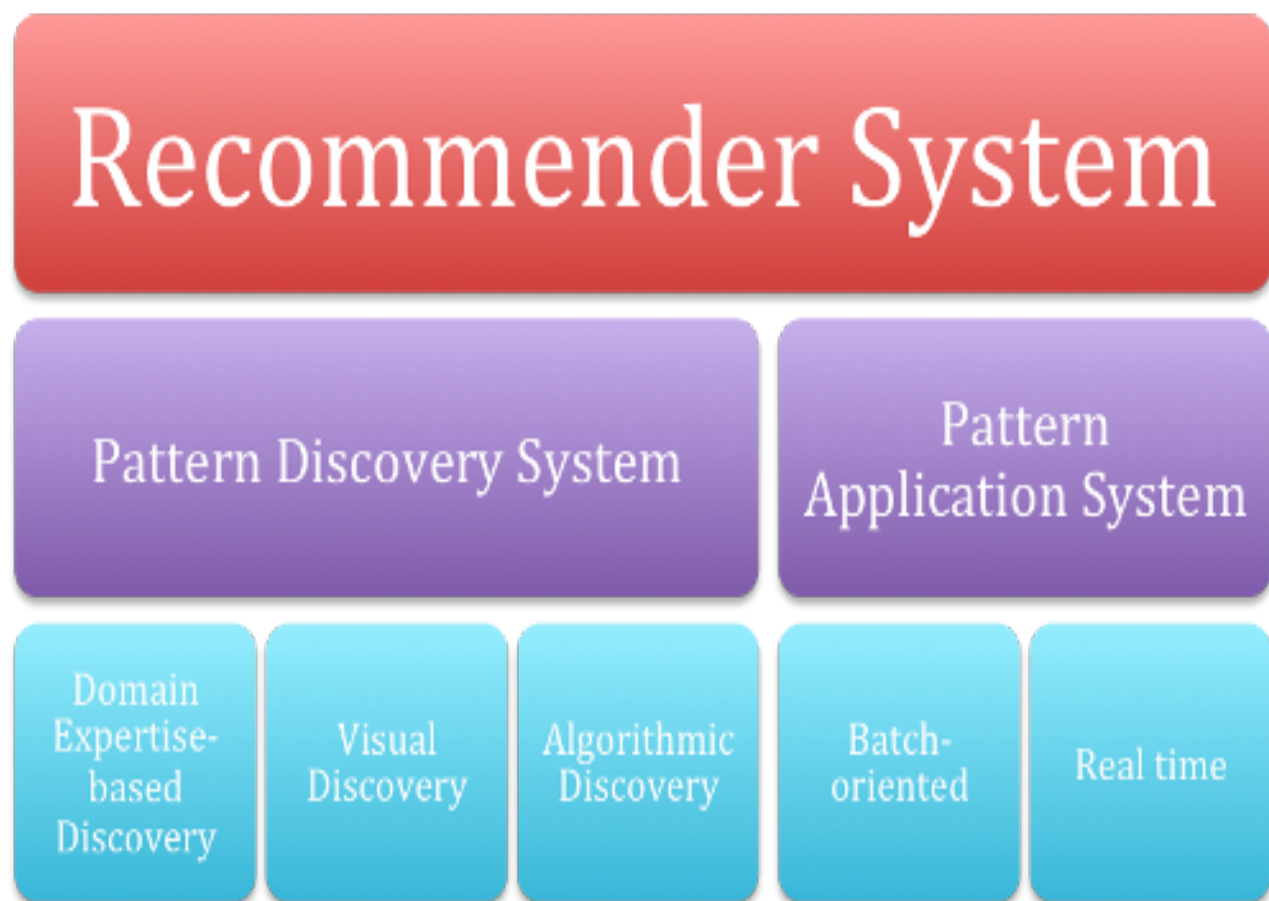


Recommender Systems

Chaitanya Devaguptapu

Project Overview

Recommender Systems are a subclass of information filtering system that seek to predict the "rating" or "preference" that a user would give to an item .Recommender systems have become extremely common in recent years, and are utilized in a variety of areas, some popular applications include movies, music, news, books, research articles, search queries, social tags, and products in general .



Under this project we are aiming to build a recommender system to make predictions based on the reviews of books on amazon . The dataset used for building this RS consists of 1 million Amazon Book reviews and the respective purchase information . Main goal of this project is to predict whether user will purchase the item based on this reviews .

