CSE 255 - Project 1
User Score for Restaurants Recommendation System using Google Dataset

Diego Cedillo
A53054538
dcedillo@eng.ucsd.edu

Idan Izhaki
A53044317
iizhaki@eng.ucsd.edu

ABSTRACT

This project report presents a suggestion of a rating predictor a user will give to a given restaurant considering the previews reviews made by the user, the type of food of the restaurant sells, the location of the restaurant, and reviews given by other people to the restaurant. For this purpose we used a Google dataset containing information about users, locations and reviews. We pre-processed this dataset to filter only the restaurants located in the US area, we clustered the restaurant locations into a smaller subset using k-means and we tuned the categories names to fit better our purpose. Finally, we used linear regression least-squares, stochastic gradient descent and l-bfgs algorithms to train over the training data, and predict the testing data. We eventually obtained a mean square error of 0.61 for lbfgs and 0.591 for SGD.

Keywords

l-bfgs, rating prediction, reviews, stochastic gradient, linear regreation, k-means

1. INTRODUCTION

Nowadays, in the Big Data era, people are connected to the world through the Internet all the time. People post pictures on the social networks every second, share the place they are visiting, the food they are eating; but more importantly, people are also giving feedback to the places they visit. Facebook, Google Maps (Figure 1), Yelp, foursquare and any other service that provides information about places on the globe has a rating scale to indicate if a place is worth visiting or not.

People are rating restaurants, bars, movie theaters, etc. in an increasing rate. People like to emphasize and share how much they enjoyed a place and, with enough motivation, to spread the word when they end up in a place they hated. These feedbacks can be expressed as comments or reviews, together with a rating of the places - usually represented on a star scale from 1 to 5.

For this project, we have decided to use the massive dataset from Google that contains information about places around the world, users with accounts in Google services and reviews that users have given to these places. Considering that the ratings of a place greatly depends on the category of the place (i. e. “Turistic”, “Restaurant”, etc), we decided to focus only on ”Restaurant” categories. Moreover, considering the amount of data presented, we take as an example for our prediction only the data that corresponds to places with GPS coordinates in the United States.

Considering the reviews given to restaurants in the US, we create a feature matrix that will help us to predict the rating that a user will give to a restaurant, considering the previous reviews that he gave to other restaurants, the categories that the restaurant defines (as type of food and ethnicity), the location of the restaurant, and the average rating given by other users. Once we collect the feature vectors and we train the data, we try our predictions on a subset of the data - known as a ”testing set”. For this data, we found a Mean Square Error (MSE) of 0.827 when using l-bfgs, sgd and linear regression as the learning algorithms.

We present statistics about the data and how we filter the data in section 2. We describe the feature matrix that we use on section 3. The methods used for learning are described on section 4. Section 5 provides the results of our prediction using different algorithms and feature matrices. Finally, we conclude our findings on section 6.

2. THE DATASET

The dataset used for this project is Google Location & Reviews information. This dataset contains information about 3.7 million users, 3 million places and 11 million reviews that users gave to those locations. Each user’s information entry is composed of a name, current place (city and GPS coordinates), level of education, jobs held, and previous places visited. Similarly, each place entry is composed of the name of the place, hours they open, phone number, address, and

