# Predictive Analysis for E-Commerce Website and Product Recommendation System

Vrushali Shah, Prashant Kabra

Under the guidance of Prof. Nicholas Brown

Northeastern University

Department of Information Systems

https://github.com/prashantk016/Predictive-Analysis-for-E-Commerce-Website-and-Product-Recommendation

**Abstract.** Recommendation System is the knowledge discovery techniques and use of statistics to deliver users with personalized content and service. It is used to solve the interaction with the customers which are targeted to provide product recommendation issue. We found this a challenging task to build a recommendation system. Customers these days are dependent on recommendations whether it is for products to purchase, news on recent launches, restaurants to visit or services to avail. Recommender systems solve this problem of searching through large volume of dynamically generated information to provide users with personalized contents and services. We found that more than half of the recommendation approaches applied content-based filtering (55 %). Collaborative filtering was applied by only 18 % of the reviewed approaches and graph-based recommendations by 16 %. Other recommendation concepts included stereotyping, item-centric recommendations, and hybrid recommendations. In this project, we attempt to understand different kind of algorithms for recommendation systems and compare their performance on E-commerce dataset. First, it remains unclear which recommendation concepts and approaches are most promising. We tried to research deep into filtering techniques and apply for our dataset to evaluate recommender system adequately. We have used Supervised Learning Algorithms like Linear and Logistic Regression, K-Nearest Neighbor, Gaussian Naïve Bayes, Decision Tree Regressor, Random Forest Classifier, Support Vector Machine, Gradient Boosting Classifier and Collaborative Filtering Algorithms like Alternating Least Square (ALS). We have also implemented Natural Language Processing using Bag of Words, TF-IDF and Hashing for Sentiment Analysis, Deep Learning using Recurrent Neural Network and Stacked Ensemble model for our recommender system.

## I. INTRODUCTION

Traditionally, people used to buy product from the stores. But now -a- days people prefer online shopping. Massive adoption of internet/web as an e-commerce platform led to change in the way business interact with their users. Recommendation System is the method to provide customers with suggestions about the product they could buy.

In this paper, we have taken the dataset from data. world and we tried to provide recommendations to our users for several products having similar taste as of our other customer. Our dataset has relevant columns like UserID, ProductID, ProductDescription, Rating, DoReccomended, DidPurchase columns to do predictive analysis. We applied various machine learning and deep learning algorithms based on regression, clustering classification and found Collaborative filtering ALS Method to be most suitable algorithms for Recommendation Systems.

The recommendation system is produced basically by two types of approaches:

**1. Collaborative Filtering** (Item based, User based)

**2. Content- based Filtering.**

In collaborative filtering we find how many of users or items in the data are like other user and using correlation and cosine similarity find item for others. While the Content- based recommendation works on the rating/data provided explicitly or implicitly by the user. BBR provides recommendation which are highly personalized by matching user interest and description. We use standard Machine learning algorithms like Regression, SVM, Decision Tress and Logistic Regression for making prediction in CBR.

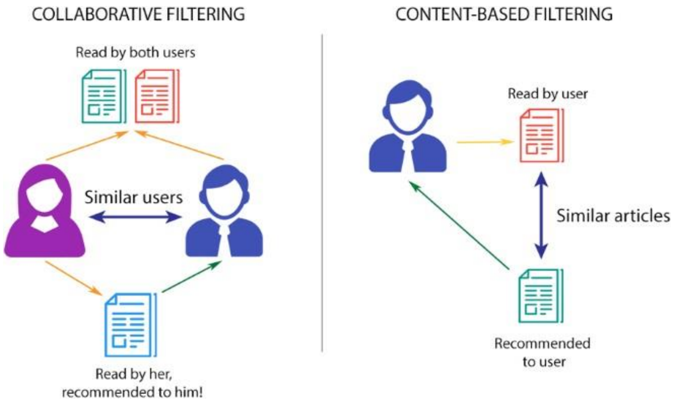


Fig. 1. Process of Recommendation System

Recommendation System is the knowledge discovery techniques and use of statistics to deliver users with personalized content and service. It is used to solve the interaction with the customers which are targeted to provide product recommendation issue.

Machine learning algorithms are commonly used to do meaningful analysis and allow computer to learn human behavior or nature and improve the performance of new task on old analysis. Additional benefits of recommendation system are:

**1. Convert Visitor to Buyer:** Recommendation systems help consumers find items that best fit the customers interests and inclinations and many times these lead to unplanned purchases driven by the buyer just because of the suggestion made.

**2. Increase in Cross-sell:** Giving Recommendation for products helps improve in cross selling by suggesting more products or services to customers. If the suggestions are perfect, the average order size increases.

**3. Create Value- added relationship:** In a competitive world where everything is one click away, building customer-loyalty becomes an essential aspect of business. Each time a customer visits a website, the system “learns” more about that customer’s preferences and interests and is increasingly able to operationalize this information to e.g. personalize what is offered. By providing each customer with an increasingly relevant experience, a corresponding improvement in the likelihood of that customer returning is achieved.

## II. EVALUATE ALGORITHM ACCURACY

### Algorithm error

The error in algorithm helps you to find the most suitable algorithm for your dataset and helps to obtain intended results. In this system, we used few statistical measures to find out the best algorithm and highest correlation. Some of these are listed below.

**1. R-squared:** It is a statistical measure of how close the data are to the fitted regression line. The higher the value, the better the model fits the dataset.

**2. Root Mean Square Error (RMSE):** It is a frequently used measure of the differences between values (sample and population values) predicted by a model or an estimator and the values observed. The lower the value, the better the model fits the dataset.

**3. Loss Function:** It is used for parameter estimation that maps an event or values of one or more variables onto a real variable intuitively representing some "cost" associated with the event. The main problem is to minimize a loss function.

## Dataset

The dataset which we are using is from data.world with 100k rows and 25 columns. We have an E-commerce dataset with details of Product like EAN number, Product Id, Name, Category, Date Added, Date Updated, Rating, Product URL, Product Manufacturer etc. Dataset also contains user information like User Id, Username, User Review for product, User Rating for product, User City, User Purchase and User Recommendation for product. User recommendation columns consist of True/False value to show if the user recommends a product or not. Rating consist of value between 1 to 5. The E-commerce dataset has approximately 72K rows and 25 columns after filtering and removing rows which are cannot be used by performing Exploratory Data Analysis and Data Cleaning.

In Algorithms used, we split dataset into two partitions: Test and train dataset by sampling in the ratios 20% and 80% respectively.

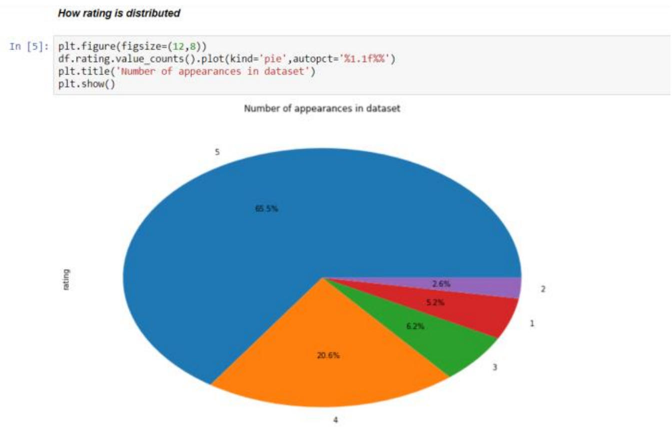
We aim to achieve the following for our system:
**1. Suggest:** Provide related items for the users from relevant and irrelevant collection of items.

**2. Predict:** Given a data of items purchased by Customers, we are trying to predict items for the user which can be useful for them based on user's past purchase history and location of the user.

