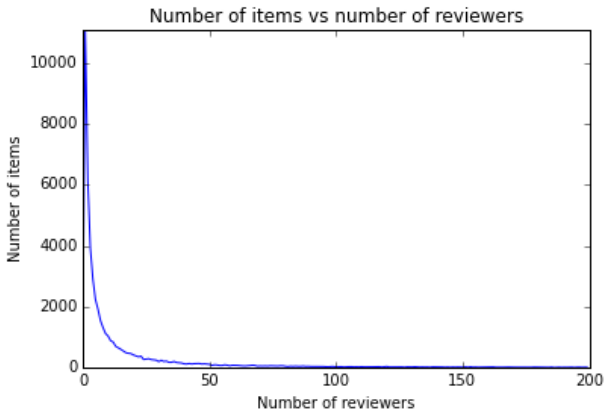
salesRank - Ranking of how well the product sells.
brand - brand name
categories - List of categories of the product.

Interesting Findings

- The dataset doesn’t include the actual name of the item for many of the items, but the asin field correlates with the exact item. It is possible (but time consuming) to find all of the names of the items. The name of the item could be used as text for prediction, since it is intuitive that “Call of Duty Black Ops II” is likely to be bought together with “Call of Duty Black Ops II: Vengeance DLC”.
- Not every item contains an “Also Bought” field. Some don’t even contain a “Related” field.
- Relations between items in the also bought list are not always two-way. Item X maybe also bought for item Y, but item Y might not be also bought for item X.
- Many of the items in the also bought are not in our data set. Meaning they show up in some item’s also bought list, but the item’s actual page is not included in the dataset. We may try automatically predicting yes to these, since they wouldn’t show up in the data unless they were being bought.
- Not all items in the metadata have a sales rank. It seems Amazon defers reporting sales ranks for items that are not popular or specifically past some age. This presents a problem since we are planning on using the sales rank in order to predict purchases, but we are opting to default missing sales ranks to a value of 99,999.
- The metadata contains a imUrl, containing a link to the image shown for the item. We may possibly be able to use this image’s feature (colors, shapes) to correlate it with a certain genre of games, and perhaps use it to help predict whether or not it will be used to be purchased with another item.

Exploratory Analysis

We noted that most items have very few reviews if any at all. In fact, 98% of items had less than 200 reviews. The following graph shows the number of items drops dramatically for items with more than 25 reviews. The graph below shows the distribution of the amount of items with a certain number of reviewers.



The lack of reviews for each item makes it difficult to correlate purchases since there is very sparse information to predict with. If we were given more reviews, or if every item had many reviews, it would be possible to reconstruct purchase history for each customer and therefore use that to help predict the also bought list.

Predictive Task

At first, we wanted our predictive task to be: “Given an item, what are the associated alsoBought items?” This relationship can be understood as a directed edge from an item A to all the alsoBought item B. However, with n items, this predictive task involves making n^2 predictions since we have to predict whether there exists an edge for each item A and for every other item B. The predictive task seemed too difficult for our limited computing resources, since there are around 50 thousand items and we would have to make 2.5 billion predictions.

In order to avoid this problem, we changed our predictive task to be: “Given two items, is there an edge (alsoBought relationship) between them?” This new predictive task allows us to directly apply many of the tools we learned in class.

Related Literature

We found an article by Jeff Sauro titled "Customers Who Bought This Item Also Bought... Affinity Analysis Explained" [2]. Amazon uses a proprietary algorithm to create their "Also Bought" category. Their State-of-the-art method involves using actual purchase information and whether or not the item was viewed or not which we don't have access to. Amazon creates matrices of each item containing features where they list whether the item is bought or not, assuming all of these items are viewed by the customer. The values for each item is a binary value representing whether or not the item was purchased or not. Amazon then lists items with the highest association first, and those with lower correlation values afterwards.

