CS145 - Data Mining Project: Yelp Rating Prediction

Group 29: The Classy Fires

Vaishnavi Ravindran — *UCLA*

Raja Mathanky Sankaranarayanan — *UCLA*

Yang Zhao — *UCLA*

Abstract

A few years ago, Yelp hosted a data mining competition on the data science problem platform Kaggle that challenged participants to use the provided user, business, and review data to predict, for a given user and business, the rating that user will give the business. In this project we are given a subset of the original Yelp data, and our goal is the same: when given a pair of (user, business), predict the rating that user will give to the business in a review. To solve this problem, we conducted experiments where we repeatedly extracted features from the data, selected a subset of the features to use, trained a model with the features, calculated RMSE to evaluate, and readjusted the features/models used. In the end, we found that a simple model, such as linear regression, works the best for this problem, and that using more features for the linear regression model gives more accurate predictions.

Introduction of the overall goal and background

Yelp is a popular review website for restaurants where users can look up reviews and ratings and also post reviews and ratings for restaurants that they have been to. Yelp itself stores various pieces of information about each restaurant and user, and it also stores each review's text along with the rating that the poster of the review gave the business. Using these pieces of stored information and reviews, it is possible to apply data mining and machine learning techniques to predict the rating that a certain user would give to a certain business. This problem can be seen as a multiclass classification problem. In approaching this problem and designing our experiments, we followed a typical workflow for solving data science problems. To prepare, we first wrote code to extract the features we thought were useful and cleaned them to replace null values and make the data useable. Then we ran through many iterations of testing, and for each iteration, we chose a set of features to use, chose a model to train on, tuned the model to get the best result possible, and evaluated the model using RMSE. Then when given a pair of (user, business), we will look through the provided training data files "users.csv", "business.csv", and "train_reviews.csv" to find supplemental features for the user and business provided, and then use those features as input for our trained model. To implement our models, we used the scikit-learn library, which automatically supports multiclass classification for all of the models we used.

Related Work

A large proportion of recommender systems used collaborative filtering using user-user collaboration and item-item collaboration.

1. Collaborative Filtering (CF)
   CF methods are based on the assumption that similar users prefer similar items or that a user expresses similar preferences for similar items. Instead of performing content indexing or content analysis, CF systems rely entirely on interest ratings from the members of a participating community. CF methods are categorized into two general classes, namely model-based and memory-based.
   Model-based algorithms use the underlying data to learn a probabilistic model, such as a cluster model or a Bayesian network model, using statistical and machine learning techniques. Subsequently, they use the model to make predictions.
   Memory-based methods, on the other hand, store the preference information in computer memory and use this information to find similarity between users or items to make predictions.
   a. User-User Collaborative Filtering
      User-based CF systems are systems that utilize memory-based algorithms, and operate over the entire user-item matrix to make predictions. The majority of such systems mainly deal with user-user similarity calculations, meaning that they utilize user neighborhoods, constructed as collections of similar users. In other words, they deal with the rows of the user-item matrix, in order to generate their

results. As users do not have to provide their opinion for all items,and the resulting user-item matrix may be a sparse matrix. This sparsity of the user-item matrix is the main reason causing filtering algorithms not to produce satisfactory results. One way to overcome this problem is through Default Voting, which is the simplest technique used to reduce sparsity. A default rating value is inserted to items that have not been rated by the user. This rating value is selected to be neutral or somewhat indicative of negative preferences for unseen items. The next step is the neighborhood formation, and this step is implemented in two parts: The similarity between all the users in the user-item matrix is calculated with the help of some proximity metrics, such as the Pearson Correlation Coefficient. The second step is the actual neighborhood generation for the active user, where the similarities of users are processed in order to select those users that will constitute the neighborhood of the current user. The similarity between two users, given the user-item matrix R, is :

$$sim(u_a, u_b) = \frac{\sum_{l=1}^{n} (R(u_a, i_l) - \overline{R}(u_a))(R(u_b, i_l) - \overline{R}(u_b))}{\sqrt{\sum_{l=1}^{n} (R(u_a, i_l) - \overline{R}(u_a))^2 \sum_{l=1}^{n} (R(u_b, i_l) - \overline{R}(u_b))^2}}.$$