**3. Forecast:** Demand forecasting can also be made of items according to the item sold in a country/continent.

## Data Visualization

After having a quick look, we understand that the dataset consists columns of different datatypes such as object, float, integer. Few rows/columns are empty and have no suitable values (NaN) which needs to be processed to get more accurate results.

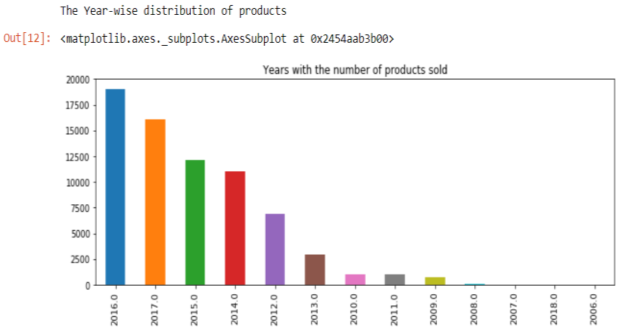


*How rating is distributed*

```
In [5]: plt.figure(figsize=(12,8))
        df.rating.value_counts().plot(kind='pie',autopct='%1.1f%%')
        plt.title('Number of appearances in dataset')
        plt.show()
```

Fig. 2. Distribution of Rating

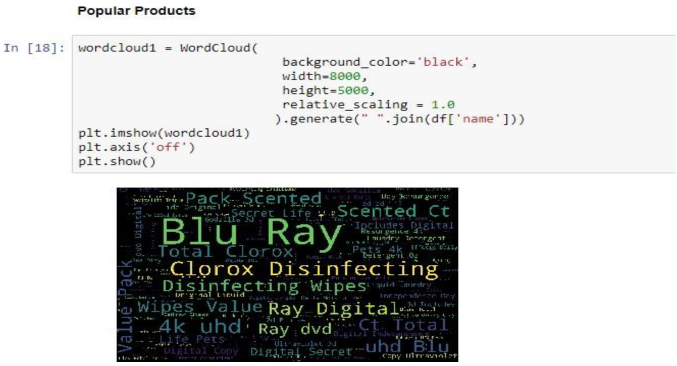
Figure 2 shows the range for giving product rating i.e. from 1 to 5 with 1 being the least rating and 5 being the highest rating.



```
The Year-wise distribution of products

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x2454aab3b00>
```

Fig. 3.Year-wise distribution of products

Figure 3 shows year-wise distribution of products from 2006-2016.



**Popular Products**

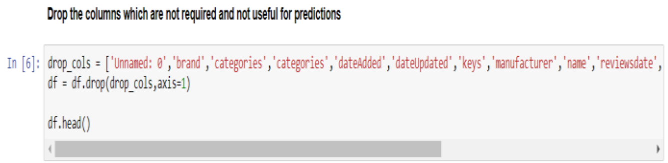
```
In [18]: wordcloud1 = WordCloud(
                                         background_color='black',
                                         width=8000,
                                         height=5000,
                                         relative_scaling = 1.0
                                        ).generate(" ".join(df['name']))
         plt.imshow(wordcloud1)
         plt.axis('off')
         plt.show()
```

Fig. 4. Work Cloud

In addition, we also analyzed the words which are most repeated in user comments for products. Using this we can build content-based recommendation system.

### Data Processing

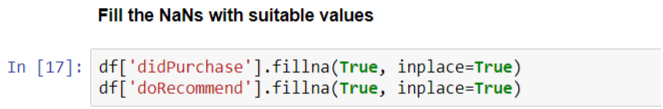
We start by filtering data and defining which columns will be used for recommendation algorithms. In most of the cases, we have not used the rows which have any kind of reviews or purchases and no recommendations while in some cases, NaNs have been replaced with suitable values. The columns which are irrelevant have been dropped so that they do not affect predictions.



Drop the columns which are not required and not useful for predictions

```
In [6]: drop_cols = ['Unnamed: 0','brand','categories','categories','dateAdded','dateUpdated','keys','manufacturer','name','reviewsdate',
        df = df.drop(drop_cols,axis=1)

        df.head()
```

Fig. 5. Drop columns not useful for predictions



Fill the NaNs with suitable values

```
In [17]: df['didPurchase'].fillna(True, inplace=True)
         df['doRecommend'].fillna(True, inplace=True)
```

Fig. 6. Fill Nans with suitable values

### Feature Selection

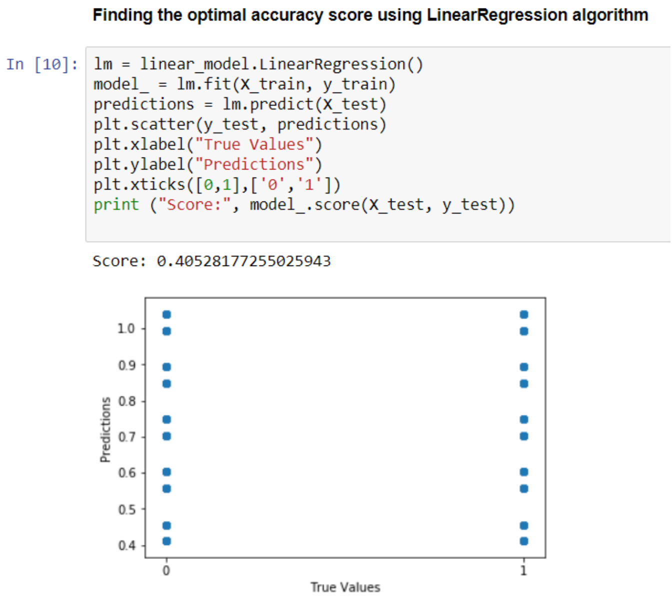
In this system, we select the most relative feature by generating the score of each feature and selecting the most relevant one. We have columns like Rating, Reviews, doRecommend (True/False), didPurchase (True/False) which are very much effective in our recommender systems.

## III. MACHINE LEARNING ALGORITHMS

### 1. Classification Algorithms:

### a) Linear Regression

Linear regression is the starting point of many statistical modeling and predictive analysis projects. The importance of fitting (accurately and quickly) a linear model to a large data set cannot be overstated. It is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables. The case of one explanatory variable is called simple linear regression. In simple words linear regression is predicting the value of a variable Y (dependent variable) based on some variable X (independent variable) provided there is a linear relationship between X and Y. This linear relationship between the 2 variables can be represented by a straight line (called regression line). Thus, such models are very popular and are very interpretable.



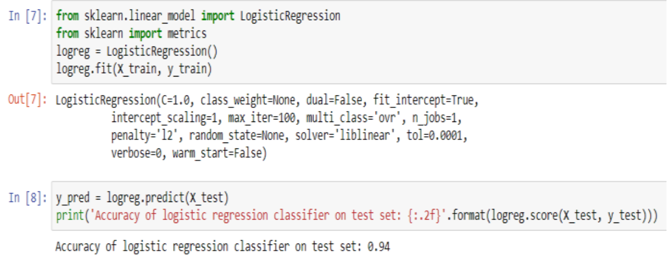
Finding the optimal accuracy score using LinearRegression algorithm

```
In [10]: lm = linear_model.LinearRegression()
         model_ = lm.fit(X_train, y_train)
         predictions = lm.predict(X_test)
         plt.scatter(y_test, predictions)
         plt.xlabel("True Values")
         plt.ylabel("Predictions")
         plt.xticks([0,1],['0','1'])
         print ("Score:", model_.score(X_test, y_test))

         Score: 0.40528177255025943
```

Fig. 7. Accuracy using Linear Regression

### b) Logistic Regression

Logistic regression is a machine learning algorithm for classification. In this algorithm, the probabilities describing the possible outcomes of a single trial are modelled using a logistic function. However, it works only when the predicted variable is binary, assumes all predictors are independent of each other, and assumes data is free of missing values.