[2] http://www.measuringu.com/blog/affinity-analysis.php

Microsoft has an algorithm called the Microsoft Association Algorithm[3]. The algorithm uses more personalized results. For example, if it knows a customer is buying beef and cheese, it will make a correlation to a hamburger and then recommend other hamburger items such as lettuce and tomatoes to the customer. Each feature is given a weight, and after a minimum amount of features are satisfied, then a correlation is found with a certain probability based on the sum of the weights.

Some members at IEEE who worked at Amazon, Greg Linden, Brent Smith, and Jeremy York, published a paper in 2003 called "Amazon.com Recommendations: Item-to-Item Collaborative Filtering" [4] where they describe the recommendation algorithm they used for Amazon and how it compares to other algorithms. The item-to-item collaborative filtering algorithm they describe recommends items on how similar other items are to that item. The way they obtain the similarity between two items is by constructing a vector for each item that says which customers bought the item, and finding the cosine similarity (or some other similarity or some other vector similarity methods) of those two vectors. They find items with strong similarities, and displays those items under "Customers Who Bought This Item Also Bought". Compared to other e-commerce recommendation algorithms, this item-based collaborative filter works extremely well, and the online component runs in linear time, which is much more efficient than the other algorithms they provided.

Data Curation

This new predictive task required a new set of data. We wanted to create a dataset of 50% positively labeled data and 50% negatively labeled data, so we first created a set of all the item pairs that have an edge between them, regardless of the direction. Then we randomly generated a set of negatively labeled data by verifying none of those randomly generated pairs appear in the positively labeled data. It is important to note that some of the randomly generated negative label pairs could actually be a positive label, but since our dataset does not contain every item, it is possible that we will falsely label a pair as false, however it is actually true outside of our dataset. However the probability of such an incidence occurring is significantly low enough that it can be considered noise.

Finally data is split into three sets. Training, testing, and validation, where training consists of 60% of the total data, and testing and validation account for 20% each. When splitting the data, we did not validate that each of the training, testing and validation data consist of an even 50/50 split between positive and negative labels. Instead, we randomly shuffled the data before splitting the data.

Prediction Models

Definition of Accuracy

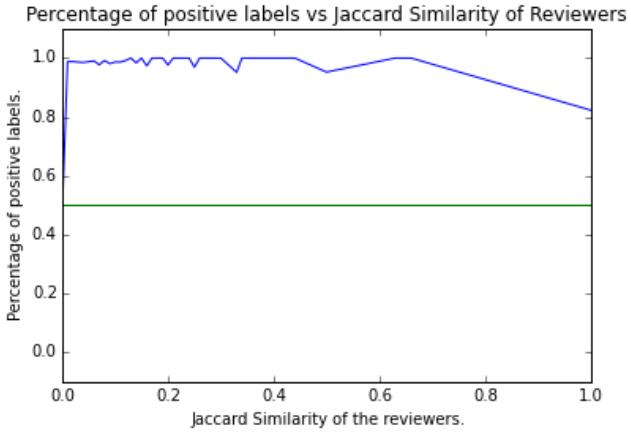
We will be defining accuracy in our models using the Classification Accuracy formula and error rate This uses the true positive, false positive, true negative, and false negative values of our predictions. $\frac{TP+TN}{TP+FP+TN+FN}$ All accuracies shown are values we obtained using our test set.

Baseline

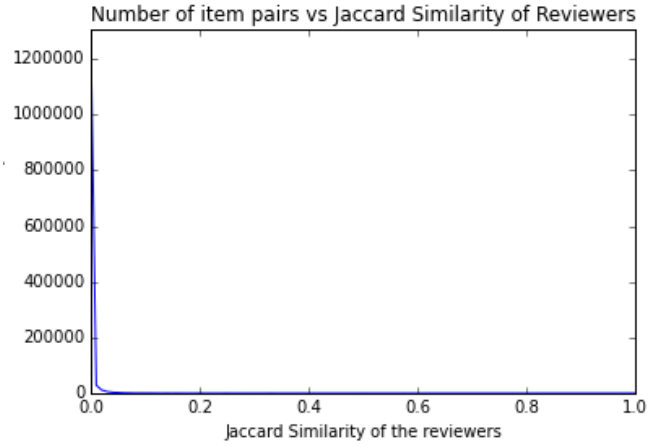
Our baseline model simply predicted that item B will appear in item A's alsoBought list if and only if both items shared a same reviewer. This means if a person reviewed both item A and item B, we would predict true. Our intuitive behind this approach is that in order for an item to appear in another item's "Customers Who Bought This Item Also Bought" related field, someone must have bought both items. To increase the confidence of our intuitive justification, we examined the relationship between