Dataset

Dataset we have used for building this RS can be downloaded from [here](#) . The dataset (train.json.gz) consists of 1,000,000 reviews , we will split the dataset to suffice the training and testing to make the RS more dynamic on unseen data. The fields in the data file are :

- **itemID** : The ID of the item. This is a hashed product identifier from Amazon.
- **reviewerID** : The ID of the reviewer. This is a hashed user identifier from Amazon.
- **helpful** : Helpfulness votes for the review. This has two subfields, 'nHelpful' and 'outOf'. The latter is the total number of votes this review received, the former is the number of those that considered the review to be helpful.
- **reviewText** : The text of the review.
- **summary** : Summary of the review.
- **unixReviewTime** : Time of the review in seconds since 1970.
- **reviewTime** : Plain-text representation of the review time.
- **category** : Category labels of the product being reviewed.
- **pairs_Purchase.txt** : Pairs on which you are to predict whether a user purchased an item or not.

The test set has been constructed such that exactly 50% of the pairs correspond to purchased items and the other 50% do not.

Problem Statement and Approach

Given a (user,item) pair from ' pairs_Purchase.txt ' our task is to predict whether the user purchased that item (really, whether it was one of the items they reviewed)

To addresses this problem we have considered three approaches .

1. Baseline prediction was to find most popular products that account for 50% of purchases in the training data , return 1 whenever such a product is seen at test time ,0 otherwise.
2. The second approach for this problem was Logistic Regression here we found two ways to generate features and response in training and validation set .
3. As the final approach we have built a user based collaborative filtering system which is implemented by Jaccard similarity between users . Besides , we redefined the percentage threshold that works better than simple 50% threshold .

Metrics

Test data is in pairs like the one shown below :

userID-itemID,	prediction
----------------	------------

A30UP2KKD5IQEP-B00ERNO9E4	
---------------------------	--

A4V35W9XNM4X4-0394800796	
--------------------------	--

ACJORQ1CEABE-1601831412	
-------------------------	--

U321264570-I114057426	
-----------------------	--

U746082164-I893042700	
-----------------------	--

Our task is to estimate the prediction column , 1 if user purchased the item ,0 otherwise

The evaluation metric that we are considering for this task is the categorization accuracy (fraction of instances labeled correctly) , which is the **1-Hamming Loss** .

Hamming loss measures the accuracy in multi label classification , it is given by :

$$\text{HammingLoss}(x_i, y_i) = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{\text{xor}(x_i, y_i)}{|L|},$$

where

$|D|$ is the number of samples

$|L|$ is the number of labels

y_i is the ground truth

x_i is the prediction.

Data Exploration

The review data ("train.json.gz") is read into the form of list in python . This list contains 1 million dictionary type objects each containing all the nine fields that are specified above in the dataset description , below is an example of one of them

```
{'category': [['Books']],  
 'helpful': {'nHelpful': 0, 'outOf': 0},  
 'itemID': 'I572782694',  
 'rating': 5.0,  
 'reviewText': 'favorite of the series...May not have been as steamy as some of the others...but the character depth, and believability were amazing. wanted to curl up with Devlin and make it all better(wink wink) amazing series...found Laura Kate when I stumbled onto Hearts in Darkness(one of my all time faves)...this is up there with my Kresley Cole and Gena Showalter favorites.',  
 'reviewTime': '05 3, 2014',  
 'reviewerID': 'U243261361',  
 'summary': 'Loved it',  
 'unixReviewTime': 1399075200}
```

Now let us look at some of the basic statistics of the data :

- Total number of reviews in the dataset : 1000000
- Total number of users (unique) : 35736
- Total number of items (unique) : 37801
- Average user Frequency : 27.98
- Average item frequency: 26.45

-
- Average Rating : 4.218
 - Average length of review text : 1151.41
 - Average helpful rate: 0.738

The dataset on a total has 1000,000 user item pairs , but there are only 35736 unique users and 37801 unique items . The average user frequency (number of reviews a user has contributed) is ~28 and the average item frequency (number of reviews for a particular book) is 26.4 . From this information we can infer that each user has many purchase records and each item has been reviewed by different users , so the item frequency and user frequency can be considered as important features when making the prediction .