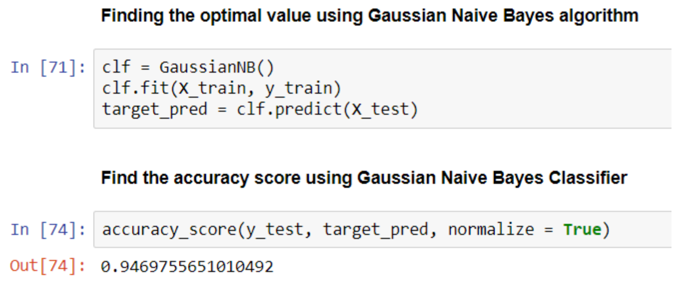


```
In [7]: from sklearn.linear_model import LogisticRegression
        from sklearn import metrics
        logreg = LogisticRegression()
        logreg.fit(X_train, y_train)

Out[7]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                  intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                  penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                  verbose=0, warm_start=False)

In [8]: y_pred = logreg.predict(X_test)
        print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.score(X_test, y_test)))

        Accuracy of logistic regression classifier on test set: 0.94
```

Fig. 8. Accuracy using Logistic Regression

### c) Gaussian Naïve Bayes

Naive Bayes algorithm based on Bayes' theorem with the assumption of independence between every pair of features. This classifier work well in many real-world situations such as document classification and spam filtering. It requires a small amount of training data to estimate the necessary parameters and are extremely fast compared to more sophisticated methods. However, it is known to be a bad estimator.



**Finding the optimal value using Gaussian Naive Bayes algorithm**

```
In [71]: clf = GaussianNB()
         clf.fit(X_train, y_train)
         target_pred = clf.predict(X_test)
```

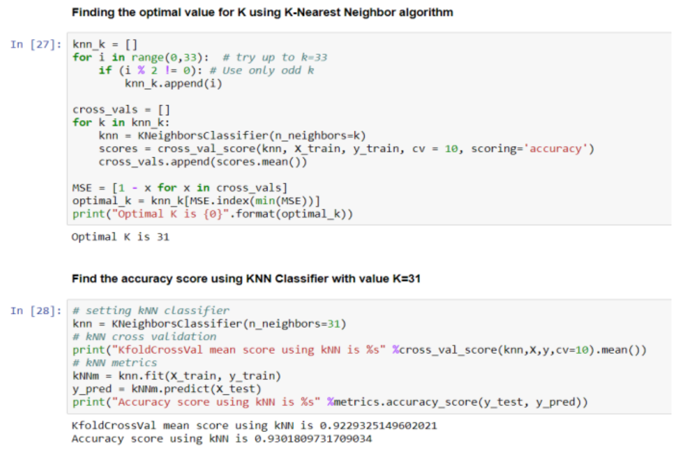
**Find the accuracy score using Gaussian Naive Bayes Classifier**

```
In [74]: accuracy_score(y_test, target_pred, normalize = True)

Out[74]: 0.9469755651010492
```

Fig. 9. Accuracy using Gaussian Naïve Bayes

### d) K-Nearest Neighbor

Neighbors based classification is a type of lazy learning as it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the k nearest neighbors of each point. This algorithm is simple to implement, robust to noisy training data, and effective if training data is large. It needs to determine the value of K and the computation cost is high as it needs to compute the distance of each instance to all the training samples.



**Finding the optimal value for K using K-Nearest Neighbor algorithm**

```
In [27]: knn_k = []
         for i in range(0,33):  # try up to k=33
             if (i % 2 != 0): # Use only odd k
                 knn_k.append(i)

         cross_vals = []
         for k in knn_k:
             knn = KNeighborsClassifier(n_neighbors=k)
             scores = cross_val_score(knn, X_train, y_train, cv = 10, scoring='accuracy')
             cross_vals.append(scores.mean())

         MSE = [1 - x for x in cross_vals]
         optimal_k = knn_k[MSE.index(min(MSE))]
         print("Optimal K is {0}".format(optimal_k))

         Optimal K is 31
```

**Find the accuracy score using KNN Classifier with value K=31**

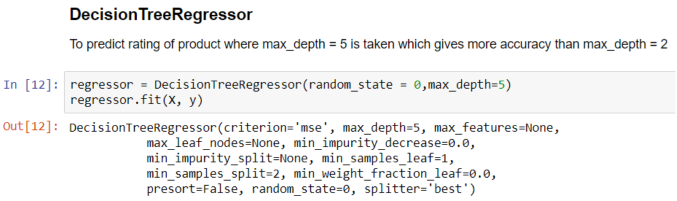
```
In [28]: # setting kNN classifier
         knn = KNeighborsClassifier(n_neighbors=31)
         # kNN cross validation
         print("KfoldCrossVal mean score using kNN is %s" %cross_val_score(knn,X,y,cv=10).mean())
         # kNN metrics
         kNNm = knn.fit(X_train, y_train)
         y_pred = kNNm.predict(X_test)
         print("Accuracy score using kNN is %s" %metrics.accuracy_score(y_test, y_pred))

         KfoldCrossVal mean score using kNN is 0.9229325149602021
         Accuracy score using kNN is 0.9301809731709034
```

Fig. 10. Accuracy using K-Nearest Neighbor with optimal value of K

In the above figure, we can see that the most accurate results were found with K=31. This is computed automatically based on the type of data present in the dataset.

### e) Decision Tree

Given a data of attributes together with its classes, a decision tree produces a sequence of rules that can be used to classify the data. It is simple to understand and visualize, requires little data preparation and can handle both numerical and categorical data. However, it can create complex trees that do not generalize well, and decision trees can be unstable because small variations in the data might result in a completely different tree being generated.



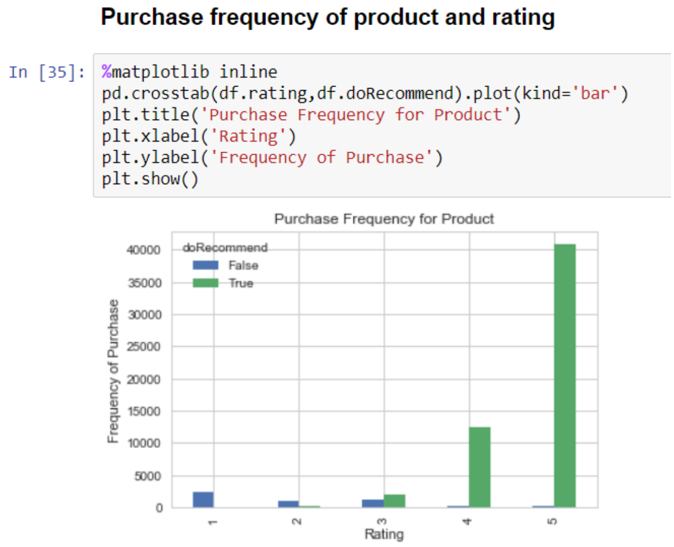
**DecisionTreeRegressor**

To predict rating of product where max_depth = 5 is taken which gives more accuracy than max_depth = 2

```
In [12]: regressor = DecisionTreeRegressor(random_state = 0,max_depth=5)
         regressor.fit(X, y)

Out[12]: DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=None,
                    max_leaf_nodes=None, min_impurity_decrease=0.0,
                    min_impurity_split=None, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    presort=False, random_state=0, splitter='best')
```

Fig. 11. Decision Tree results with max_depth

With max_depth = 5, we get more accuracy as compared to smaller values for depth.



**Purchase frequency of product and rating**

```
In [35]: %matplotlib inline
         pd.crosstab(df.rating,df.doRecommend).plot(kind='bar')
         plt.title('Purchase Frequency for Product')
         plt.xlabel('Rating')
         plt.ylabel('Frequency of Purchase')
         plt.show()
```

Fig. 12. Purchase frequency of products based on Product Ratings

### f) Random Forest

Random forest classifier is a meta-estimator that fits several decision trees on various sub-samples of datasets and uses average to improve the predictive accuracy of the model and controls over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement. Reduction in over-fitting and random forest classifier is more accurate than decision trees in most cases. It is a complex algorithm and offers slow real-time prediction and is difficult to implement.

