Yelp Rating Predictor Final Report

Group ID: 1
Group Name: Miners for Datas

Rajiv Anisetti
UID: 904801422
rajiv.anisetti@gmail.com

Ken Gu
UID: 904836308
ken.qgu@gmail.com

Andrew Ding
UID: 504748356
andrewxding@ucla.edu

Caleb Chau
UID: 204805602
caleb.h.chau@gmail.com

Yuci Shen
UID: 604836772
shen.yuci11@gmail.com

ABSTRACT

This paper details the methodologies as well as analysis of several modeling attempts towards deriving a prediction model for Yelp ratings based on a pool of prior database knowledge. Data is preprocessed to optimize usability, then used to train multiple models such as Kth Nearest Neighbor (KNN), Neural Networks (MLP Regressor/Classifier), and Random Forest.

KEYWORDS

KNN, Neural Network, Random Forest, MLP Classifier, MLP Regressor, Yelp, Rating, Review

1 INTRODUCTION

With the advent of the internet, Yelp reviews have become a major benchmark of customer satisfaction for businesses and are an excellent source of consumer and business data.

In this project, we aim to identify key customer and business features that influence user rating and use this information to train fitting models that predict new user ratings for a given business. Using these models, we will be able to predict a new customer's rating for a business based on characteristics like a business's noise level and ambiance and a customer's average rating and review history.

2 RELATED WORK

What we did in this project is part of a larger pre-existing algorithm field called collaborative filtering. Asides from Yelp, there are many other recommendation systems in the world around us. Netflix recommends shows, Amazon recommends purchases, and Facebook even recommends friends. These recommendation systems are all tied together by collaborative filtering models, which are models based on an assumption that people like things that 1) are similar to other things they like and 2) liked by other people with similar taste.

There are two major approaches to collaborative filtering: memory-based, and model based. Model-based collaborative filtering is what we implemented in this project, where machine learning algorithms are used to develop fitting models that predict a user's rating of a new item. Memory-based collaborative filtering, however, is slightly different, and is split into two major types: user-item filtering and item-item filtering. User-item filtering takes a user, finds similar users, and then creates recommendations based off of items the similar users liked. Item-item filtering takes an item, finds users who liked that item, and compiles a list of items that those users also liked to output as recommendations.

Asides from economic gain, collaborative filtering also has potential to be an extremely powerful political tool. Especially with the advent of social media, we now have easier than ever access to users' ideological stances on various political issues. We can thus apply collaborative filtering to this situation, where the item is an ideological stance that can be used to connect the user with similar-minded users or political platforms. The efficacy of this technique was demonstrated in a social informatics study performed a few years ago, where it was shown that we can identify a user's political party with an 89% accuracy with a model trained off of ideological stance data.

3 PROBLEM DEFINITION

Before we can derive our prediction model, we must first amass a database of user, business, and review information – this has already been provided to us in the form of .csv files. After we obtain the data, we must preprocess it to eliminate irrelevant features that have no bearing on user rating and convert the remaining information into a format that can be analyzed by our models.

Our approach will consist of the following steps:

1. Reading in the data from the CVS datasheets
2. Preprocessing the data to eliminate irrelevant features, and convert the rest into information our fitting models can read
3. Designing and training fitting models with a training subset to predict user rating based on user/business properties.
4. Running each fitting model on a test subset to obtain predicted ratings for each user/business pair.

5. Analyzing the efficacy of each training model by using the model to predict the ratings of the verification dataset, then calculating the Least Means Squared Error between the predicted ratings and the actual ratings.

4 DATA PREPROCESSING

The raw data in the CSV datasheet must be preprocessed before the information can be analyzed by our fitting model. Parameter values must be converted where necessary to a data type "legible" to our models (numerical); parameters that do not intuitively relate to a user's review are removed to reduce the amount of noise in our dataset.

4.1 Business Data Preprocessing

The first task at hand is to identify useful features, and then convert the values of said useful features into numerical data.