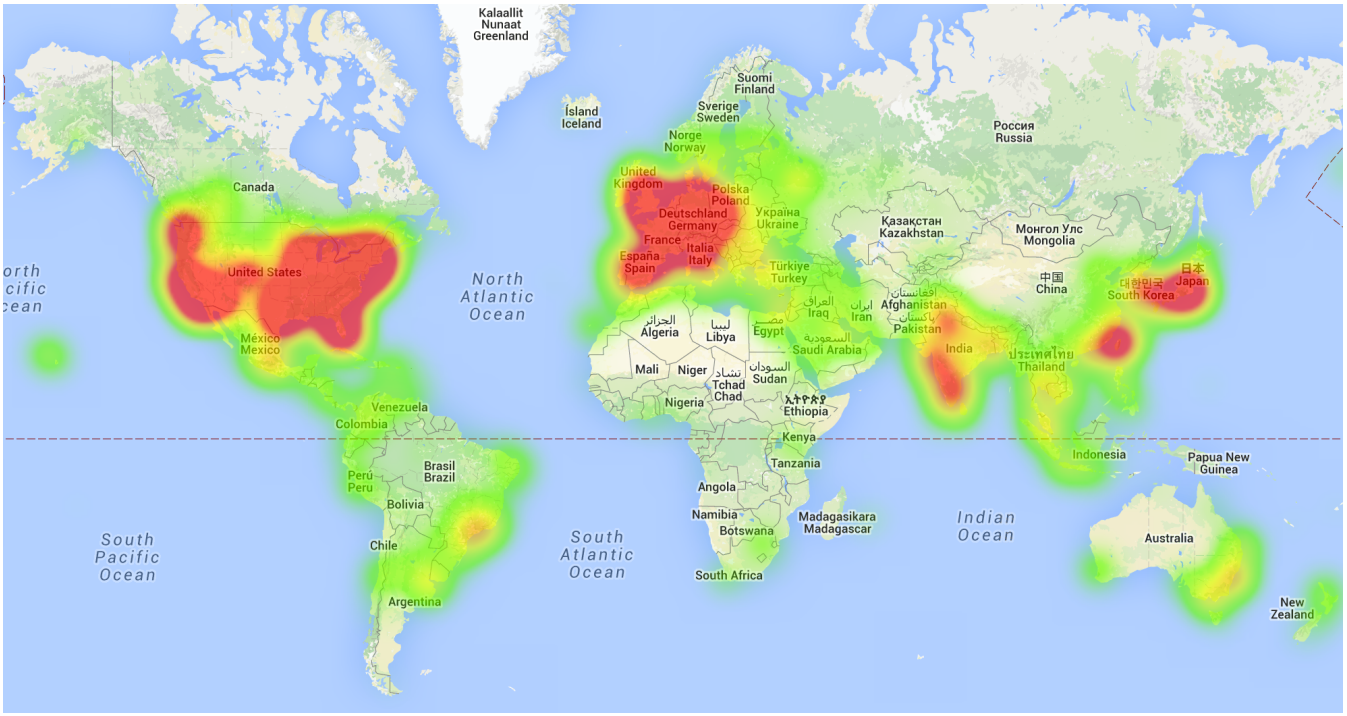

Figure 1: Places on the Google dataset Heat Map

GPS coordinates that determine where the restaurant is located. Finally, the reviews entry is composed of the user that gave the review, the place reviewed, a rating for the place between 1000 and 5000 (we normalize it to the range of 1 to 5), a text review of the place, a list of categories that the reviewer gave to the place (Sushi Restaurant, Coffee Bar, Church, Movie theater, etc), and the time that the review was created.

For the purpose of our project, we filtered and processed the dataset as follows:

First, we took all the reviews from each restaurant and we collected all the categories reviewers defined to each location, so we can map any place to its categories. Second, we took the average of all reviews per place to compute the average rating for the place, and we stored these categories and average rating as a characteristic of the places. Then, we filtered only places that are in the US using the latitude and longitude coordinates. Since we are only interested in predicting the rating a user would give to a specific restaurant, we filtered places whose categories correspond to restaurants ("Italian restaurant", "Asian Restaurant", "Fast Food restaurant", etc) . This left us with 357,191 restaurants in the US. Figure 2 shows a heat map of the locations of restaurants in the US by GPS coordinates.

For the reviews, we filtered out restaurants that are outside of the US, using our previously filtered dataset. Likewise, we took out all reviews whose user is not in out data set (caused by corrupt or missing data). This left us with 1.6 million reviews of restaurants in the US. Finally, we selected only the users that appear on these reviews (692,157 users in total). Figure 3 shows a heat map of rating of each restaurant correspondingly to its location.