b. Item-Item Collaborative Filtering Since the relationships between users are relatively dynamic, as they continuously buy new products or visit new places, it is computationally costly to calculate the user-to-user matrix. This causes the user-based CF approach to be relatively expensive in terms of computational load. In the item-based CF algorithm, we take a look into the set of items that the active user has rated and compute its similarity to the target item. Then, we select the k most similar items based on their corresponding similarities. The predictions can then be computed by taking a weighted average of the active user's ratings on these similar items. The main steps in this approach are the same as in user-based CF. The difference in the present approach is that instead of calculating similarities between two users who have provided ratings for a common item, we calculate similarities between two items which have been rated by a common user. Using Pearson's correlation as the similarity metric, the item-item similarity is given by :

$$sim(i_t, i_j) = \frac{\sum_{l=1}^{n} (R(u_l, i_t) - \overline{R}(i_t))(R(u_l, i_j) - \overline{R}(i_j))}{\sqrt{\sum_{l=1}^{n} (R(u_l, i_t) - \overline{R}(i_t))^2 \sum_{l=1}^{n} (R(u_l, i_j) - \overline{R}(i_j))^2}}$$

2. Content-based Filtering
All the information about a user, extracted either by monitoring user actions or by examining the objects the user has rated, is stored and utilized to customize the services offered to each user. This user modeling approach is known as content-based learning. The main assumption behind it is that a user's behavior remains unchanged through time. Therefore, the content of past user actions may be used to predict the desired content of their future actions. In content-based recommendation methods, the rating R(u,i) of the item i for the user u is typically estimated based on ratings assigned by user to the items that are "similar" to an item in terms of their content, as defined by their associated features

Problem definition and formalization:

The rating prediction problem can be thought of as a multiclass classification problem where the classes K = {1, 2, 3, 4, 5}. When we choose a model, we train it using features extracted from the data. By using the scikit-learn library, we can take advantage of its inherent multiclass classification features. For Linear Regression, the scikit-learn library implements it using the One vs. All strategy. The initial inputs for this problem are a pair of ids: (user_id, business_id). We then, through our code, use these ids to search our training data to find additional features for the business, user, and any reviews the user might have written.

(user_id, business_id) --- Processing to add populate features ---> (user_id, avg_stars, text ….) ----> M(user_id, avg_stars, text ….) ----> {1, 2, 3, 4, 5}

Data preparation description and preprocessing:

Data Cleaning:

1. The users and businesses training data had missing values for many attributes. This was approached by filling such empty values with one of the following three approaches:
    - Use the mean of the attribute's given values in the training data.
    - Use a Random value picked from the training data.
    - Simply assign the most common default value like 0.

See table 1. Which shows how missing values were assigned for a few attributes (examples.

2. The users and businesses training data had few garbage values in a few rows. (Eg> Row 10772 in users had the following value "CtuYzX06mHoWOPdhlf-wLA" for the attribute "average_stars" which does not make sense. We remove such rows from the training data.

3. Feature extraction and preprocessing:
    - **Ready-to-use:** For the features that already have a straightforward and useable range of values, we do not touch them. We only perform normalization for such features. For example, the attribute "average_stars" for users and "stars" for business have float values in the range [1,5]. The list of attributes that fell under this category were:
    - **Boolean Features:** For features whose values were "True" and "False", we assign values 1 and -1 respectively. Sometimes, features also have values like "True", "False", "None". In this scenario, we assign 1, -1, and 0 respectively. The list of attributes that fell under this category were:
    - **Categorical features:** For features with categorical values we assign integer values for each of the categories. For example, the feature "city" for business had 208 unique values for all the training data and we assign 208 integer value for each city. The list of attributes that fell under this category were:
    - **Get multiple features from one:** For features which have a nested set of values for each data point, that is, they have multiple values for the given attribute, we extract each such value as a separate feature. For example, the feature "attributes_BusinessParking**"** has sub-categories like "garage", "street", "validated", "lot", and "valet". As a result, we extracted, five corresponding features from this single feature and further performed data cleaning for each of the five features.
    - **One-hot-encoded:** For features described in "*Get multiple features from one",* it sometimes made more sense and improved the overall accuracy of the algorithms to keep these feature values intact. To do this, we performed one-hot-encoding. For example, for the feature "attributes_Ambience" there are 9 sub-categories namely: 'romantic', 'intimate', 'classy', 'hipster','divey', 'touristy', 'trendy', 'upscale', 'casual'. For this feature, we tried both "*Get multiple features from one"* and "*One-hot-encoded"* and found that it was better for accuracy to use "*One-hot-encoded".*
    - **Time related features:** For the feature, "yelping_since" we extracted is an object of type time and subtracted from the current date, to get the duration. For the features such as "hours_Friday", "hours_Monday", etc., we use them as string values.
    - **Text features:** The review text was extracted and encoded with a bag-of-words model and then we used TruncatedSVD to reduce the dimension from about 100000 to 50.