[3] https://msdn.microsoft.com/en-us/ms174916.aspx
[4] http://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf

the Jaccard similarity of the reviewers and the percentage of positive labels. The graph shows that it is highly likely for an item pair to be labeled true if the Jaccard score is non-zero, meaning that they share at least one reviewer.



The baseline gave us a predictive accuracy of 0.59349. This result went against our intuitive notion of the data as well as the empirical analysis. However after further exploration, we discovered the reason for the low accuracy was due to the fact that 0.90344 item pairs have a Jaccard score of 0, in which the percentage of positive labels is 0.40482. Therefore we are predicting a false negative on at least 0.36573 of the data.



Attempts

We tried using various features such as using the average rating of each item being over a threshold. Unfortunately this gave us worse-than-baseline results. We believe this is the case especially for items with a single reviewer giving it 5-stars, doesn't necessarily mean others will buy it. The fact that we don't take into account how many people review the item may be at fault here as well. Here are other approaches we tried:

Sales Ranking Threshold Predictor

We tried using the sales rank to find whether item B will be in the also bought list for item A. Our rationale behind this was because if both items are popular, then there would be a pretty good chance that they will both be bought. We initially didn't believe it would be such a great correlation, because someone buying a game for the Wii probably won't buy something for an XBox just because

they're both in the top 100 sales. However, we looped through multiple iterations of different sales rank thresholds, where we would predict true if both items' sales ranks were above said threshold. We were surprised to find that when using sales rank alone, 31111 was the optimal sales rank, giving us a classification accuracy of 0.81129 and an error rate of 0.18871. Our true/false positive/negative values were as such
True Positive: 129657
True Negative: 681400
False Positive: 1975
False Negative: 553537

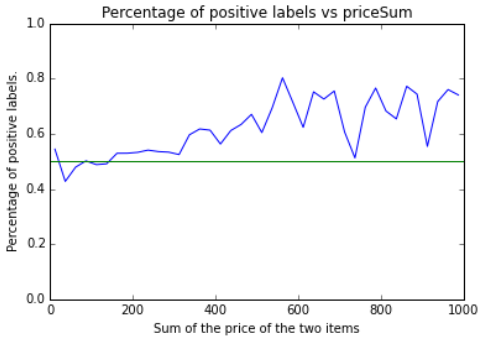
Jaccard Similarity of Categories Threshold Predictor

After doing some research, we discovered that Amazon provides recommendations using related items. We thought a good way to determine if two items were related was by looking at the categories. We found the Jaccard Similarity between the different categories of the items to help predict whether or not the items will be bought. This approach avoids the problem in the Sales Ranking Predictor where different consoles or categories of items were ignored. In this method, we predicted that if the Jaccard Similarity between the two items' categories was above some threshold, predict 1. Otherwise, predict 0. We found that a threshold of 0.38 is best for predicting using Jaccard Similarities. Alone, the Jaccard predictor gave us an accuracy of 81.8%.