Since the size of the feature matrix we selected depends on

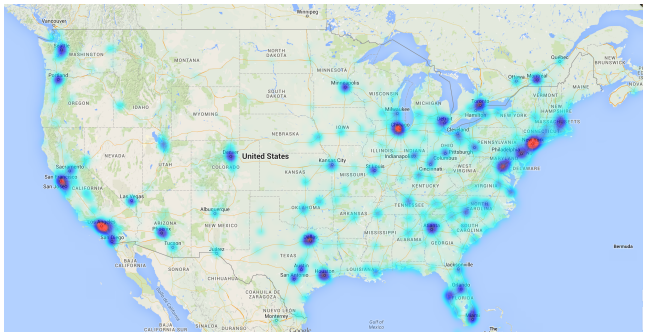

Figure 2: Restaurant Locations Heat Map

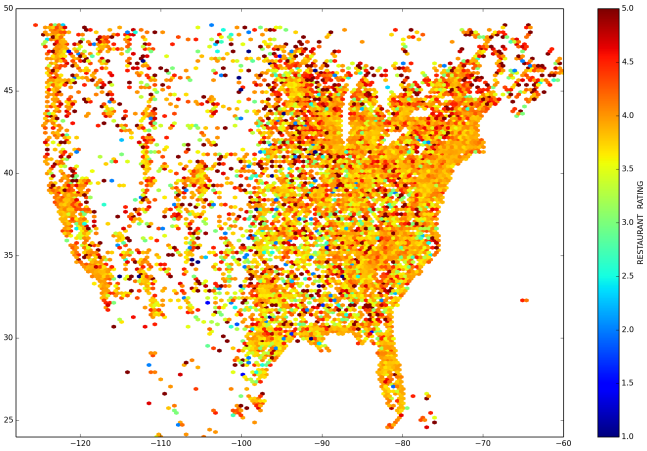

Figure 3: Restaurant Ratings

the number of categories of the restaurants, and this significantly increases the time to train the data, we decided to reduce the amount of categories of restaurants from 363 to 200 in total. First, we merged categories that described the same type of food but had alternative names, such as “Restaurants - Seafood” and ”Seafood Restaurant”, then we also merge categories with misspelling errors. Finally, we checked the categories that were used less than 5 times on restaurants and we replaced them with some other categories. Usually, these categories were either spelling mistakes or too specific for training purposes. During this process we were cautious not to merge restaurant categories that would change their context or specificity, unless it was extremely necessary (i. e. “Restaurants - Uruguayan” with “Restaurants - Latin American”).

2.1 Dataset Statistics

From this filtered data we can highlight the following statistics:

Table 1 shows general statistics about the data. Table 2 shows 10 restaurant categories with most number of reviews. Figure 5 shows the average rating for the 20 restaurant categories with the highest rating. From this graph, we can see that restaurant on the east coast have a higher average rating than the ones on the west coast.

Figure 6 shows the average rating per location on a heat map. This figure does not provide a perfect shape of the US as Figure 3 does, but considering the shape of the later, we can still see that the rating on the west and east coasts are higher than the ones in central USA.

Table 1: General Statistics

Number of distinct restaurant categories	200
Average review rating for places	3.994
Average review rating for places in the US	3.978
Average review rating for restaurants in the US	3.86
Average number of reviews per restaurant	4.5
Minimum average rating of a restaurant in the US	1.0

Table 2: Top 10 restaurant categories with most number of reviews

Restaurant category	# of reviews
Restaurant (Generic)	95225
Restaurants - Fast food	62372
Restaurants - American	51259
Restaurants - European	50239
Restaurants - Asian	43532
Restaurants - Pizza	41081
Restaurants - Italian	39191
Restaurants - Mexican	32950
Restaurants - Burgers	29797
Restaurants - Latin American	27985

2.2 Clustering locations

The Google dataset contains places of interest (e.g., restaurants, shops, etc.), their addresses and their GPS coordi-