- **Features that were ignored:** There are some features that do not add any value to any of our methods used either due to their sparsity(missing values) or because of the significance of their values. The feature "name" under users is one such feature that will not have any correlation to the ratings. Similarly , we also dropped the attributes "attributes_HairSpecializesIn" because it has 99% missing values. We also dropped "hours" as it has 100% missing values.
- **List of features extracted for businesses :**
  address, attributes_AcceptsInsurance ,
  attributes_AgesAllowed attributes_Alcohol
  attributes_Ambience
  attributes_BYOB
  attributes_BYOBCorkage
  attributes_BestNights
  attributes_BikeParking
  attributes_BusinessAcceptsBitcoin
  attributes_BusinessAcceptsCreditCards
  attributes_BusinessParking
  attributes_ByAppointmentOnly
  attributes_Caters
  attributes_CoatCheck
  attributes_Corkage
  attributes_DietaryRestrictions
  attributes_DogsAllowed
  attributes_DriveThru
  attributes_GoodForDancing
  attributes_GoodForKids
  attributes_GoodForMeal
  attributes_HappyHour
  attributes_HasTV
  attributes_Music
  attributes_NoiseLevel
  attributes_Open24Hours
  attributes_OutdoorSeating
  attributes_RestaurantsAttire
  attributes_RestaurantsCounterService
  attributes_RestaurantsDelivery
  attributes_RestaurantsGoodForGroups
  attributes_RestaurantsPriceRange2
  attributes_RestaurantsReservations
  attributes_RestaurantsTableService
  attributes_RestaurantsTakeOut
  attributes_Smoking
  attributes_WheelchairAccessible
  attributes_WiFi business_id
  categories
  city
  hours_Friday
  hours_Monday
  hours_Saturday hours_Sunday
  hours_Thursday
  hours_Tuesday
  hours_Wednesday
  is_open
  latitude
  longitude
  name
  neighborhood
  postal_code
  review_count
  stars
  state
- **List of features extracted for users:**
  average_stars
  compliment_cool
  compliment_cute
  compliment_funny
  compliment_hot
  Compliment_list
  Compliment_more
  Compliment_note
  Compliment_photos
  Compliment_plain
  Compliment_profile
  Compliment_writer
  Cool
  elite
  Fans
  Friends
  Funny
  Name
  Review_count
  Useful
  User_id
  yelping_since
- In total, we extracted 90 features from both users and businesses.

4. Normalization: Since each feature can have a very different range of values, we performed normalization for each feature individually using the formula :

Methods description

1. **Mean predictors:** Mean predictors is the simplest way of predicting. However, for a large amount of training data, it performs well and gives results comparable to other machine learning algorithms. Given a pair of user i and business we predict the rating this user will give the business by using the following formula:

Rating = $\mu + u_i + b_j$
Where, μ is the average of all the ratings given to us in train_reviews.csv and and u is a vector containing the average of ratings of all users which calculated using the feature "average_stars" present in users.csv. Similarly, b is a vector containing the average of ratings of all businesses which are calculated using the feature "stars" present in business.csv.

In this method, we would have to deal with the cold start problem. That is, what if we do not know the average rating for a user beforehand? In this case, we simply assigned a value of 0 to such a user. The same holds for business.

The other challenge in this method is what are known as "sand-bag ratings". Consider the scenario where for a given user, most of the ratings are in the range of 4-5 but there are few ratings of 1 and 2. Such ratings will affect the mean of the user rating and we observed that it negatively affected the accuracy of our results. The approach we took to deal with this problem was to filter out all those ratings which are not within three standard deviations of the mean for a given user and a given business. This improved the overall accuracy.