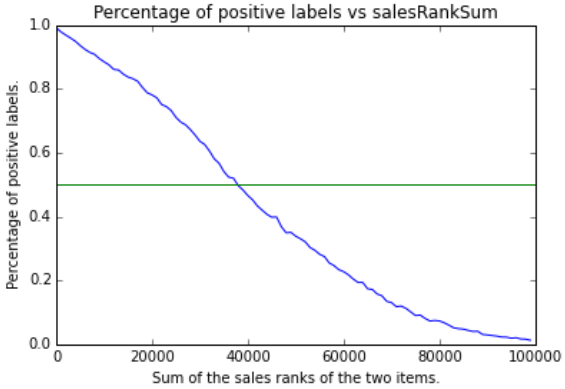
Decision Trees

In order to iterate on the previous attempts, we wanted to programmatically determine the threshold, therefore we chose to use the decision tree as our classifier. We ran the predictor with each of the chosen features independently to compare the relative effectiveness of each feature. The decision tree we're using is Scipy's DecisionTreeClassifier. We chose this library mainly for convenience, but it doesn't include pruning, thus in order to reduce overfitting we set the max depth of the tree to different values and ran the classifier on the validation set, picking a depth value that minimizes validation error.

`priceSumFeature, priceDiffFeature:` We want to use the average of the price of the items as a feature, so we chose to sum the prices and skip the step of dividing that sum by 2. Dividing it by 2 will take extra computation and only scale the feature without actually producing any additional information. Intuitively, the average of the price of the items helps section the data into "price ranges" that may be modelled differently. We also used the difference in the prices as a feature. Intuitively, if price of the two items are close, they are more likely to be bought together. The accuracy of these features together is 0.75587 with a max depth of 40. The following graphs show that there is a slight relationship between the sum of the prices and the positively labelled data, but little if any noticeable relationship using the difference in the prices.

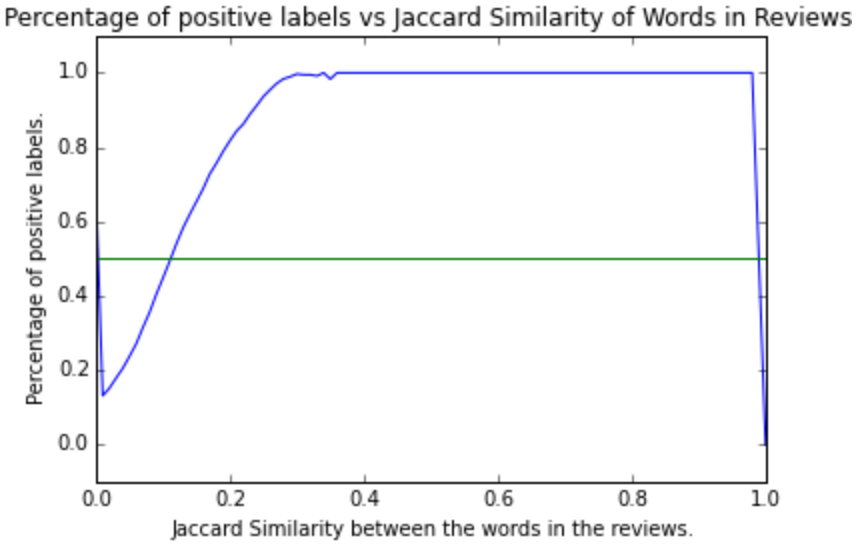


`salesRankSumFeature:` Since predicting using a sales rank threshold performed well when we tuned it by hand, we also tried using it in the decision tree. Instead of using the max of the two sales ranks as the feature, we found better results using the average of the item ranks (which is essentially the sum). The decision tree with a max depth of 1 produced an accuracy of 0.82643. As we expected, the threshold that the decision tree used 31023 was close to our manually tuned threshold of 31111. The following graph shows that there is a strong correlation between label and the sales rank sum, such that larger sales rank sums are more likely to be negatively labelled.

