Jaya Krishna M
Mithun Phadnis
Mukhar Gupta
Sumedh Khandeparkar
Varun Vohra

Data Programming
December 8, 2016

Book Recommendation Engine

An engine which incorporates personalization to recommend books to the users based on their interests and previous choices.

Introduction

The more books there are to choose from, the harder it becomes to pick new authors and novels to try. That too much choice makes it harder to choose is a well-known phenomenon, so the sensible thing to do would be to restrict choice in some way thus making it easier to pick out books we want. One way to do that is simply to ask your friends to tell you which new books and authors they like, but their taste isn't necessarily yours. Another way is to ask a computer! The computer will recommend you books by using various algorithms, which is exactly what this recommendation engine does. The goal of this project is to help readers determine what they might like to read next based on their past reads. Users can rate book titles they have read in the past to determine what they might like next.

About Book-Crossing Dataset

The dataset has been collected in a four-week crawl (August / September 2004) from the Book-Crossing community. It contains the demographic information about 278,858 users (anonymized) who are distinctly identified by their UserID. The users have provided 1,149,780 ratings on 271,379 books.

The Book-Crossing dataset comprises of three tables :

BX-Users:

Contains information about the users. Note that the UserIDs have been anonymized and mapped to integers. The demographic information ('Location' , 'Age') has been provided if available. Otherwise, these fields contain *NULL* values.

BX-Books:

Contains information about the books. Books are identified by their respective International Standard Book Number (ISBN). Content based information is given such as the Book-Title, Book-Author, Year-Of-Publication, In case of several authors, only the first author is provided.

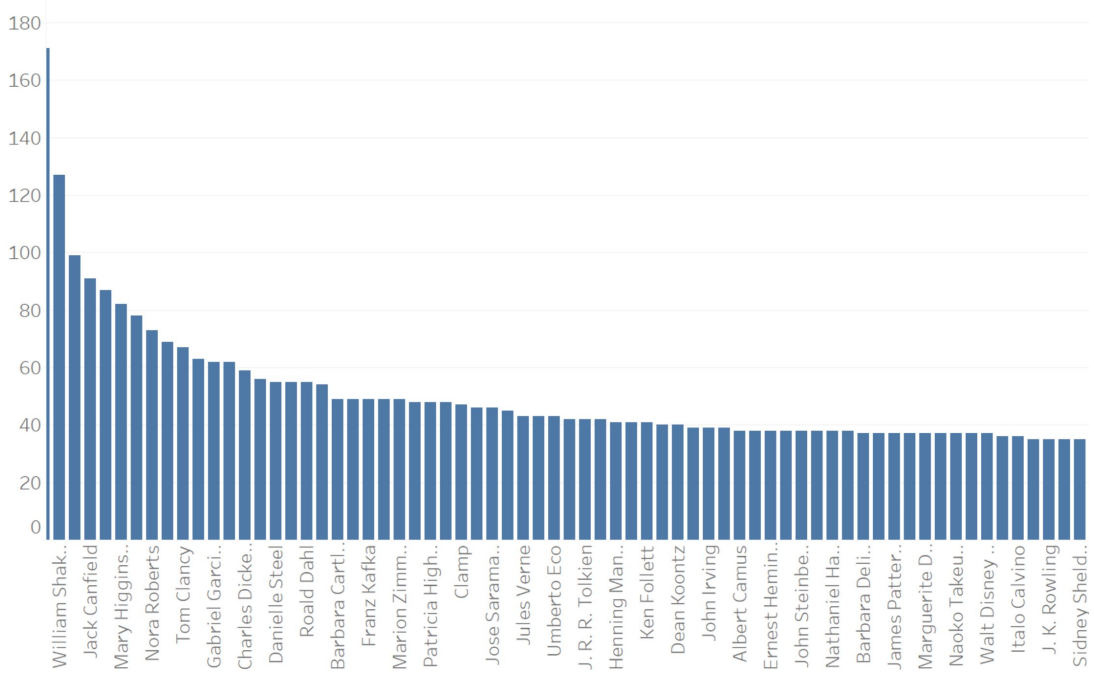
BX-Rating:

Contains information about the ratings of books. Ratings are expressed on a scale of 1-10 with higher values denoting higher appreciation. If the ratings have not been provided then this field contains a 0.