After expanding all dictionary strings into extra columns, with each column being a key of the dictionary and each row content being the value, we found that businesses on Yelp are described by over 60 features. Thus, to ensure efficiency and reduce noise it is necessary to discard features that either have little to no effect on the user rating or are inappropriate to be used in our fitting models. The following features were removed:

1. *hours*, *latitude*, *longitude*, *name*, and *address* were all too complex to convert into numerical values. Furthermore, based on our domain knowledge, factors like a business' longitude have very little effect on a user's review.
2. All features where more than 50% of its values are NaN – such features have too few datapoints with meaningful values and would not add much to the model. The percentage, however, can be modified as seen fit.

This feature selection left us with the following features:

1. *business_id* – used to identify the business
2. Categorical
    a. *attributes_Alcohol*
        i. Represents whether a business serves alcohol.
        ii. Numerical – ranges from 0 to 1
    b. *attributes_RestaurantsPriceRange2*
        i. Represents the number of dollar signs a business has.
        ii. Numerical – ranges from 1 to 4
3. Numerical
    a. *review_count* – total number of reviews
    b. *stars* – average number of stars reviewers give

To verify that this feature set was an optimal, we re-processed our data in several other ways to factor in features such as business hours and name, and amenities/ambiance, which some may argue may influence user rating. After training and testing our models with this data, however, we discovered that the addition of these features feature increased the MSE of the predictions, leading us to the conclusion that the above selection of features is optimal.

4.2 User Data Preprocessing

Once again, features that do not intuitively have a huge influence on user reviews are removed from the feature pool. For user data, the features that are removed are:

elite, *friends*, *name*, and *yelping_since* – all four values are mostly personal attributes, and reflect little on the user's personal preferences.

This feature selection left us with the following features:

1. *user_id* – used to identify the user
2. Numerical
    a. *average_stars* – average star rating per review
    b. *review_count* – total number of reviews given
    c. *useful* – total number of "useful" upvotes given to this use's reviews

4.3 Review Data Preprocessing

This set of data will remain largely unchanged, with the following features:

user_id, *business_id*, and *stars*

The purpose of this dataset is to let us join our business and user datasets together using the user/business ids as a reference. This forms the basis of our training dataset.

4.4 Formatting Training Data

To form our training dataset train_df_x, join our our user data with our business data by using the *user_id* and *business*_id relationships detailed in the Review data. This results in a set of data with the following features for each review:

average_stars, *review_count_x*, *useful*, *attributes_alcohol*, *attributes_RestaurantsPriceRange2*, *review_count_y*, and *business_stars*.

To form our training dataset train_df_y, the label for each datapoint, we simply record the stars for each review, resulting in a set of data with one column of data: *stars*

4.5 Normalizing the Data

Finally, the usefulness score should be normalized to properly reflect the true usefulness of a user. This is necessary because the usefulness score reflects the number of upvotes a user's reviews received, rather not the average quality of the reviews. Under this definition, a user with 100 reviews each with 1 *useful* upvote would have the same usefulness as a user with 10 review each with 10 *useful* upvotes.

To normalize this feature, we use the formula:

$$useful = useful/review\_count\_x$$

5 METHODS DESCRIPTION

To classify our data, we need a model that takes in several inputs, and outputs a value from 1 to 5, representing 1-star to 5-star ratings. We formulated three approaches for designing our model.

5.1 K-Nearest Neighbor

To set up this model, we first map a set of training data to a feature space – each datapoint point in the space is defined by the

7 parameters in the training dataset and labeled according to review rating associated with the 7 parameters.

To classify an unlabeled datapoint, we place it in the feature space, find the K nearest neighboring training data points using the Euclidian distance function, and then use a majority vote to classify the unlabeled datapoint and output a value from 1 to 5.

Pros: Easy to implement, and very fast training time
Cons: Can become skewed if define our feature space with irrelevant features; it can be difficult to select features properly.

5.2 Random Forest

The basic unit of a random forest is a decision tree, which uses the entire feature set when creating branches, and has access to the whole training data set.