<u>2.</u> **<u>Collaborative filtering:</u>** The way collaborative filtering works is, if a user 'i' has given a high rating for businesses A, B,C and if another user 'j' has given a similar such high rating for businesses B,C,D, then we can say that the two users have similar interests. We can also conclude (with some certainty that user i would give a high rating to D and user j would give a high rating to C.
This is "user-user" collaborative filtering, which is a very common used algorithm in the industry.

For the yelp dataset, instead of using "user-user" collaborative filtering, we used "business-business" collaborative filtering.
Training step: Here, we find the similarity between each business pair in the training data. This can be considered as the training step.

Prediction : For a given pair of user i and business j in the test data, we find all the businesses rated by user i. We then calculate the similarity between each of these businesses and the given bussiness j. We then set a threshold to find the most similar businesses. Based on the ratings that these similar businesses have received from the user i , we will predict the rating for business j.

The pairwise similarity between the businesses was calculated using cosine similarity using the eqn:

Here, sim(i,j is the similarity between a pair of businesses i and j.
The threshold for finding the most similar business for a given business 'b' was done using the eqn :

$$t = \sum_i S_{b,i}/N$$

where $S_{b,I}$ is the cosine similarity value between business b and every other business i in the training data.
N is the total number of businesses i in the training data.

The following formula was used predict:
Here, $P_{u,i}$ is the prediction given by user u for business i.
$S_{i,n}$ is the similarity between business i and business N.
$R_{u,n}$ is the rating given by user u to the business n.
N is the number of businesses similar to business i.

Challenges: We faced what is known as the cold start problem here. That is, in our test data set, we can get a pair of user and business such that either this user is not present in the training data or this business is not present in the training data or both are not present in the training data. We will call the two types of missing scenarios as:
1. User cold start
2. Business cold start

1. In the case of user cold start, we do not have any history of this user. So after calculating the most similar businesses to given business in our test pair of (user, business, we just find the average of the ratings of all these similar businesses. This average value will be our prediction.
2. In the case of business cold start, we do not have any prior knowledge about this new business we encounter in the test pair of (user, business, as a result we cannot find the most similar business for it. Instead, we find the average of the ratings given by this user in the test pair of (user, business, to every business he or she has rated. This average value will be our prediction.

3. Hybrid Recommender System: The hybrid recommender system is an amalgamation of the content-based filtering system and the collaborative filtering system.

Content-based filtering, also referred to as cognitive filtering, recommends items based on a comparison between the content, or description of items. The text of the description of the items are compared against each other using various text-similarity metrics, such as cosine similarity and jaccard similarity. Using content-based filtering as a stand-alone method for building a recommender system would not work very well, due to the following reasons :

1. A content-based filtering system will not select items if the system does not know the user's preferences, which is usually taken in the form of a survey everytime a new user appears. Additional techniques have to be added to give the system the capability to make suggestion outside the scope of what the user has already shown interest in.
2. The content-based recommender system uses the 'bag-of-words' approach to identify the similarity between the texts. This fails to identify sarcasm, humour and points of view.

In order to overcome these challenges, a combination of the content-based filtering system and a collaborative filtering system is used to build a Hybrid Recommender System. Through collaboration via content, both - the rated items and the content of the items are used to construct a user profile. The selection of terms which describe the content of the items is done using content-based techniques. The weight of terms indicate how important they are to the user. Predictions are made based on a weighted average of the content-based recommendation and the collaborative recommendation. The Pearson correlation coefficient can be used to compute the correlation between users. Instead of determining the correlation with user ratings, term weights are used. This method has a greater number of items to determine similarity than collaborative filtering, the problem of users not having enough rated items in common is not an issue anymore. Furthermore, unlike content-based filtering, predictions are based on the impressions of other users which could lead to recommendations outside the normal environment of a user.

The following formulas represent Cosine Similarity, Jaccard Similarity and Pearson's Correlation:

Cosine Similarity : The cosine similarity between two documents, each represented by a vector of number of words of type w in the vocabulary, is given by :

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

Jaccard Similarity : The Jaccard similarity index between two documents A and B is given by :

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Pearson's Correlation Coefficient :

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

In our implementation of the Hybrid recommender, the content-based recommender uses the "categories" attribute of each business, in order to find similar restaurants. Collaborative filtering was implemented just as in method 2.