The dataset was downloaded in csv format. Most cleaning and structuring of the data to transform it into an appropriate form for the application of our models was done in MS Excel.

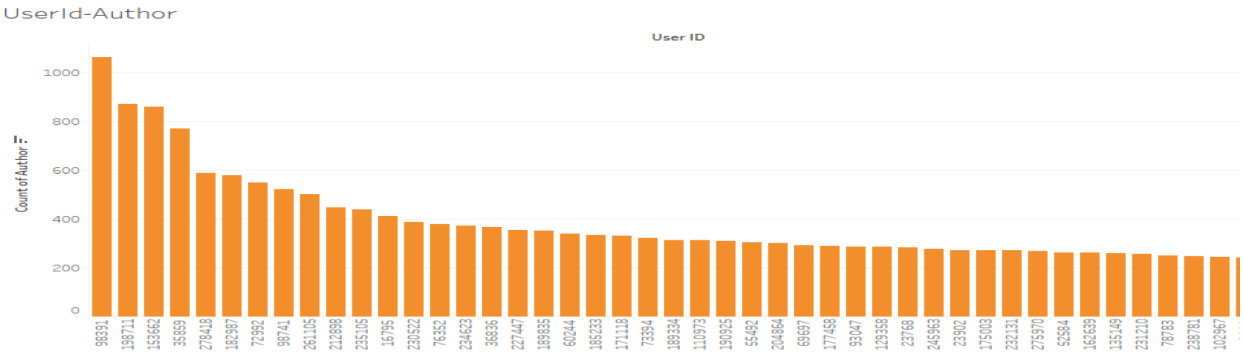
Some insights about the data:

- In the graph below we can see the number of books written by each author. These parameters are important for building our model.



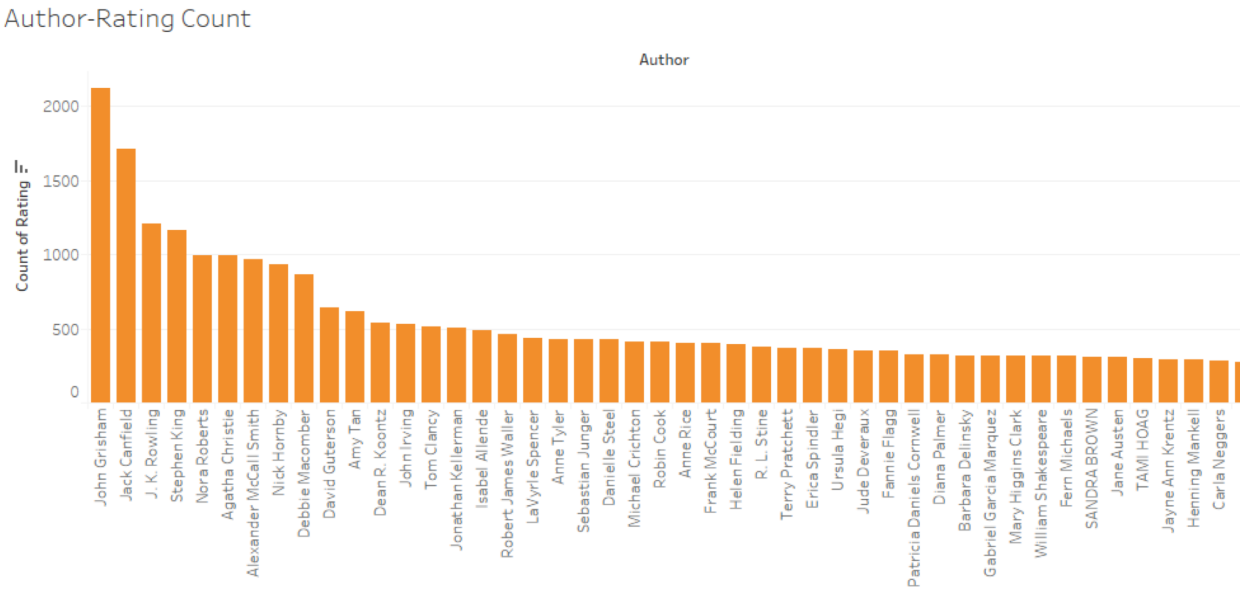
No. of Books written By Authors.