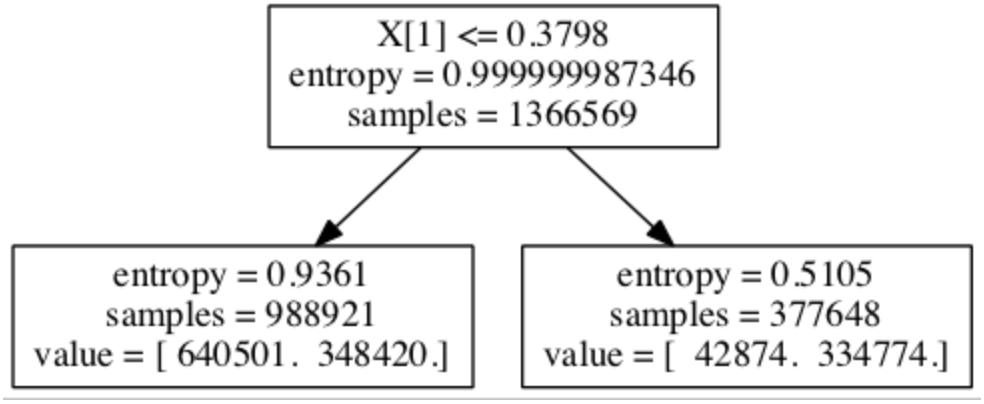


`jscoreOfReviewersFeature:` We also used the baseline as one of the features for the decision tree by finding the Jaccard similarity between the set of reviewers for each item in the pair. The decision tree classified with an accuracy of 0.69931.

`jscoreOfWordsFeature:` We also found the Jaccard similarity between words in reviews to help our predictor. We took all the words in all the reviews for each item and created a bag of words for that item. We filtered the bag of words to remove the stop words and then stemmed all the words. By plotting the percentage of positive labels against the Jaccard similarity, we noticed a linear correlation. The decision tree classified with an accuracy of 0.78264 at a max depth of 2.



`jscoreOfCategoriesFeature:` With the success of using Jaccard similarity on the categories and classifying based on a hand-tuned threshold, we also used the similarity of the categories as a feature in the decision tree. The tree of max depth 1 has a root node decision boundary of 0.378, confirming the accuracy of our hand-tuned threshold of 0.38. The decision tree classified with an accuracy of 0.81261 which is slightly better than the manual classification.



Best Predictor

The predictor that produced the highest accuracy used a decision tree that combined the best features. The decision tree predicted with an accuracy of 0.91438. We used the following features:

- **`jscoreOfCategoriesFeature`**
- **`jscoreOfWordsFeature`**
- **`priceSumFeature`**
- **`priceDiffRelativeFeature`**
- **`salesRankSumFeature`**
- **`jscoreOfReviewersFeature`**

The addition of each of these features improved our decision tree's classification accuracy so we included them. We tweaked the priceDiffFeature so that it was relative to the total price, by dividing the priceDiffFeature by the sum of the prices. However, this tweak only improved the decision tree by a tenth of a percent in accuracy.

Results

What worked well

In our final model, the features we used were the Jaccard similarities of categories, words in each review, the rating score, the sum of prices, relative price differences, and sales rank sum. Using a decision tree worked better than other algorithms because it combines other working algorithms and finds optimal values to where to split. Though for each of these individual features, a predictor based solely on it works quite well (classification algorithm above 70%), combining them creates a predictor that is substantially better (classification error >90%).

What didn't work well

Difference between sales rank did not work well because closeness in sales rank does not mean that the items are related. Therefore there is no correlation between item relatedness and closeness in sales rank.

Using price difference as a feature for a predictive model didn't work so well. We infer that this is because a relatively large difference between two small numbers can be the same as a relatively small difference between two large numbers. For example, if item A costs $10 and item B costs $60, this should be different than if item A costs $5000 and item B costs $5050. In the first pair, the "large" difference can determine whether a person is more inclined to buy item A than item B. In the latter, the "small" difference doesn't really affect the buyer's decision. As a result, we used a relative price difference, which normalizes the data by computing the ratio between the price difference and the size of the difference.

$$\frac{|price(A) - price(B)|}{|price(A) + price(B)|}$$

Conclusions

We managed to predict with a high degree of accuracy the products that are "also bought" by using a decision tree classifier with features consisting of item and review data. Even without any purchasing history of all the users, our predictor still manages to accurately predict related purchases. However, it is important to note that this model wouldn't be possible without the original labels provided by Amazon's predictions, so it cannot be used to replace Amazon's models. Instead, it is possible to use our model on a competitor's website, provided they have comparable features to provide our predictor.