The 'rating', 'reviewText' and 'summary' features seem to be useful features, because we can conclude how much each user's likes the item they purchased, but in the original dataset, we do not have 'rating' information of 'non-purchased' user-item pairs , so we decided to drop this feature .

As a result , we are only interested in the purchase information , so we generated user-item pairs from original dataset and stores the in a list as shown below .

```
[ 'U243261361 - I572782694' ,  
  'U947346309 - I172992448' ,  
  'U405075077 - I236549536' ,  
  'U931293227 - I707281702' ,  
  'U398632015 - I127335134' ,  
  'U282193926 - I970107598' ,  
  'U681436293 - I732947093' ,  
  'U913252976 - I273175352' ,  
  'U584647862 - I040055669' ,  
  'U887091398 - I563838330' ]
```

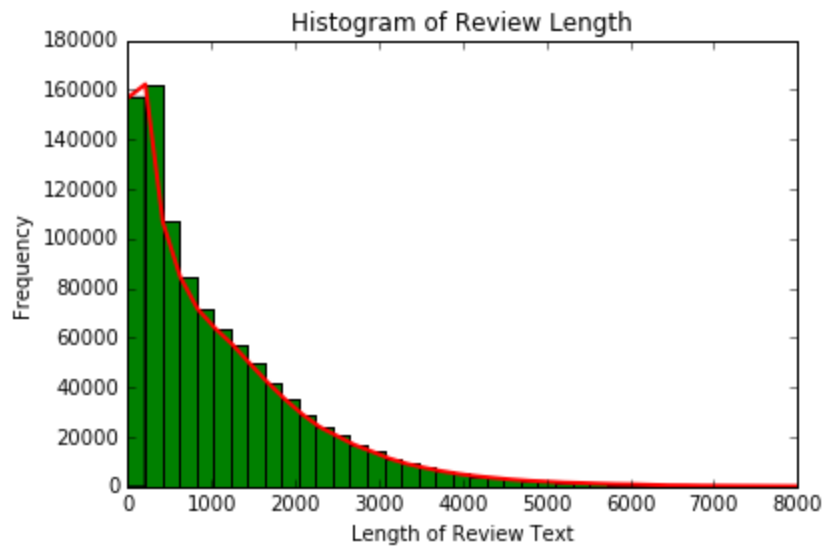
Since we only care about user-item pairs in this project, we do not need to be worried about abnormalities problem in any other features, in the reveiwerID and temID features, there aren't really any outliers and missing values.

We need to do some other reconstructions of training and validation set in the later steps according to specific learning algorithms and methods, we will talk about them in details later in this report.

Exploratory Visualization

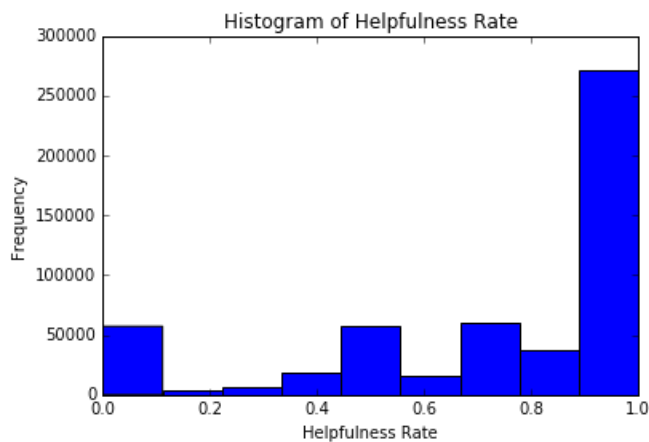
Histogram of Length of the Review :

Below shown is a plot for getting an intuition on length of the reviews



From the histogram of length of 'reviewText', we found most review content are not very long. Usually if the review are really long, the reviewers may complain about the items, so we can use this as another implicit evidence which shows that most users are satisfied with their purchase.

Histogram of Helpfulness Rate :



From the Helpfulness Rate histogram, it is very straightforward to show us that usually users reviews are considered to be very helpful and useful since the frequency of helpfulness rates between 0.9-1.0 are much higher than others. So users' feedback is a very useful resource to affect other people's feelings about items .