- We can see in the below graph that there is relation between users and authors they prefer. Hence, these two are also important attributes that need to considered while building the model.



User Author Preference

- In the below graph we plot author and the count of ratings for each of them. These two variables are also important while building the model for recommending the books.



Author - Rating Count

Data Cleaning:

The initial data was in highly unstructured format, few columns contained data which had special characters in location column as many as NULL values.

Main attributes in the Dataset:

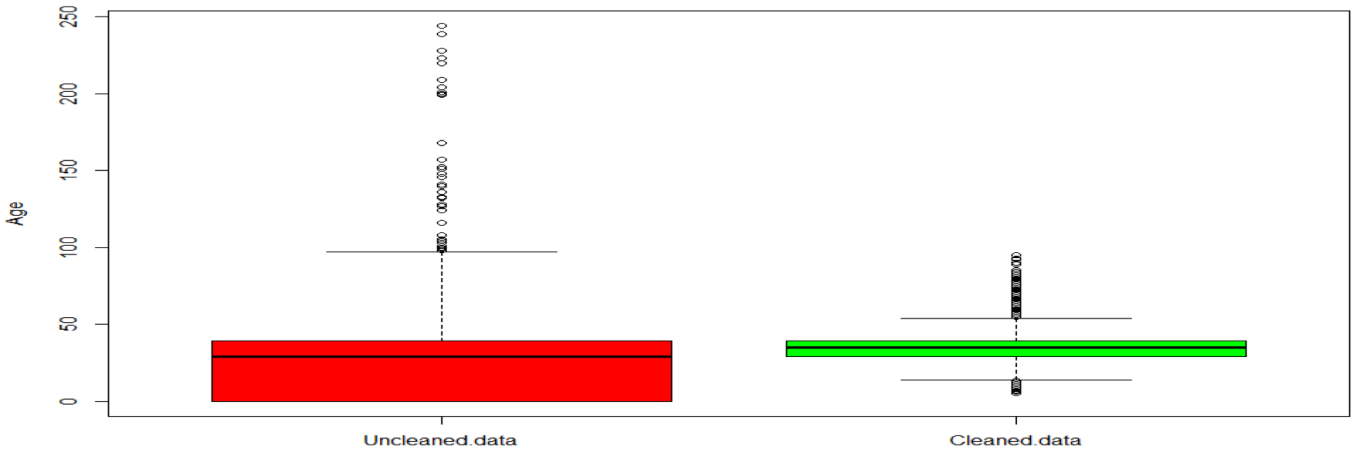
- Age: Age of all the users is given in the dataset.
- Author: Author of all the books is given in the dataset
- Rating: The user ratings given for books read is provided.

Process used for cleaning each column:

Age:

- The age column contained values ranging from 0 to 244. Practically there will be no user of age 244 who would be rating the book . We have also considered kids below the age of 5 are not reviewing and rating the books.
- So, we have capped the upper age limit of all the users greater than 95 to 95 and in addition we have also capped the lower limit to 5.

We have plotted comparison box-plots of age before and after cleaning the data. Below is the graph.



The below can be inferred from the graph.