4. Linear regression: The ordinary least squares regression was used. This method gives us the best results.

5. Neural network: A multilayer perceptron was used with hidden_layer_sizes = (number of features , number of features , number of features) and number of iteration = 100.

6. Random Forest Classifier: A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size.

7. Decision Tree Classifier: Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
We used the gini index for measuring the impurity.

8. Logistic Regression: We used regularized logistic regression with cross- entropy loss.

9. KNeighborsClassifier: A simple majority vote is computed from the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

10. RidgeClassifier: Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of coefficients. The ridge coefficients minimize a penalized residual sum of squares.

11. Naive Bayes classifier for multivariate Bernoulli models: Implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions.

Experiments design and Evaluation:

Experiments design

1. Data cleaning and preprocessing
2. Feature extraction
3. For each of the methods described in, we trained using the training data given in "train_reviews.csv"
4. For comparing and evaluating the models, we used the validation data set given as "validate_queries.csv. For the purpose of validation we used Mean Square Error as the metric given by the equation:
5. For the methods linear regression, LinearSVC, Random Forest Classifier, Decision Tree Classifier, Naive Bayes classifier for multivariate Bernoulli models, LogisticRegression, KNeighborsClassifier, RidgeClassifier and neural network. We had to find the optimal set of features that gives high accuracy. To do this, we performed k-fold cross validation, to find the "correct" or the most "optimal" combination of features for each algorithm. The steps involved in this were :
    - Random choose of i features from the feature pool, where:
      $i = \{0, N-1\}$
      N = Total number of features extracted.
    - We always used the features "stars" and "average_stars" from businesses and users respectively. Therefore,
      feature set to train = "stars" +"average_stars" + randomly chosen i features
    - For each value of i, we sampled features 20 times and performed k = 2 fold cross validation on each iteration.

Evaluation and results :

1. After comparing all the algorithms detailed in the "methods" section by their accuracies using MSE on the validation data set, we found that Linear Regression gave the best results. The mse was 1.07310 (kaggle score) and our position is 13 on the leaderboard (current position). Since we tried many combinations of features, the combination of features that led to the best accuracy are shown in Table 1.
2. The accuracies of Linear regression with few other combinations were also close to our best score and we have tabulated these results as well in Table 1.
3. After Linear regression, surprisingly, mean predictors was our next best method with an MSE of 1.12380.
4. The next best algorithm was collaborative filtering using business-bussiness similarity which gave us an mse of 1.14979.
5. For many of the methods we tried, the accuracy decreased with an increase in too many features From the table, we can see that this was the case for KNN, neural networks, Random forest, Decision Trees.
6. It was also interesting to note that, even some sparse features that did not have 100% values, led to the final best accuracy. Examples of such features are: attributes_byob and attributes_bestnights. This tells us, that the presence of values for such features, perhaps makes a difference to the classification.
7. In the table we report upto two best accuracies for each algorithm along with the features used for that method. (Since each algorithm was tried with many combinations of features).

Table 1: Results of algorithms tried along with the features used for each and the MSE on the validation data set.

Method	Features	MSE on validation data set
Linear Regression	lunch hours_Sunday attributes_BestNights neighborhood attributes_ByAppointmentOnly city attributes_DogsAllowed latenight attributes_Music compliment_list attributes_BYOB categories classy attributes_RestaurantsPriceRange2 attributes_Alcohol attributes_RestaurantsAttire attributes_WiFi attributes_DogsAllowed attributes_BikeParking attributes_OutdoorSeating garage_BP latenight lunch dinner brunch romantic attributes_DriveThru attributes_ByAppointmentOnly attributes_BusinessAcceptsBitcoin attributes_AcceptsInsurance neighborhood intimate Classy upscale attributes_Caters attributes_HasTV city is_open hours_Sunday hours_Tuesday hours_Wednesday hours_Friday yelping_since elite compliment_writer compliment_plain compliment_more compliment_list compliment_hot compliment_funny attributes_RestaurantsCounterService attributes_HappyHour attributes_GoodForDancing categories attributes_Music attributes_DietaryRestrictions attributes_BestNights attributes_AgesAllowed	1.0688977837932432
Mean predictor	"stars" and "average_stars"	1.12380
Collaborative filtering	"stars" and "average_stars"	1.14979
Random Forest	User stras, Business Stars, review count, attributes_BusinessAcceptsCreditCards, attributes_BusinessParking	1.4749008112926354
Random Forest	Stars of users and business	1.254366961484695
Random Forest	With 31 features	1.5069193658086835