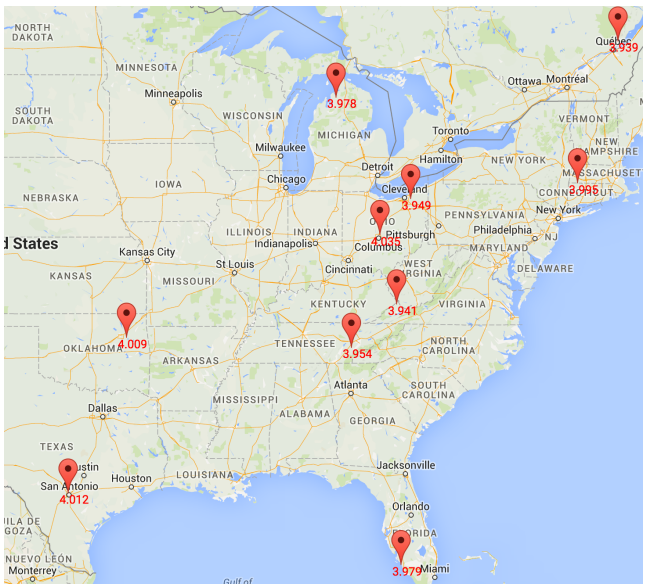

Figure 4: Top 10 locations with the highest rating

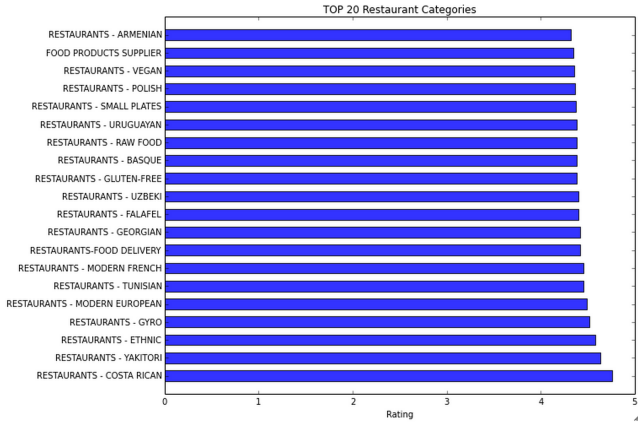

Figure 5: Top 20 Restaurant Categories

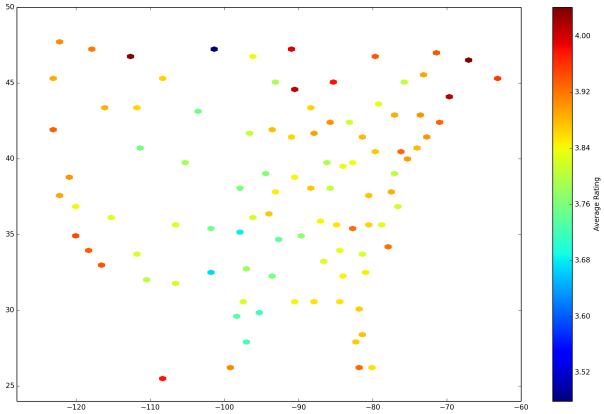

Figure 6: Average rating per location

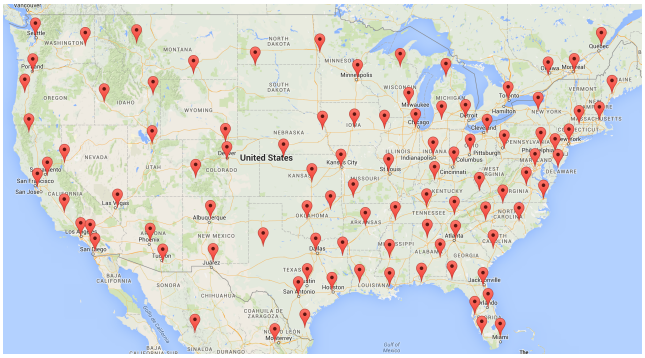

Figure 7: 100 centroids of labeled locations

nates as decimal degrees with negative numbers of south and west. Some of the users on Google's data set have GPS coordinates as well, and we filtered out those who do not have it. We believe that different restaurants in the US will get different scores for the exact same restaurant in a different location. Asian food will probably get a higher score in California than in Texas, American food will probably get higher score in South Carolina than in Oregon, and so on. One way to approach it is by using zip-codes, but that is an information we did not have. To imitate zip-codes we needed to map 2-dimensional space into a natural single range. This is done using K-Means algorithm. We took all GPS coordinates of restaurant and grouped them together into a matrix, each group got its own label. Intuitively, regions (labels) with more restaurants will have more samples as part of it, and places that are more isolated will have one big region. After investigating the amount of labels, we decided to use $K = 300$ over all of the US as a starting point. Later on, we saw that $K = 300$ makes some of the algorithm run for too long, and we believed the accuracy from such granularity is not worth it, so we reduced it to $K = 100$ labeled regions. The result mapping is as follows, where each red dot is a labeled region:

K-Means clustering algorithm uses as an input a features matrix. In our case, each column is the "south" and "west" real value coordinates, and each row is a restaurant sample. The training result is called a vector of centroids. A centroid is defined as the center of gravity of each group of samples assigned to a label. Given a new sample, in order to predict its label, we simply need to find the centroid that is closest in Euclidean distance to that sample. The minimal distance is defined as

$$argmin_k \left\| X - C_k \right\|_2^2$$

. This can be computed in $O(K \cdot n)$. The greedy algorithm for K-Means is relatively simple. We pick K random (or more sophisticatedly computed) centroids and, apply the following formula on all C_k-s until no more changes are made:

$$C_k = \frac{\sum_i \delta(y_i = k) \cdot X_i}{\sum_i \delta(y_i = k)}$$

and

$$\delta(true) = \vec{1}, \delta(false) = \vec{0}$$

2.3 Relevant words on reviews

As part of our efforts to improve the *coefficient of determination*, we tried to reinforce the scores per review by using positive and negative common words in reviews. We defined Positive (P) reviews as