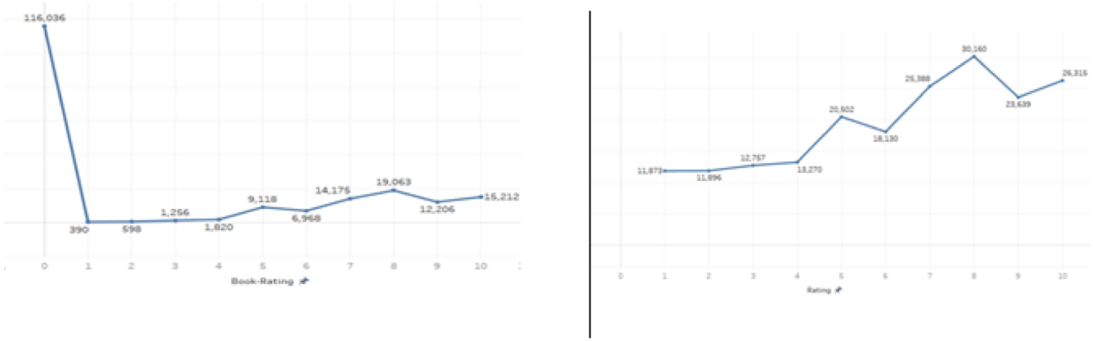
Before Cleaning: Range of age was [0,244], mean was 36.08.

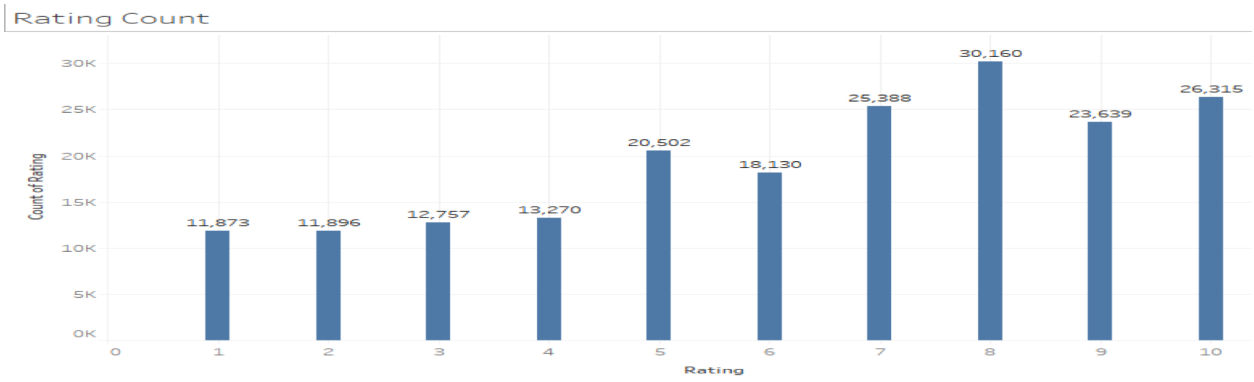
After Cleaning: Range of age was [5,95], mean was 35.6.

Rating:

- Initially, the rating column was ranging from 0 to 10. With more than half the rows being either NULL or zeroes.
- The approach we have followed for cleaning these NULL values or zeroes is that we have checked the rating given by other users for this same book and then taken the mean of ratings received for this book, thereby replacing the zeroes or nulls with this mean of ratings (rounding off to the nearest integer).

Below is the comparison of Age before and after cleaning the data.





The below can be inferred from the graph.

Before Cleaning: Range of rating was [0,10], mean was 3.1

After Cleaning: Range of rating was [1,10], mean was 6.35

Also the graph above gives the count of books for each rating.

A Simple Popularity Model

A simple approach that recommends the most highly rated books by the user on the entire dataset. Basically, it recommends the same set of books to each user since popularity is defined on the entire dataset. This is fast and inappropriate method to recommend the list of books to the users as there is no personalization involved with this approach.

To get the best results out of this model, we considered only those customers who have rated atleast 20 books on the dataset. We have run the model with k = 3, it means that we are recommending top 3 books to each user.

UserID	ISBN	score	rank
92652	2277330140	10.0	1
92652	1556433468	10.0	2
92652	1579540112	10.0	3
165308	2277330140	10.0	1
165308	1556433468	10.0	2
165308	1579540112	10.0	3
168245	2277330140	10.0	1
168245	1556433468	10.0	2
168245	1579540112	10.0	3
16634	2277330140	10.0	1
16634	1556433468	10.0	2
16634	1579540112	10.0	3

Output of Simple Popularity Model

As the output shows, the top 3 highly rated books with ISBN are 2277330140, 1556433468, 1579540112. Also, these 3 books have been recommended to all the users. Thus we can see that our popularity system works as expected. But it is not a good enough as the personalization of each user doesn't exit.