Neural network	User stras, Business Stars, review count, Useful_user, attributes_BusinessAcceptsCreditCards attributes_RestaurantsPriceRange2 attributes_WiFi	1.2384645010738997
Neural network	All 90 features	1.5899710825469506
KNN	Review Text, Business ID, Business Stars, User Stars	1.35169755756
KNN	With 45 features	1.443636272107959
Decision Tree Classifier	Stars of users and business	1.2177554888929158
Decision Tree Classifier	Stars of users and business, Validated_parking, brunch, attributes_DriveThru	1.5814113272481594
Naive Bayes classifier for multivariate Bernoulli models	Stars of users and business, valet_BP compliment_plain fans attributes_AgesAllowed	1.4241535136586172
Naive Bayes classifier for multivariate Bernoulli models	Stars of users and business,compliment_more street_BP attributes_WiFi	1.4313607161270359
Logistic Regression	Stars of users and business,attributes_DriveThru compliment_cute	1.5040367546779543
Ridge Classifier	Stars of users and business,attributes_HasTV, attributes_RestaurantsTakeOut	1.4560197230847591
Ridge Classifier	With 45 features	1.263762062347897
Hybrid Classifier	User_id, Business_id, Ratings given to businesses by users (stars), Business Categories.	1.789789063426908

Conclusion

After extracting the features and performing the required data cleaning on them, we tried 12 methods/algorithms for our dataset. Out of these 12 data methods, we saw that Linear Regression performed the best giving us an overall accuracy of 1.0688977837932432 and placing us 13th on the leaderboard. Unlike other algorithms we tried, for linear regression, an increase in the number of features actually led to an increase in accuracy with a total of 47 features giving the best accuracy. Using k-fold cross-validation, we found the best set of features for each algorithm. What was interesting to note was that for this given dataset, the mean predictor method also performed quite well, being the 2nd best method for us followed by collaborative filtering and the decision tree classifier.

References:

[1]Recommender Systems — User-Based and Item-Based Collaborative Filtering: 2018. *https://medium.com/@cfpinela/recommender-systems-user-based-and-item-based-collaborative-filtering-5d5f375a127f*. Accessed: 2018- 12- 11.

[2]Faridani, V., Jalali, M. and Jahan, M. 2017. Collaborative filtering-based recommender systems by effective trust. *International Journal of Data Science and Analytics*. 3, 4 (2017), 297-307.

[3]Adomavicius, G. and Tuzhilin, A. 2005. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*. 17, 6 (2005), 734-749.

[4]Chen, L., Chen, G. and Wang, F. 2015. Recommender systems based on user reviews: the state of the art. *User Modeling and User-Adapted Interaction*. 25, 2 (2015), 99-154.

[5]Hussein, T., Linder, T., Gaulke, W. and Ziegler, J. 2012. Hybreed: A software framework for developing context-aware hybrid recommender systems. *User Modeling and User-Adapted Interaction*. 24, 1-2 (2012), 121-174.

Task	People
Feature extraction of 89 features with data cleaning. Implemented Collaborative filtering, Mean predictor. linear regression, LinearSVC, Random Forest Classifier, Decision Tree Classifier, Naive Bayes classifier for multivariate Bernoulli models, LogisticRegression, KNeighborsClassifier, RidgeClassifier and neural network Evaluation of above methods using K-fold cross validation Report writing	Vaishnavi Ravindran
Feature extraction and data cleaning. Implemented collaborative filtering, content-based learning and hybrid recommender system using Cosine similarity, Jaccard similarity and Pearson's correlation coefficient. Performed k-fold cross validation to evaluate all of the above methods. Report Writing	Raja Mathanky Sankaranarayanan
Feature extraction and data cleaning for review text, and features from Users and Businesses. Condensed review text features to reduce dimensionality. Tested features and tuned hyperparameters for Linear Regression, Linear Regression, KNN, Linear SVM Report Writing	Yang Zhao