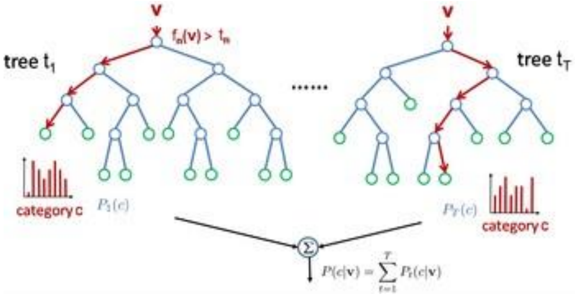
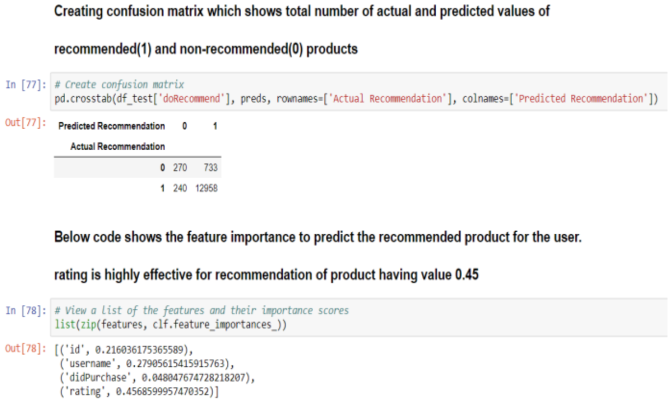


Fig. 13. Random Forest Classifier



Creating confusion matrix which shows total number of actual and predicted values of

recommended(1) and non-recommended(0) products

```
In [77]: # Create confusion matrix
         pd.crosstab(df_test['doRecommend'], preds, rownames=['Actual Recommendation'], colnames=['Predicted Recommendation'])
```

```
Out[77]: Predicted Recommendation    0      1
           Actual Recommendation
                               0   270    733
                               1   240  12958
```

Below code shows the feature importance to predict the recommended product for the user.

rating is highly effective for recommendation of product having value 0.45

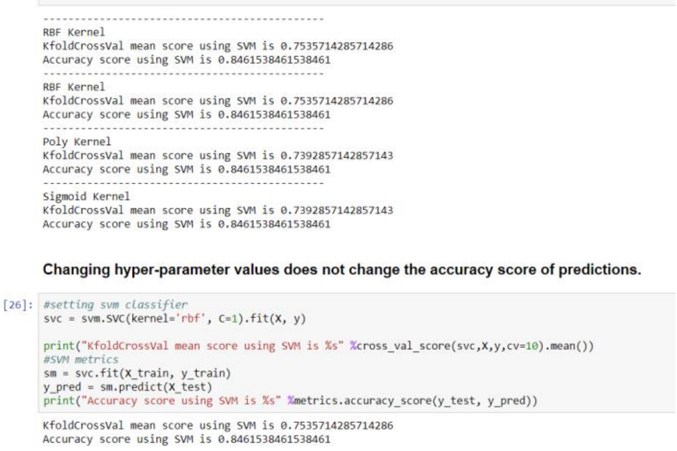
```
In [78]: # View a list of the features and their importance scores
         list(zip(features, clf.feature_importances_))

Out[78]: [('id', 0.216036175365589),
          ('username', 0.27905615415915763),
          ('didPurchase', 0.048047674728218207),
          ('rating', 0.4568599957470352)]
```

Fig. 14. Important features for recommending

Above figure shows the key features for recommending products to users are ProductId, Username, didPurchase and Rating columns.

### g) Support Vector Machine

Support vector machine is a representation of the training data as points in space separated into categories by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. It is effective in high dimensional spaces and uses a subset of training points in the decision function so it is also memory efficient. The algorithm does not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.



```
----------------------------------------------
RBF Kernel
KfoldCrossVal mean score using SVM is 0.7535714285714286
Accuracy score using SVM is 0.8461538461538461
----------------------------------------------
RBF Kernel
KfoldCrossVal mean score using SVM is 0.7535714285714286
Accuracy score using SVM is 0.8461538461538461
----------------------------------------------
Poly Kernel
KfoldCrossVal mean score using SVM is 0.7392857142857143
Accuracy score using SVM is 0.8461538461538461
----------------------------------------------
Sigmoid Kernel
KfoldCrossVal mean score using SVM is 0.7392857142857143
Accuracy score using SVM is 0.8461538461538461
```

Changing hyper-parameter values does not change the accuracy score of predictions.

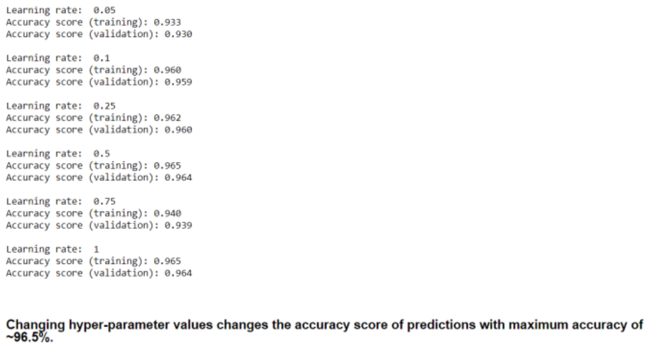
```
[26]: #setting svm classifier
      svc = svm.SVC(kernel='rbf', C=1).fit(X, y)

      print("KfoldCrossVal mean score using SVM is %s" %cross_val_score(svc,X,y,cv=10).mean())
      #SVM metrics
      sm = svc.fit(X_train, y_train)
      y_pred = sm.predict(X_test)
      print("Accuracy score using SVM is %s" %metrics.accuracy_score(y_test, y_pred))

      KfoldCrossVal mean score using SVM is 0.7535714285714286
      Accuracy score using SVM is 0.8461538461538461
```

Fig. 15. Accuracy of SVM by changing hyper-parameters

Analysis shows that changing hyper-parameters does not alter the accuracy of predictions.

### h) Gradient Boosting Classifier

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It is a greedy algorithm and can overfit a training dataset quickly. It can benefit from regularization methods that penalize various parts of the algorithm and generally improve the performance of the algorithm by reducing overfitting. GB builds an additive model in a forward stage-wise fashion and it allows for the optimization of arbitrary differentiable loss functions.



```
Learning rate:  0.05
Accuracy score (training): 0.933
Accuracy score (validation): 0.930

Learning rate:  0.1
Accuracy score (training): 0.960
Accuracy score (validation): 0.959

Learning rate:  0.25
Accuracy score (training): 0.962
Accuracy score (validation): 0.960

Learning rate:  0.5
Accuracy score (training): 0.965
Accuracy score (validation): 0.964

Learning rate:  0.75
Accuracy score (training): 0.940
Accuracy score (validation): 0.939

Learning rate:  1
Accuracy score (training): 0.965
Accuracy score (validation): 0.964
```

Changing hyper-parameter values changes the accuracy score of predictions with maximum accuracy of ~96.5%.

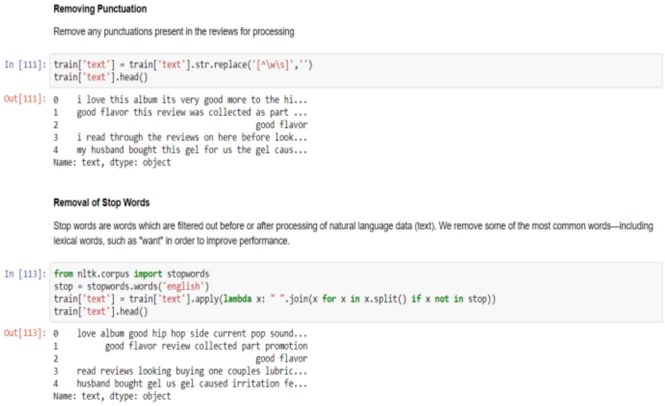
Fig. 16. Accuracy using GB Classifier by changing hyper-parameters

Analysis shows that changing hyper-parameters alters the accuracy of predictions.