A Collaborative Filtering Model

It is the most commonly used algorithm for recommendation systems as it works based on the past history of the user and not on the context and not dependent on any additional information.

There are two types of collaborative filtering algorithms:

- User-User Collaborative filtering
- Item-Item Collaborative filtering

We chose Item-Item Collaborative filtering over User-User Collaborative filtering for our dataset because of two reasons:

- There are more number of users as compared to the numbers of books in our dataset.
- Users profile are always dynamic. So, it would not be a great model for future.

How Item-Item Collaborative filtering works?

Rather than matching user with similar users, this algorithm matches each of the user's rated items to similar items by creating similar item matrix and then combines those similar items into a recommendation list.

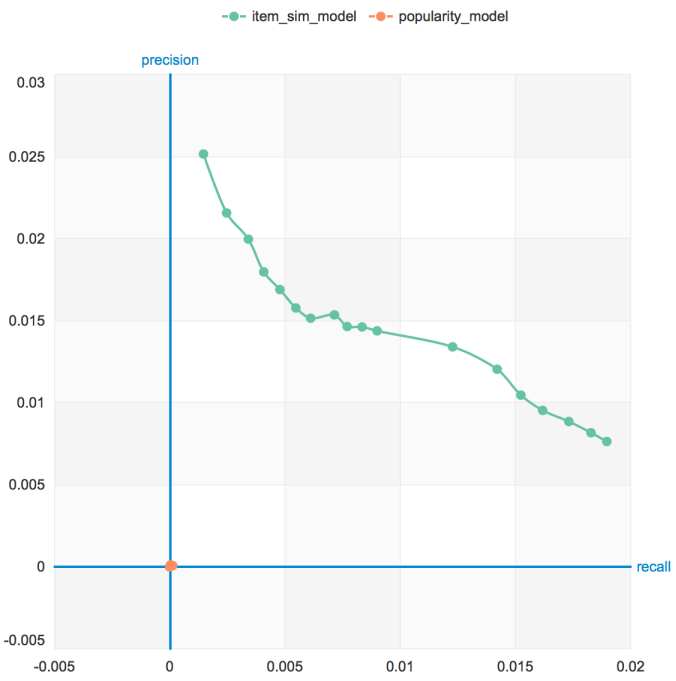
Again, we have run this model for k = 3 (recommending 3 books to each user).

UserID	ISBN	score	rank
215225	3442120772	0.215824087461	1
215225	3778736140	0.215824087461	2
215225	3550071671	0.215824087461	3
138441	7653064521	0.492897213499	1
138441	1590580664	0.492897213499	2
138441	3991472521	0.492897213499	3
159440	2253021539	0.184014995893	1
159440	2070568237	0.184014995893	2
159440	2266071173	0.184014995893	3
163202	1577292847	0.392837093936	1
163202	1569248184	0.392837093936	2
163202	1565041909	0.392837093936	3