Algorithms and Techniques

As discussed earlier we will be using three methods for prediction :

1. Baseline popularity model: Baseline for this task simply ranks products by popularity, and returns '1' for popular products, by finding the most popular product that account for 50% of purchases . The Baseline model is an implementation of the benchmark code which can be seen in the downloaded data .
2. Logistic Regression: As discussed earlier our second implementation would be the LR .LR is good and common choice for classification problem, here we used the popularity which got from baseline model as a feature in a logistic regressor. The specific Logistic Regression code is from sklearn library
3. User-based Collaborative Filtering : This is a method that performs recommendation in terms of user similarity. Evaluated if items were purchased by users based on similarity measures between users. The items we may infer to be purchased by a user are drawn from those purchased by similar users. The collaborative filtering is a good idea for recommender system model, but we still have a few problems, since this method is actually kind of slow given a huge dataset . This method is implemented by Jaccard Similarity between users. Collaborative filtering using Jaccard Similarity between users is a standard implementation which defined as the intersection of two users divided by the union of them . Some resources that inspired the use of this method are mentioned in the References section .

Reason :

-
1. Based on the data exploration we have done earlier, we found popularity is a very important feature. In reality, usually the popular items are purchased in high probability, so we chose the baseline popularity model to implement this idea.
 2. There are lots of other classification machine learning algorithms, like SVMs. But we do not use SVMs here because they do not work well in very large data sets and the training time happens to be cubic in the size of the dataset.
 3. Collaborative Filtering works best when the user space is large and more or less insensitive to user size. In this project, we have a large user space and collaborative filtering is a good idea for recommender systems.

Benchmark

Benchmark accuracy is 0.638, because when we considered the baseline popularity method with the best threshold, the accuracy is 0.638, we are hoping to select some other models which perform better than the baseline.

Methodology

Data Preprocessing

Generating non-purchases data :

We need to be careful when generating a validation set for this data . We previously (discussed in Data Exploration) have have generated user-item pairs for based on all users and items which also include the non purchased items . Instead of just taking the last 100k reviews for validation, we instead took 50,000, but also randomly select 50,000 'non-purchases' by randomly selecting a user and an item, and checking that they do not already show up together in the training set. We have used the following chunk of code to generate the non-purchases data :

```
# generate non_purchased half
length = 0
while length < 50000:
    user = list(np.random.choice(unique_user, size = 70000, replace = True))
    item = list(np.random.choice(unique_item, size = 70000, replace = True))
    select = set(['{}-{}'.format(user[i], item[i]) for i in range(len(item))])
    select = select.difference(set(all_pairs[0:100000]))
    select = select.difference(set(all_pairs[950000:]))
    length = len(select)

non_purchase = list(np.random.choice(list(select), size = 50000, replace = False))
```

Now we got non-purchases dataset

```
['U501478638-I507471294',
 'U380506315-I820654026',
 'U780033890-I254842988',
 'U764574234-I898493465',
 'U558625745-I143437438',
 'U307898551-I092906927',
 'U978009713-I686717301',
 'U569185065-I897859165',
 'U148756148-I071552252',
 'U147006061-I936415489']
```

Split training and Validation set :

Since we don't have access to the test labels. We will need to simulate validation/test sets of our own. Now we split the training data as follows:

- Reviews 1-900,000 for training
- Reviews 950001 - 1,000,000 + 50,000 non-purchases for validation!

```
train_set = all_pairs[0: 900000]
val_set = all_pairs[950000: ] + non_purchase
```

In the Collaborative Filtering model, since it takes really a long time for prediction when testing set is large, so we used a smaller size validation set (500 Reviews + 500 non-purchases) . For Collaborative filtering the validation set is generated as follows :

```
val = all_pairs[999500: ] + non_purchase[0:500]
```

This may cause the accuracy of Collaborative Filtering model to be lower than using the same size of original validation set. We should consider this situation when comparing three models.

Build feature matrix for logistic regression:

In order to get the model feature matrix X for logistic regression, we will convert features represented as lists of dict object to matrix, each row of the matrix represents for each user-item pair. For example, if there are three pairs in dataset [item3 - user1, item1 - user2, item2 - user3] .

```
[{'user': 'user1', 'item': 'item3'},  
{ 'user': 'user2', 'item': 'item1'},  
{ 'user': 'user3', 'item': 'item2'}]
```

We want to build a matrix to represent all pairs in the above list as the feature matrix in logistic regression:

```
array([[ 0.,  0.,  1.,  1.,  0.,  0.],  
       [ 1.,  0.,  0.,  0.,  1.,  0.],  
       [ 0.,  1.,  0.,  0.,  0.,  1.]])
```

Each vector in a row of the matrix represents each pair, each row can be visualized as two small vectors, for example, the first row [0,0,1, 1,0,0] stands for the pair item3 - user1, here the first half of this vector [0,0,1] represents item3, and the second half [1,0,0] represents user 1, the combination of these two vectors is the row for pair item3 - user1.