## 2. Natural Language Processing (NLP):

Natural Language Processing, or NLP for short, is broadly defined as the automatic manipulation of natural language, like speech and text, by software. NLP is a collective term referring to automatic computational processing of human languages. This includes both algorithms that take human-produced text as input, and algorithms that produce natural looking text as outputs.

We have analyzed the Review column of our dataset to understand customer reviews for each product. We have done basic pre-processing for the text such as removal of special characters (@,#), digits, punctuations, stopwords etc.



**Removing Punctuation**

Remove any punctuations present in the reviews for processing

In [111]:
```
train['text'] = train['text'].str.replace('[^\w\s]','')
train['text'].head()
```

Out[111]:
```
0    i love this album its very good more to the hi...
1    good flavor this review was collected as part ...
2                                          good flavor
3    i read through the reviews on here before look...
4    my husband bought this gel for us the gel caus...
Name: text, dtype: object
```

**Removal of Stop Words**

Stop words are words which are filtered out before or after processing of natural language data (text). We remove some of the most common words—including lexical words, such as "want" in order to improve performance.

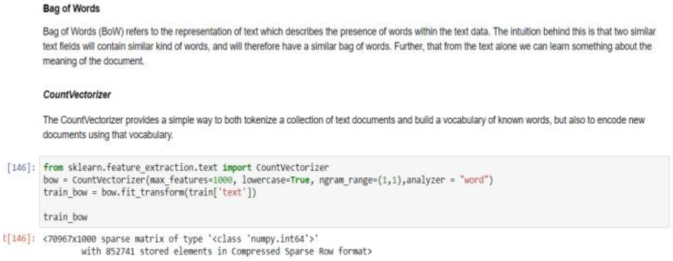
In [113]:
```
from nltk.corpus import stopwords
stop = stopwords.words('english')
train['text'] = train['text'].apply(lambda x: " ".join(x for x in x.split() if x not in stop))
train['text'].head()
```

Out[113]:
```
0    love album good hip hop side current pop sound...
1          good flavor review collected part promotion
2                                          good flavor
3    read reviews looking buying one couples lubric...
4    husband bought gel us gel caused irritation fe...
Name: text, dtype: object
```

Fig. 17. Removing punctuations and stopwords

### a) Bag of Words

In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. In the given dataset we considered the column “text” which contains the reviews given by a user for a product to achieve this concept. Using regular expression, we search a pattern and clean the data by removing the punctuation marks and special characters. By importing stop words from natural language toolkit, we later try to segregate the important and frequent used terms from the familiar words of English dictionary. We used Count Vectorizer to count the number of times a word occurs in a corpus. Hence, we display a list of words with their counts which are important and frequent in a corpus.



**Bag of Words**

Bag of Words (BoW) refers to the representation of text which describes the presence of words within the text data. The intuition behind this is that two similar text fields will contain similar kind of words, and will therefore have a similar bag of words. Further, that from the text alone we can learn something about the meaning of the document.

***CountVectorizer***

The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.

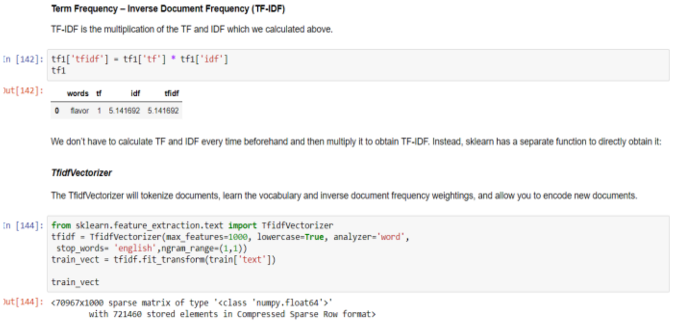
[146]:
```
from sklearn.feature_extraction.text import CountVectorizer
bow = CountVectorizer(max_features=1000, lowercase=True, ngram_range=(1,1),analyzer = "word")
train_bow = bow.fit_transform(train['text'])

train_bow
```

t[146]:
```
<70967x1000 sparse matrix of type '<class 'numpy.int64'>'
        with 852741 stored elements in Compressed Sparse Row format>
```

Fig. 18. Bag of Words using CountVectorizer

### b) TD-IDF

TF-IDF stands for "Term Frequency, Inverse Document Frequency." It's a way to score the importance of words (or "terms") in a document based on how frequently they appear across multiple documents or corpus. If a word appears frequently in a document, it's important. Give the word a high score. But if a word appears in many documents, it's not a unique identifier. Give the word a low score. Therefore, common words like "the" and "for," which appear in many documents, will be scaled down. Words that appear frequently in a single document will be scaled up.



**Term Frequency – Inverse Document Frequency (TF-IDF)**

TF-IDF is the multiplication of the TF and IDF which we calculated above.

In [142]:
```
tf1['tfidf'] = tf1['tf'] * tf1['idf']
tf1
```

Out[142]:

|  | words | tf | idf | tfidf |
|---|---|---|---|---|
| 0 | flavor | 1 | 5.141692 | 5.141692 |

We don't have to calculate TF and IDF every time beforehand and then multiply it to obtain TF-IDF. Instead, sklearn has a separate function to directly obtain it:

***TfidfVectorizer***

The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents.

In [144]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(max_features=1000, lowercase=True, analyzer='word',
 stop_words= 'english',ngram_range=(1,1))
train_vect = tfidf.fit_transform(train['text'])

train_vect
```

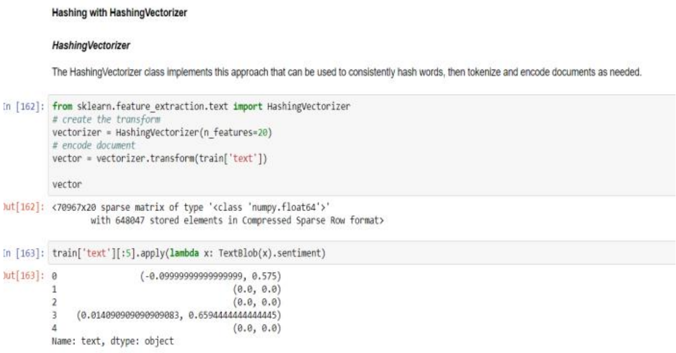
Out[144]:
```
<70967x1000 sparse matrix of type '<class 'numpy.float64'>'
        with 721460 stored elements in Compressed Sparse Row format>
```

Fig. 19. TF-IDF using TfidfVectorizer

### c) Hashing

Hashing Vectorizer applies a hashing function to term frequency counts in each document or corpus. Hash functions are an efficient way of mapping terms to features; it doesn’t necessarily need to be applied only to term frequencies but that’s how Hashing

Vectorizer is employed here. Depending on the use case for the word vectors, it may be possible to reduce the length of the hash feature vector (and thus complexity) significantly with acceptable loss to accuracy/effectiveness (due to increased collision). If the hashing matrix is wider than the dictionary, it will mean that many of the column entries in the hashing matrix will be empty, and not just because a given document doesn't contain a specific term but because they're empty across the whole matrix. If it is not, it might send multiple terms to the same feature hash - this is the 'collision'.



**Hashing with HashingVectorizer**

***HashingVectorizer***

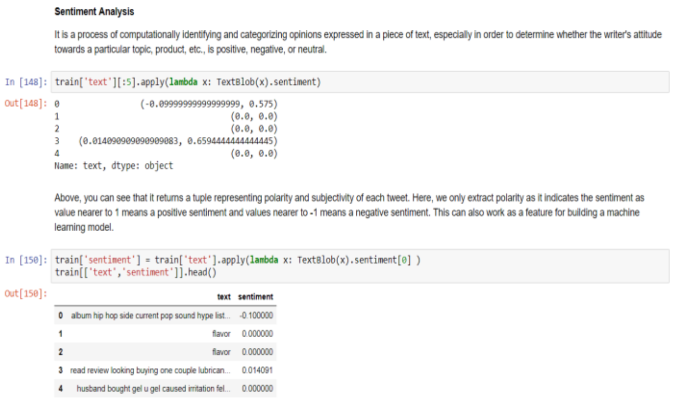
The HashingVectorizer class implements this approach that can be used to consistently hash words, then tokenize and encode documents as needed.