Output of Simple Item-Item Collaborative Filtering Model

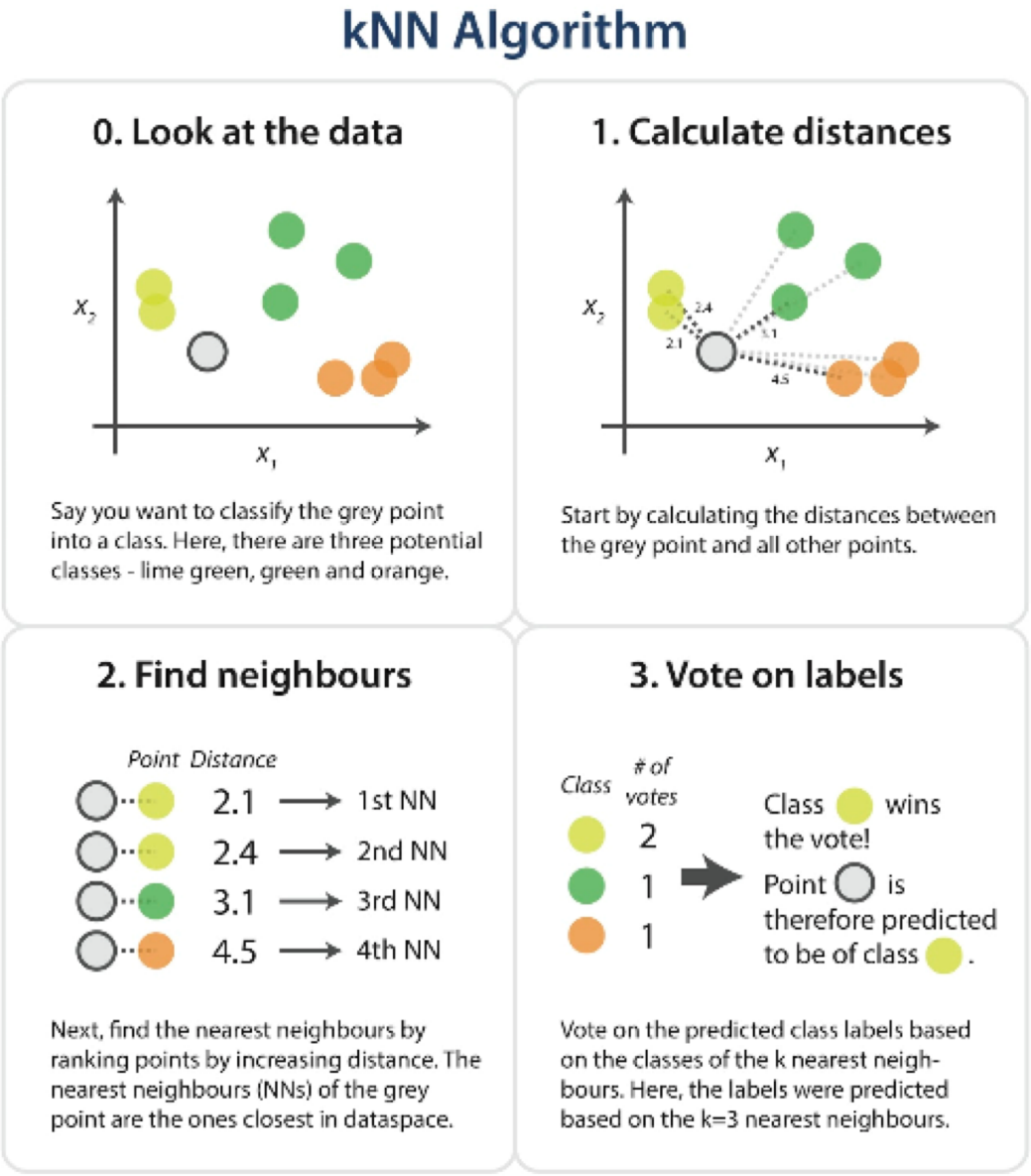
From the above output, we figured out that unlike the simple popularity model, all the users have different recommendation list based on their personalization.

Evaluating Recommendation Models



After evaluating both the models graphically, it is clear that recall and precision plot of popularity model is tending towards zero as compared to item based collaborative filtering. So, this model is certainly far better than simple popularity model as is considers the personalization of each user.

K-nearest neighbor Algorithm



K-Nearest Neighbor algorithm is a non-parametric method used for classification and regression. For both, the input consists of the k closest training examples in the feature space.

In K-nn regression the output value is average of value of it's k nearest neighbors. This can be useful to assign weights to contribution of neighbors, so the nearest neighbors contribute more to the average than the distant ones.

Distance metric is chosen based upon the type of the variables. For continuous variables, the most common distance used is Eucledian and for discrete we use the Hamming distance.

A common problem that occurs with k-nn is the majority voting problem. To overcome this problem, when we find that class distribution is skewed , we apply weights. In regression problems , each of the k nearest point is multiplied by a weight proportion to inverse of distance from point to the test points

Working of k-nn regression

- Estimating continuous variables:
    - Taking weighted average of k nearest neighbor, weighted by inverse of distance
- Order labelled examples by increasing distance
- Finding optimal number of k nearest neighbors
- Calculating the k – nearest

Feature selection algorithm to apply k-nn regression on feature set:

The feature dropping algorithm works in a way that features are turned off one after the another.