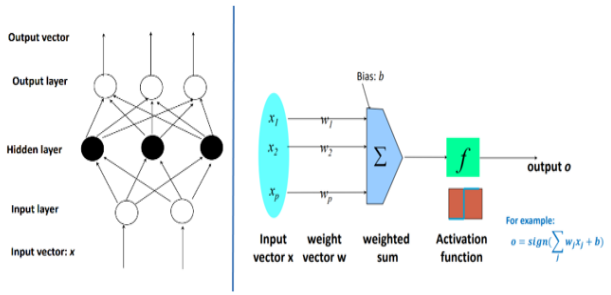
To set up a random forest, we create a predetermined number of decision trees. Each tree only has access to a random subset of training data points, and only considers a random subset of features when creating branches and deepening the tree.

For example, a decision tree in our random forest model may only consider *attributes_alcohol*, *review_count*, and *attributes_RestaurantsPriceRange2* when creating a branch, rather than all features; the features are a random subset of all features at each branching event. Furthermore, this decision tree would only have a subset of the training data to work with.

To classify an unlabeled data point, we run its parameters through all the decision trees in the random forest and then take a majority vote to decide which classification to label the point with. This model is superior to the regular decision tree, as the plurality of decision trees in the random forest eliminates the overfitting issue.

5.3 Neural Network

A neural network structure consists of an input layer, that feeds into a series of hidden layers, the last of which feeds into the output layer. The output of each layer is determined by the values fed in from the previous layer adjusted with a bias, then run through a predetermined activation function. Below is a diagram that illustrates such a neural network.



A neural network is trained through repetition of forward and back propagation:

1. The values of each layer are populated through a forward propagation.
2. The errors of each layer after the input later are calculated through back propagation and comparison with the ground truth output.
3. The biases are updated based off the errors, and we go back to step 1.

The above backpropagation is called "supervised learning", and is repeated until the error of the output nodes are within satisfactory limits, at which point the network is considered trained.

5.3.1 MLP Classifier/Regressor

MLP stands for multilayer perceptron, and it is a neural network that consists of an input layer, one or more hidden layers, and an output layer non-linear activation functions on their nodes. This kind of neural network is very useful, as non-linear activation functions allow us to deal with data that is not linearly separable, for example our Yelp data.

To set up the MLP model, set the input as the 7 parameters defined in the training dataset, and take the output as the predicted rating the user will give the business.

- If we are using an MLP Classifier, the range of the output layer will be discrete, with 5 outputs: [1, 2, 3, 4, 5]. This model will essentially classify each datapoint as one of 5 classes, each of which represent a different rating score.
- If we are using an MLP Regressor, the range of the output layer will be continuous, ranging from 0 to 5.0 – this output rounded to the nearest whole number will be out predicted rating for the datapoint.

6 EXPERIMENT DESIGN AND EVALUATION

6.1 Experiment Design

We implemented KNN, Random Forest, MLP Regressor, and MLP Classifier on Jupyter Notebook, in a Windows environment using machine learning functions taken from the *scikit-learn* library.

6.11 K-Nearest Neighbor

To implement KNN, we used the KNeighborsClassifier function from the scikit-learn library. We set the weight of each point to be measured by distance, used the "auto" option for the algorithm, and set k = [15, 25, 40, 55, 80, 125, 250, 350, 500, and 1000].

6.12 Random Forest

To implement Random Forest, we used the RandomForestClassifier function from the scikit-learn library. Each random forest had 150 decision trees, and a minimum split threshold of 5 datapoints. To find the optimal parameters for our model, we varied the depth of each decision tree and tested three depths: [7, 10, 15]

6.13 MLP Classifier

To implement the MLP Classifier, we used the MLPClassifier function from the scikit-learn library. We identified the optimal parameters for our fitting model by testing three max_iter values:

[200, 500, 700] and three alpha values: [1e-4, 1e-3, 1e-2] using a for loop to run in total nine variations of the MLP Classifier model.

6.14 MLP Regressor

To implement the MLP Regressor, we used the MLPRegressor function from the scikit-learn library. We identified the optimal parameters for our fitting model by testing three max_iter values: [200, 500, 700], three alpha values: [1e-4, 1e-3, 1e-2], and two hidden layer sizes (total # of hidden layers): [25, 50].