```
In [162]: from sklearn.feature_extraction.text import HashingVectorizer
          # create the transform
          vectorizer = HashingVectorizer(n_features=20)
          # encode document
          vector = vectorizer.transform(train['text'])

          vector
Out[162]: <70967x20 sparse matrix of type '<class 'numpy.float64'>'
                  with 648047 stored elements in Compressed Sparse Row format>

In [163]: train['text'][:5].apply(lambda x: TextBlob(x).sentiment)
Out[163]: 0               (-0.09999999999999999, 0.575)
          1                                 (0.0, 0.0)
          2                                 (0.0, 0.0)
          3    (0.014090909090909083, 0.6594444444444445)
          4                                 (0.0, 0.0)
          Name: text, dtype: object
```

Fig. 20. Hashing using HashingVectorizer

**Sentiment Analysis**

Sentiment is like a combination of tone of voice, word choice, and writing style all rolled into one. Natural language with labels about positivity or negativity (or any other spectrum we want to gauge), we can develop agents that can learn to understand the sentiments underlying new messages. Using NLP, we can understand what people like and dislike about products by crawling the thousands of reviews already posted on the website.



**Sentiment Analysis**

It is a process of computationally identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writer's attitude towards a particular topic, product, etc., is positive, negative, or neutral.

```
In [148]: train['text'][:5].apply(lambda x: TextBlob(x).sentiment)
Out[148]: 0               (-0.09999999999999999, 0.575)
          1                                 (0.0, 0.0)
          2                                 (0.0, 0.0)
          3    (0.014090909090909083, 0.6594444444444445)
          4                                 (0.0, 0.0)
          Name: text, dtype: object
```

Above, you can see that it returns a tuple representing polarity and subjectivity of each tweet. Here, we only extract polarity as it indicates the sentiment as value nearer to 1 means a positive sentiment and values nearer to -1 means a negative sentiment. This can also work as a feature for building a machine learning model.

```
In [150]: train['sentiment'] = train['text'].apply(lambda x: TextBlob(x).sentiment[0] )
          train[['text','sentiment']].head()
Out[150]:
```

|  | text | sentiment |
|---|---|---|
| 0 | album hip hop side current pop sound hype list... | -0.100000 |
| 1 | flavor | 0.000000 |
| 2 | flavor | 0.000000 |
| 3 | read review looking buying one couple lubrican... | 0.014091 |
| 4 | husband bought gel u gel caused irritation fel... | 0.000000 |

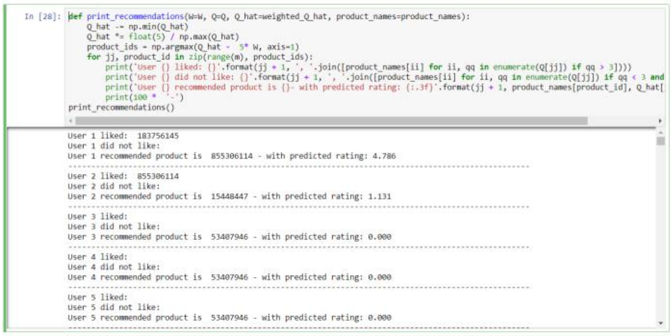
Fig. 21. Sentiment Analysis for reviews

## 3. Matrix Factorization:

### a) Alternating Least Square (ALS)

The alternating least squares (ALS) algorithm is a well-known algorithm for collaborative filtering. It nowadays is available as the standard algorithm for recommendations. It is a two-step iterative optimization process. Since OLS solution is unique and guarantees a minimal MSE, in each step the cost function can either decrease or stay unchanged, but never increase. Alternating between the two steps guarantees reduction of the cost function, until convergence. Similar to gradient descent optimization, it is guaranteed to converge only to a local minima. Since the actual cost function includes a regularization term, it is slightly longer. ALS is that it can be easily parallelized since calculations for each user and each item are independent. So, each iteration in a loop above could be done in parallel.

We have reduced the size of data for predictions where we have created a matrix for 500 users and 200 products out of 58k users and 600 products for the products purchased by our users. This enables us to analyze likes, dislikes and products which are likely to be recommended by the customers.



```
In [28]: def print_recommendations(W=W, Q=Q, Q_hat=weighted_Q_hat, product_names=product_names):
             Q_hat -= np.min(Q_hat)
             Q_hat *= float(5) / np.max(Q_hat)
             product_ids = np.argmax(Q_hat - 5* W, axis=1)
             for jj, product_id in zip(range(m), product_ids):
                 print('User {} liked: {}'.format(jj + 1, ', '.join([product_names[ii] for ii, qq in enumerate(Q[jj]) if qq > 3])))
                 print('User {} did not like: {}'.format(jj + 1, ', '.join([product_names[ii] for ii, qq in enumerate(Q[jj]) if qq < 3 and
                 print('User {} recommended product is {}- with predicted rating: {:.3f}'.format(jj + 1, product_names[product_id], Q_hat[
                 print(100 * '-')
         print_recommendations()

User 1 liked:  183756145
User 1 did not like:
User 1 recommended product is  855306114 - with predicted rating: 4.786
----------------------------------------------------------------------------------------------------
User 2 liked:  855306114
User 2 did not like:
User 2 recommended product is  15448447 - with predicted rating: 1.131
----------------------------------------------------------------------------------------------------
User 3 liked:
User 3 did not like:
User 3 recommended product is  53407946 - with predicted rating: 0.000
----------------------------------------------------------------------------------------------------
User 4 liked:
User 4 did not like:
User 4 recommended product is  53407946 - with predicted rating: 0.000
----------------------------------------------------------------------------------------------------
User 5 liked:
User 5 did not like:
User 5 recommended product is  53407946 - with predicted rating: 0.000
----------------------------------------------------------------------------------------------------
```

Fig. 22. ALS for product recommendations

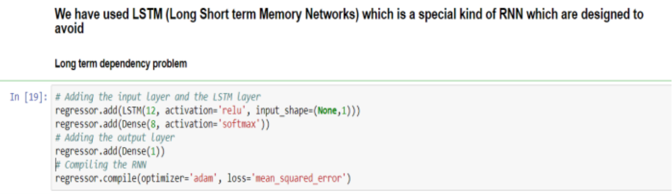
## 4. Deep Learning Algorithm:

### a) Recurrent Neural Network

Recurrent Neural Networks (RNN) are a powerful and robust type of neural networks and belong to the most promising algorithms out there at the moment because they are the only ones with an internal memory. Because of their internal memory, RNN’s can remember important things about the input they received, which enables them to be very precise in predicting what's coming next. The RNN maintains a vector of activation units for each time step in the sequence of data, this makes RNN extremely deep. A usual RNN has a short-term memory. In combination with a LSTM they also have a long-term memory. This memory can be seen as a gated cell, where gated means that the cell decides whether or not to store or delete information based on the importance it assigns to the information.

We have implemented RNN using Tensowflow which is low level implementations of the algorithms and a low-level API as well as Keras which simply focuses on defining layers for neural network and we don't have to deal with tensors.

Keras can use TensorFlow for deep learning backend runtime, most outputs are expected to be the same. Below results are analysis of RNN using Keras.

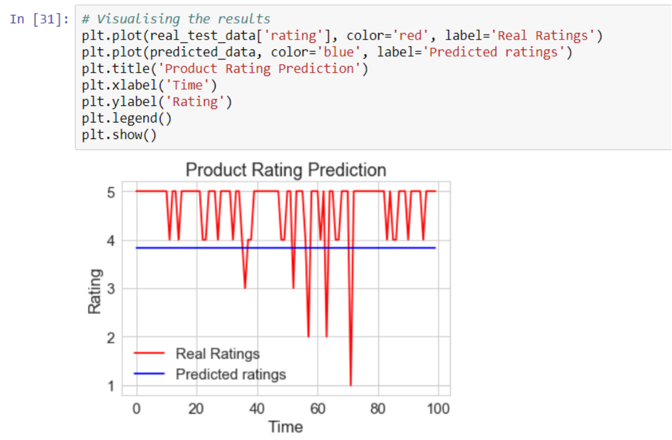


We have used LSTM (Long Short term Memory Networks) which is a special kind of RNN which are designed to avoid

Long term dependency problem