Implementation

Baseline Popularity Model

First we generated a popularity list to record how many times each item was purchased and sorted items by decreasing popularity . Then we picked out the most popularity items that accounts for 50% of purchases.(The sum of purchases of these items is 50% of

the total purchases). To predict the pairs 'user-item' in test set, if the item showed up in the most popularity list, we predict this pair as '1'(purchase), otherwise, considered the pair as '0' (non-purchase).

Result: The accuracy of the baseline item popularity model on the validation set is 0.6366 (threshold = 0.5), the accuracy of the baseline user popularity model on the validation set is 0.65174 (threshold = 0.5).

Logistic Regression

We have used the scikit learn LogisticRegression with all the default parameters .

In order to implement logistic regression, our first step is to construct the response in training set. This is because the training set are all "purchase" , which means $y_{train} = [1,1,...,1]$, so we only have label 1 without any label 0. The idea to construct response is to use the popularity of items, if the pair in training is considered to be popular, then we label the pair as '1', otherwise as '0'. In other words, we generate the response by using baseline popularity model to predict on the training set . In the data preprocessing step, we have already got the model matrix X by using DictVectorizer to convert features list, now we got a one-zero matrix to represent the purchase information, each row in the matrix represents for each user-item pair.

The logistic Regression aims to model $p(\text{label} | \text{data})$ by training a classifier of the form:

$$y_i = \begin{cases} 1 & \text{if } X_i \cdot \theta > 0 \\ 0 & \text{otherwise} \end{cases}$$

Result : The accuracy of logistic regression with threshold = 0.5 is 0.65174.

Collaborative Filtering :

Collaborative filtering is to build a matrix which shows the relationship between users or items, then we can infer the preference of an user by checking the matrix and matching the user's information.

Similarity Measure : In this project, we measured the similarity by Jaccard Similarity, the Jaccard similarity between A and B is calculated by the following formula:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} :$$

To simplify the problem, we considered if two users purchased at least one common item, then recorded the similarity between two users as 1, otherwise is 0.

Utility Matrix : First we built a matrix, the row label represents each user and column label represents each item. The value in this matrix is either 1 or NaN, 1 represents the user purchased the item and the value is given for each user-item pair. The matrix is sparse, meaning that most of the entries are unknown. Here is an example of a matrix, describing three users: user1, user2, user3. The available items: item1, item2, item3 :

item	item1	item2	item3
user			
user1	1	NaN	1
user2	1	NaN	NaN
user3	NaN	1	1

Prediction Process : The goal of the recommender engine is to predict the 'NaN' in a matrix. For example, we want to predict user1-item2, we checked other users who purchased item2, here we found user3 purchased item2. The second step is to measure the Jaccard similarity between user1 and user3, since user1 and user3 have common purchase item-item3, so according to the definition of Jaccard similarity above, the similarity between user1 and user3 is 1. So we predict the value of user1-item2 as 1.

As another example of prediction, suppose we want to predict user2-item2, we checked other users who purchased item2 and we found user3. Then we measure the similarity between user2 and user3, they have no items in common, so the similarity of user2 and user3 is 0. As a result, we predict the value of user2-item as 0.

Here, because this model is time-consuming when test set is large, so we reduced the validation set to 1000.

Result : The accuracy of user-based collaborative filtering is 0.771 .

Metric : The accuracy is calculated by $(\text{true positive} + \text{true negative}) / \text{all predictions}$.

Complications :

- In this project, we need to think harder to generate validation set for this data. Instead of just split the original dataset. We took the last 50,000, but also randomly select 50,000 'non-purchases' by randomly selecting a user and an item, and checking that they don't already show up together in the training set.
- The computation time is large when implementing Collaborative Filtering, so we reduced the validation set size for this model to only 1000, this may cause the result not accurate.

Refinement

Instead of using a simple 50% threshold, we compared the performance of many different thresholds and select the best threshold .