The leave-one-out test performed on the training file and results improve when some (unwanted) features are turned off and degrade when some important features are turned off.

Features leading to highest accuracy are selected as important whereas those that degrade the accuracy are considered to be least important and removed from the input set and whole procedure repeated till the input set left only with significant features.

We used this approach to select our features and we were left with three prominent features in our input set:

a) Author Name

b) Location-country wise

c) Age

For applying the k-nn regression we selected the KNeighborsRegressor library from scikit-learn package. The default K-nn Neigbors Regressor code is as follows

KNeighborsRegressor(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30 , p=2,

Metric='minkowski' , metric_params = None)

1) **N_neighbors (Finding the optimal K value)**

   As k-value increases, the cross validation (C.V) error goes down, after a point the C.V error goes up. Since larger the k-value, more smoothening happens and eventually results in underfitting. If the cross validation error doesn't start to rise again than the attribute not informative for distance metric. The default value is 5.

   The other strategy to follow is to choose k-value near to the square root of size of training set. Since the total input size of our data was 193801 , and we took a train-test split of 70:30 , the training set size was 135660 , and we settled for the for approximate square root , that is k-value of 369

2) **Weights**

   a) Uniform – All points weighted equally

   b) Distance – weigh points by inverse of distance. Closer neighbors have greater influence than neighbors which are further away.

      For our model, we went with the distance choice.

3) **Algorithms**

   We have choice of four options = ball_tree ,kd_tree , brute_force and auto.The auto option attempts to decide the most appropriate algorithm based on the

input values passed to the fit method.For our model , we went and selected the auto option.

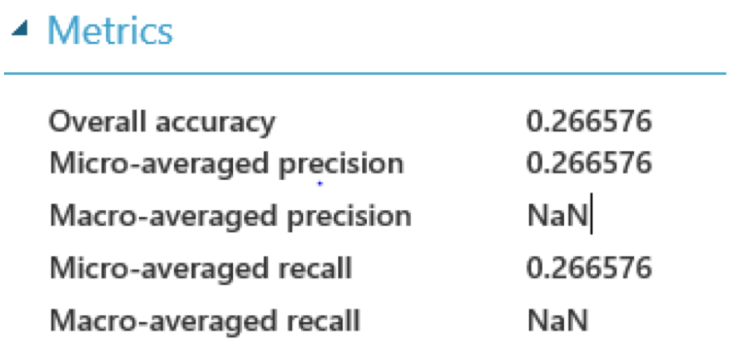
4) **Leaf size**

   By default this is 30. It is passed to Ball tree or KD tree. It affects the speed of construction of tree , the querying and hence the efficiency and runtime of code.

5) **Distance Metric**

   By default it is ‘Minkowski’. we can also select Eucledian or Manhattan or even give user defined function for distance calculation. The p value is option , the p value of 1 means manhattan while p=2 means Euclidean. We applied the model on Minkowski and Euclidean distances

Accuracy achieved:



Conclusion

As the dataset we had taken for our project is highly unstructured and we applied many data manipulation techniques as well, so we were able to achieve an accuracy of 27%. To improve this accuracy further, we will plan to use dimensionality reduction techniques like Principal Component Analysis(PCA) to reduce dimensionality.

Tools & References

https://turi.com/products/create/docs/generated/graphlab.SFrame.html

http:///~cziegler/BX/www2.informatik.uni-freiburg.de

Code File Names and Data Files

- **Simple Popularity and Collaborative Filtering Model**
    - GraphLabCF.ipynb or GraphLabCF.py
- **K-Nearest Neighbor Model**
    - KNN-Classifier.ipynb
    - KNN-Regressor.ipynb
- **Initial Dataset**
    - BX-Users.csv
    - BX-Ratings.csv
    - BX-Books.csv
- **Final Dataset**
    - Final_DataSet.xlsx (For Simple Popularity & Collaborative Filtering)
    - Fd_python.csv (For KNN-Regressor)
    - KNN_Classifier_Dataset.csv (For KNN-Classifier)