```
In [19]: # Adding the input layer and the LSTM layer
         regressor.add(LSTM(12, activation='relu', input_shape=(None,1)))
         regressor.add(Dense(8, activation='softmax'))
         # Adding the output layer
         regressor.add(Dense(1))
         # Compiling the RNN
         regressor.compile(optimizer='adam', loss='mean_squared_error')
```

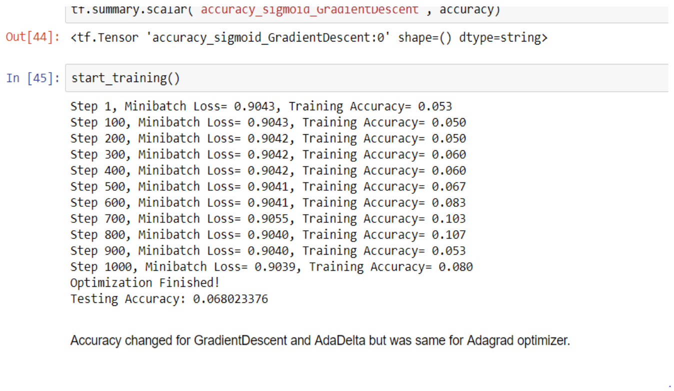
Fig. 23. RNN using LSTM layer



```
In [31]: # Visualising the results
         plt.plot(real_test_data['rating'], color='red', label='Real Ratings')
         plt.plot(predicted_data, color='blue', label='Predicted ratings')
         plt.title('Product Rating Prediction')
         plt.xlabel('Time')
         plt.ylabel('Rating')
         plt.legend()
         plt.show()
```

Fig. 24. RNN Real vs Predicted Ratings

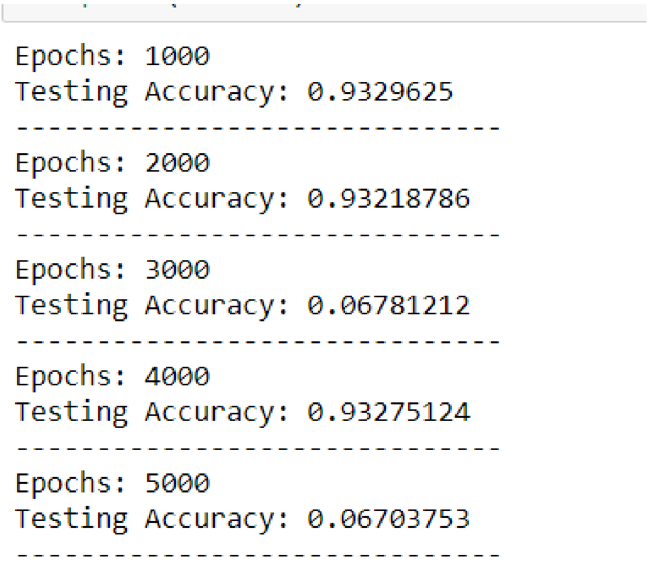
We also tried analyzing the results using different activation functions, cost functions, epochs range, gradient estimation, network initializations and its architecture and did not find much variations in the accuracy. However, for some cases, the accuracy varies drastically keeping other parameters the same.



```
         tf.summary.scalar('accuracy_sigmoid_GradientDescent', accuracy)
Out[44]: <tf.Tensor 'accuracy_sigmoid_GradientDescent:0' shape=() dtype=string>

In [45]: start_training()

         Step 1, Minibatch Loss= 0.9043, Training Accuracy= 0.053
         Step 100, Minibatch Loss= 0.9043, Training Accuracy= 0.050
         Step 200, Minibatch Loss= 0.9042, Training Accuracy= 0.050
         Step 300, Minibatch Loss= 0.9042, Training Accuracy= 0.060
         Step 400, Minibatch Loss= 0.9042, Training Accuracy= 0.060
         Step 500, Minibatch Loss= 0.9041, Training Accuracy= 0.067
         Step 600, Minibatch Loss= 0.9041, Training Accuracy= 0.083
         Step 700, Minibatch Loss= 0.9055, Training Accuracy= 0.103
         Step 800, Minibatch Loss= 0.9040, Training Accuracy= 0.107
         Step 900, Minibatch Loss= 0.9040, Training Accuracy= 0.053
         Step 1000, Minibatch Loss= 0.9039, Training Accuracy= 0.080
         Optimization Finished!
         Testing Accuracy: 0.068023376
```

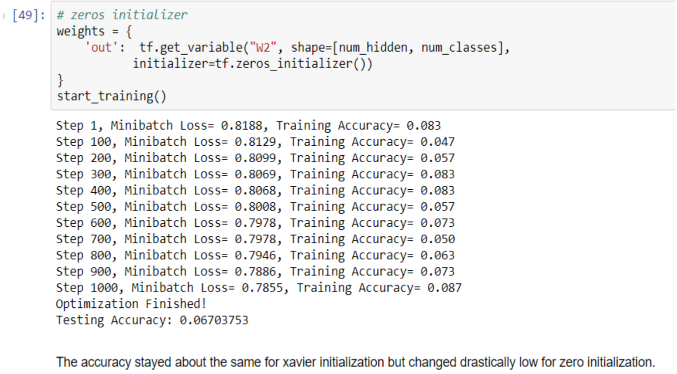
Accuracy changed for GradientDescent and AdaDelta but was same for Adagrad optimizer.

Fig. 25. RNN using different Gradient Estimation



```
Epochs: 1000
Testing Accuracy: 0.9329625
------------------------------
Epochs: 2000
Testing Accuracy: 0.93218786
------------------------------
Epochs: 3000
Testing Accuracy: 0.06781212
------------------------------
Epochs: 4000
Testing Accuracy: 0.93275124
------------------------------
Epochs: 5000
Testing Accuracy: 0.06703753
------------------------------
```

Fig. 26. RNN using varying Epochs



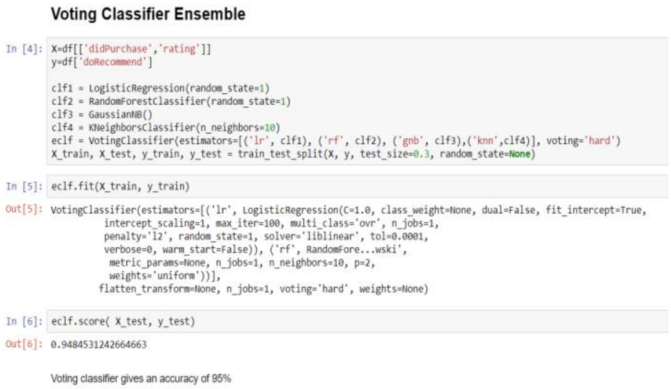
```
[49]: # zeros initializer
      weights = {
          'out':  tf.get_variable("W2", shape=[num_hidden, num_classes],
                  initializer=tf.zeros_initializer())
      }
      start_training()