We checked the performance of the following thresholds

```
array([ 0.1 ,  0.11,  0.12,  0.13,  0.14,  0.15,  0.16,  0.17,  0.18,
        0.19,  0.2 ,  0.21,  0.22,  0.23,  0.24,  0.25,  0.26,  0.27,
        0.28,  0.29,  0.3 ,  0.31,  0.32,  0.33,  0.34,  0.35,  0.36,
        0.37,  0.38,  0.39,  0.4 ,  0.41,  0.42,  0.43,  0.44,  0.45,
        0.46,  0.47,  0.48,  0.49,  0.5 ,  0.51,  0.52,  0.53,  0.54,
        0.55,  0.56,  0.57,  0.58,  0.59,  0.6 ,  0.61,  0.62,  0.63,
        0.64,  0.65,  0.66,  0.67,  0.68,  0.69,  0.7 ,  0.71,  0.72,
        0.73,  0.74,  0.75,  0.76,  0.77,  0.78,  0.79,  0.8 ,  0.81,
        0.82,  0.83,  0.84,  0.85,  0.86,  0.87,  0.88,  0.89,  0.9 ,
        0.91,  0.92,  0.93,  0.94,  0.95,  0.96,  0.97,  0.98,  0.99])
```

- The best accuracy of baseline item popularity model is 0.63865 with a threshold of 0.56
- The best accuracy of baseline user popularity model is 0.65683 with a threshold of 0.6
- The best accuracy of logistic regression model is 0.65683 with a threshold of 0.6.

Results

-
1. Baseline Popularity Model: The most efficiency model, very fast, but the performance is not good as my expectation. The user popularity model performs better than the item popularity model, the best performance of the user popularity model is 0.63357.
 2. Logistic Regression: Slower than the baseline popularity model, the performance is similar to the baseline popularity model.
 3. User-Based Collaborative Filtering: Performs best with really high accuracy, but when test set is large, this method is slower than the above two. The accuracy is 0.771 on the validation set.

Although the collaborative filtering is the slowest one, we still considered this model as our final model, since it's performance is really good compare to the baseline and logistic regression.

Compared to Benchmark :To demonstrate the result of our final model is significantly better than the benchmark, we considered running the code 20 times with different pools of randomly selected data and applied t-test to justify the hypothesis, we got accuracies :

0.774, 0.788, 0.775, 0.761, 0.795, 0.8, 0.793, 0.769, 0.766, 0.779, 0.794, 0.788, 0.795, 0.784, 0.8, 0.8, 0.769, 0.776, 0.782, 0.798 .

We got t statistic = 47.8 and p value = $2.856e-21$, p value is smaller than 0.05, so we can reject the hypothesis: mean of accuracy is equal to 0.65, which demonstrate that the result of collaborative filtering model is significantly better than the benchmark.

Justification

Finally we decided to use user-based collaborative filtering to do prediction, the result of the validation set is 0.771 which is higher than the accuracy we expected.

But the accuracy was got by just testing on 5000 validation set, which is not large enough, so the reliability of the accuracy should be doubted. But we still have reason to justify that the collaborative filtering works better than the baseline and logistic regression, this is because when generated the validation set in the first two methods, we got

non-purchase data set by generation instead of from real world, so the validation set is not as reliable as what we used in the last model.

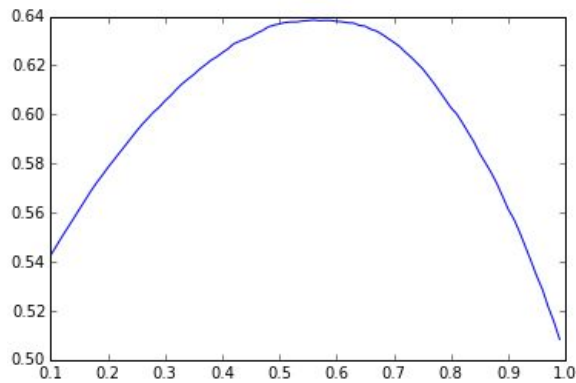
Conclusion

This project is a recommender system problem, the main idea is to evaluate people's preference and purchase habits and make some recommendations. In this specific problem, we are given 1 million Amazon book reviews, which provided us purchase information, our target is to predict when user purchased the item given some user-item pairs.