6.2 Evaluation

Following the experimental designs above, we identified the best parameters for each model, based on lowest MSE:

1. KNN:
    a. n_neighbors = 125
    b. algorithm: 'auto', weights: 'distance'
2. Random Forest:
    a. max_depth = 10
    b. min_samples_split = 5
    c. n_estimators = 150
3. MLP Classifier:
    a. max_iter = 500, alpha = 2e-4
    b. hidden_layer_sizes = (100, 100, 8)
    c. learning rate: adaptive, initially 0.001
    d. solver: 'sgd'
4. MLP Regressor:
    a. max_iter = 500, alpha = 2e-4, hidden_layer_sizes = (50,50,8)
    b. learning rate: adaptive, initially 0.001
    c. solver: 'adam'

Using these parameters, we also found the training and test accuracy of all four of these models, listed in the table below:

	Training Report		Validation Report	
	MSE	Accuracy	MSE	Accuracy
KNN	0.0109	0.9963	1.3737	0.4638
Random Forest	1.1327	0.5786	1.3418	0.4511
MLP-C	1.1091	0.3974	1.2712	0.3349
MLP-R	1.0886	0.4053	1.2181	0.3502

Based on these results, the best fitting model for our Yelp dataset is the MLP Regressor.

7 CONCLUSIONS

The goal of this project was to develop and train fitting model that could accurately predict a Yelp user's review of a business based off user history and business features. With this model, we would be able to feed in the features of a large group of users and a single business to find ratings a business would receive if they were to suddenly advertise and attract the patronage of said user group; the vice versa could also be applied to predict how a single user would react to a group of new businesses. These examples succinctly demonstrate the power of such modeling techniques; a accurate model could be used to help a business to boost its rating by handpicking target markets, or help a user find the perfect travel location with the most ideal businesses.

We were able to create a prediction model, but the accuracy and LMSE was far from ideal. All four models we tried had error in the range of 1.2 to 1.4 and accuracy in the range of 35% to 47% - both of which would be unacceptable in practical application. One reason for this was that we lacked the proper tools to select the features with which to train the model. While we were able to verify that our selected feature set was a good feature set using our domain knowledge (intuition) and simple verification, we likely did not find the optimal set of features. For example, there is likely a relationship between ambiance and user rating, but our model was not able to accommodate such data as there were too many NaN values. In addition, we lacked a modeling technique that could make full use of all the features, such as business hours, would have likely also improved prediction accuracy.

If we were able to carry out this project again in the future, we would thus like to spend more time on algorithms that determine feature usefulness to curate a more complete set of features. In addition, we could either find a new modeling technique, or combine pre-existing modeling techniques such that more features would be viable.

8 TEAM INFORMATION

Task	Members
Preprocessing Data	Ken Gu / Andrew Ding
Algorithm 1: KNN	Caleb Chau
Algorithm 2: Random Forest	Rajiv Anisetti
Algorithm 3: MLP-C	Andrew Ding
Algorithm 4: MLP-R	Ken Gu
Evaluation	Yuci Shen / Rajiv Anisetti
Report	Yuci Shen / Caleb Chau

Ranking: 19
Score: 1.09852
Entries: 16

References

[1] Prince Grover. 2017. Various Implementations of Collaborative Filtering – Towards Data Science. (December 2017). Retrieved December 12, 2018 from https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

[2] Sebastian Ruder. 2018. An overview of gradient descent optimization algorithms. (November 2018). Retrieved December 12, 2018 from http://ruder.io/optimizing-gradient-descent/index.html#adam

[3] Swapna Gottipati, Minghui Qiu, Liu Yang, Feida Zhu, and Jing Jiang. 2013. Predicting User's Political Party Using Ideological Stances. (November 2013). Retrieved December 12, 2018 from https://link.springer.com/chapter/10.1007/978-3-319-03260-3_16

[4] Anon. sklearn.neural_network.MLPClassifier¶. Retrieved December 12, 2018 from https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[5] Anon. sklearn.neural_network.MLPRegressor¶. Retrieved December 12, 2018 from https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

[6] Anon. sklearn.neighbors.KNeighborsClassifier¶. Retrieved December 12, 2018 from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[7] Anon. 3.2.4.3.1. sklearn.ensemble.RandomForestClassifier¶. Retrieved December 12, 2018 from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html