      Step 1, Minibatch Loss= 0.8188, Training Accuracy= 0.083
      Step 100, Minibatch Loss= 0.8129, Training Accuracy= 0.047
      Step 200, Minibatch Loss= 0.8099, Training Accuracy= 0.057
      Step 300, Minibatch Loss= 0.8069, Training Accuracy= 0.083
      Step 400, Minibatch Loss= 0.8068, Training Accuracy= 0.083
      Step 500, Minibatch Loss= 0.8008, Training Accuracy= 0.057
      Step 600, Minibatch Loss= 0.7978, Training Accuracy= 0.073
      Step 700, Minibatch Loss= 0.7978, Training Accuracy= 0.050
      Step 800, Minibatch Loss= 0.7946, Training Accuracy= 0.063
      Step 900, Minibatch Loss= 0.7886, Training Accuracy= 0.073
      Step 1000, Minibatch Loss= 0.7855, Training Accuracy= 0.087
      Optimization Finished!
      Testing Accuracy: 0.06703753
```

The accuracy stayed about the same for xavier initialization but changed drastically low for zero initialization.

Fig. 27. RNN using different Network Initialization

## 5. Stacked Ensemble Model

Ensemble modeling is the process of running two or more related but different analytical models and then synthesizing the results into a single score or spread in order to improve the accuracy of predictive analytics and data mining applications. Stacking is a way of combining multiple models, that introduces the concept of a meta learner. Applying stacked models to real-world big data problems can produce greater prediction accuracy and robustness than do individual models. The model stacking approach is powerful and compelling enough to alter your initial data mining mindset from finding the single best model to finding a collection of good complementary models. Of course, this method does involve additional cost both because you need to train a large number of models and because you need to use cross validation to avoid overfitting.



Voting Classifier Ensemble

```
In [4]: X=df[['didPurchase','rating']]
        y=df['doRecommend']

        clf1 = LogisticRegression(random_state=1)
        clf2 = RandomForestClassifier(random_state=1)
        clf3 = GaussianNB()
        clf4 = KNeighborsClassifier(n_neighbors=10)
        eclf = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3),('knn',clf4)], voting='hard')
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=None)

In [5]: eclf.fit(X_train, y_train)

Out[5]: VotingClassifier(estimators=[('lr', LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                  intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                  penalty='l2', random_state=1, solver='liblinear', tol=0.0001,
                  verbose=0, warm_start=False)), ('rf', RandomFore...wski',
                   metric_params=None, n_jobs=1, n_neighbors=10, p=2,
                   weights='uniform'))],
                 flatten_transform=None, n_jobs=1, voting='hard', weights=None)

In [6]: eclf.score( X_test, y_test)

Out[6]: 0.9484531242664663
```

Voting classifier gives an accuracy of 95%

Fig. 28. Accuracy using Stacked Ensemble for Voting Classifier

Even after performing Cross validation, the accuracy did not increase. Instead it decreased the accuracy by approximately by 1.5%. But this gives us a clearer picture of the actual accuracy of the model.

## IV. RESULTS

Keeping the independent and dependent variables same across various algorithms, we found acceptable results of various algorithms used for our E-commerce dataset.

**Dependent variable:** doRecommend
**Independent variable:** ProductId, didPurchase, Username, Rating.

Since we take into consideration various independent variables and find the correlation between them, our aim to predict the likelihood for the user to recommend a product to another user is as follows:

| | |
|---|---|
| Linear Regression | ~40.5% |
| Logistic Regression | 94% |
| K-Nearest Neighbor | 93% |
| Gaussian Naïve Bayes | 94.69% |
| Decision Tree Regressor | 90% |
| Random Forest Classifier | 92.3% |
| Support Vector Machine | 84.6% |
| Gradient Boosting Classifier | ~96.5% |
| Alternating Least Square (ALS) | list of products liked, disliked and recommended by individual user with predicted rating |
| Natural Language Processing for Sentiment Analysis | Bag of Words<br>TF-IDF<br>Hashing |
| Recurrent Neural Network | Simple RNN ~94%<br>LSTM layer ~95.5%<br>TF ~93.29% |
| Stacked Ensemble model | Voting Classifier ~94% |

## V. CONCLUSIONS

By building this recommendation system, we are able to find products which are similar and can be recommended to a set of users who have similar buying patterns. Moreover, by using algorithms like content based filtering, we found similar reviews given by multiple users. After using various algorithms, we concluded that Gradient Boosting Classifier, Gaussian Naïve Bayes, RNN (Recurrent Neural Networks) are most suitable for accurately predictions. NLP using Bag of Words is very much suitable for our dataset and for performing the required sentiment analysis whereas ALS (Alternating Least Square) serves as a way to let us know products which are most likely to be recommended by the users. Linear Regression and SVM (Support Vector Machine) are not suitable for the same since accuracy is way too way for it to qualify in our project.

## VI. REFERENCES

[1]https://github.com/nikbearbrown/NEU_COE/blob/master/INFO_7390

[2]https://chrisalbon.com/machine_learning/trees_and_forests/random_forest_classifier_example/

[3]https://towardsdatascience.com/re commender-engine-under-the-hood-7869d5eab072

[4]https://machinelearningmastery.com/deep-learning-bag-of-words-model-sentiment-analysis/

[5]https://www.kdnuggets.com/2016/01/seven-steps-deep-learning.html/2

[6]https://www.ibm.com/developerworks/library/os-recommender1/

[7]https://www.analyticsindiamag.com/7-types-classification-algorithms/

[8]https://machinelearningmastery.com/natural-language-processing/

[9]https://medium.com/udacity/natural-language-processing-and-sentiment-analysis-43111c33c27e

[10]https://datascience.stackexchange.com/questions/22250/what-is-the-difference-between-a-hashing-vectorizer-and-a-tfidf-vectorizer

[11]https://www.infofarm.be/articles/alternating-least-squares-algorithm-recommenderlab

[12]https://blogs.sas.com/content/subconsciousmusings/2017/05/18/stacked-ensemble-models-win-data-science-competitions/

[13]https://jakevdp.github.io/PythonDataScienceHandbook/05.06-linear-regression.html

[15] https://rpubs.com/ferranmt/80166

[16]https://medium.com/learning-machine-learning/recommending-animes-using-nearest-neighbors-61320a1a5934

[17]https://datascienceplus.com/building-a-book-recommender-system-the-basics-knn-and-matrix-factorization/

[18]https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/

[19]https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/

[20]https://bugra.github.io/work/notes/2014-04-19/alternating-least-squares-method-for-collaborative-filtering/

[21]https://gist.github.com/victorkohler

[22]https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe

[23]https://github.com/Mona19/Recommendation-System-using-ALS-Algorithm-in-python/blob/master/als.ipynb

[24]http://dataaspirant.com/2017/02/20/gaussian-naive-bayes-classifier-implementation-python/

[25]https://blog.sicara.com/naive-bayes-classifier-sklearn-python-example-tips-42d100429e44

[26]https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/

[27]https://cambridgespark.com/content/tutorials/implementing-your-own-recommender-systems-in-Python/index.html

[28]http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html

[29]https://www.kaggle.com/cyruseption/iris-dataset-using-svm-and-linear-svm

[30]https://www.analyticsvidhya.com/blog/2016/06/quick-guide-build-recommendation-engine-python/

[31]https://conversionxl.com/blog/product-recommendations/

[32]http://www.data-mania.com/blog/recommendation-system-python/

[33]https://github.com/rish-16/Hybrid-Recommendation-System/blob/master/recommendations.py