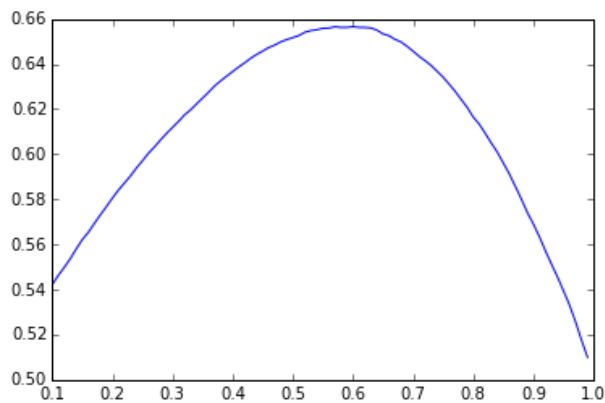
To solve the problem, first we generate a user-item list from the original list of review information. Then we reconstruct the training and validation data by randomly selecting user-item pairs which do not show up in the training set as non-purchase pairs. As for models, we considered three models: popularity model, logistic regression and collaborative filtering. Here is a table displaying average accuracy of different methods over 20 times:

	Average Accuracy
item popularity	0.63851
user popularity	0.65647
logistic regression	0.65647
collaborative filtering	0.78430

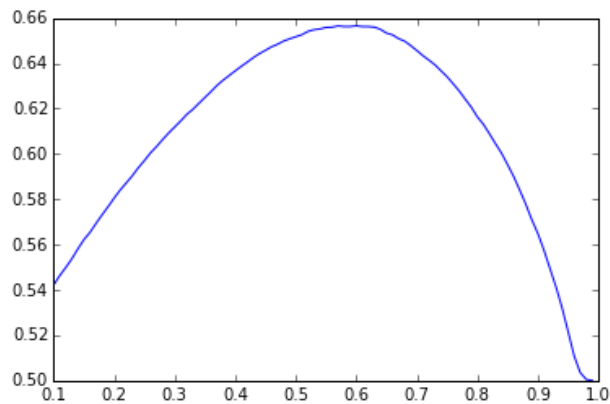
For popularity model, we considered the most popular items that accounts for 50% of purchases as label 1, and to refine the result, we compare the accuracy of different thresholds and found the best threshold is 0.56 :



Similarly, we considered the most 'popular' users that accounts for 50% of purchases as label 0, and compare the accuracy of different thresholds to refine the result the best threshold 0.6



For logistic regression, we combined the popularity in popularity model as a feature in logistic regression model, in other words, we use the prediction of popularity model on training set as the model response, and then transform the feature list to one-zeroes matrix to fit the model, also we select the best threshold by comparing the results of different thresholds, the best threshold is 0.6.



For collaborative filtering, we implemented this model by using Jaccard similarity to measure the similarity between users and evaluated users' preference.

We met some difficulties when we worked with the entire dataset for collaborative filtering model, working on the whole dataset is time-consuming, so we changed the validation set size to be smaller. This may resulted in a lower accuracy and undermine the comparison of three models.

As for interesting aspect of this project, recommender systems are increasingly popular application of Machine Learning in many industries. By evaluating the users and items popularities and users similarities, we can infer the users preference and purchase habits, which is a very practical and useful application in a variety of areas, we can use this idea in many other places such as music , movies and news recommendation, social tags or online dating.

Improvements

1. Collaborative filtering in practice is kind of slow given a huge enough dataset. In this project, we reduced the validation set size short the prediction time, but this solution may undermine the comparison of models because of the differences of validation set size. Considering some other models which also have good performance but faster than collaborative filtering may be a good idea.
2. In reality, the non-purchase pairs will be comparatively larger than purchased pairs, so the metrics of using accuracy may not be a good idea in practice, we

should consider assign additional weight to negative instances, for example, F₁ score

References :

- https://en.wikipedia.org/wiki/Hamming_distance
- http://scikit-learn.org/stable/modules/generated/sklearn.metrics.hamming_loss.html
- <http://cseweb.ucsd.edu/~jmcauley/cse190/files/assignment1.pdf>
- <https://www.coursera.org/learn/recommender-systems>
- <http://infolab.stanford.edu/~ullman/mmds/ch9.pdf>
- http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html
- <http://www.cs.ubc.ca/~laks/534l/collaborativeFiltering.pdf>