$$P = \{\omega | r \in Reviews, \omega \in r, Score(r) \geq 4,000\}$$

and

$$N = \{\omega | r \in Reviews, \omega \in r, Score(r) \leq 2,000\}$$

. Of course, we wanted to avoid words that are potentially neutral, and that can be done by computing $P = P \setminus (P \cap N)$ and $N = N \setminus (N \cap P)$. Finally, we sorted each set by the number of occurrences (high to low). By taking top 50 positive reviews and top 50 negative reviews, we could accumulate for each restaurant the amount of times it used each positive and negative words as a vector of size 100. The result we got for the positive and negative vectors (sorted) were:

Table 3: Top 50 positive and negative words in reviews

Positive	'wonderful', 'fantastic', 'loved', 'perfect', 'reasonable', 'attentive', 'outstanding', 'variety', 'pricey', 'specials', 'yummy', 'brunch', 'helpful', 'unique', 'die', 'beautiful', 'ambiance', 'incredible', 'perfectly', 'beers', 'vegetarian', 'neighborhood', 'homemade', 'patio', 'flavors', 'cuisine', 'miss', 'yum', 'chicago', 'glad', 'beat', 'packed', 'chocolate', 'affordable', 'flavorful', 'reasonably', 'pho', 'environment', 'easy', 'healthy', 'gem', 'downtown', 'comfortable', 'crispy', 'cozy', 'interesting', 'favorites', 'consistently', 'generous', 'tender'
Negative	'awful', 'disgusting', 'nasty', 'waste', 'worse', 'charged', 'sucks', 'sick', 'gross', 'hair', 'burnt', 'soggy', 'barely', 'employee', 'tasteless', 'refused', 'overcooked', 'sent', 'refund', 'supposed', 'crap', 'cashier', 'mess', 'driver', 'stale', 'apparently', 'bother', 'ridiculous', 'health', 'poisoning', 'unprofessional', 'returned', 'ignored', 'joke', 'spoke', 'receipt', 'nor', 'threw', 'hung', 'messed', 'answer', 'telling', 'complained', 'shame', 'needless', 'calling', 'standing', 'upset', 'disappointment', 'poorly'

We assumed at that point that using keywords will not improve accuracy by much, as the scoring information per review is more comprehensive and accurate than a combination of common positive and negative words. Nevertheless, we hoped that using it will improve regions separation, and users' unique taste. A user that mentioned positively the waitresses beforehand, will probably mention the service afterwards again, and a restaurant with an exceptionally good service will probably get a higher score from that user.

3. FEATURE MATRIX

In order to apply a learning algorithm on the data, we needed to choose the features that will be useful for the pre-

diction task. On this stage of the project, we tried different combinations of feature vectors to improve our prediction.

First, we created a feature matrix with a column of 1s to represent the constant offset. We added 300 columns to represent the location of each restaurant, computed using K-means algorithm. Column K could be "1" iff the GPS location was labeled as value K (and "0" otherwise). If the restaurant is in cluster labeled 5, all columns are 0 expect column 5 that has a value of one. Additionally, we included 363 columns (decreased later on) that represented the initial categories. For each restaurant, value one on a column C represented that the restaurant had at least category labeled C. With this feature matrix we tried to predict the rating of a restaurant considering its location and the type of food it sells (given by the categories). From that, we discovered that the location of the restaurant does not significantly affect the final rating of the restaurant. The weights of this training were mainly significant on the offset and the restaurant categories. The final results were not as impressive as we expected them to be, and that is the reason we created a new feature matrix.

As a second feature matrix, we decided to predict the rating given to a restaurant by an user. This prediction is useful for recommending restaurants to users and giving an approximate rating to the recommended restaurants according to the user's past reviews. In this new feature matrix we had a row for each restaurant review (1.6 million) rather than restaurant sample, each feature vector had, as before, one that represented the offset, 300 columns for representing the cluster of the restaurant as described previously, 363 columns to represent the categories given to the restaurant (type of food they have). Additionally, we added the average rating given to the restaurant, and 363 columns that represent the unique features about the user. Each column represented a restaurant category and the number included in the feature vector was the average rating given by the user, who created the review, to that specific category. In other words, each category go the average rating users gave to that restaurant, and defined it with that category. With this feature matrix, we could only train it with the method of least-squares and stochastic gradient descent efficiently, since the amount of data was too big to handle with l-bfgs, and did not present convergence even after 40 hours.

We also tried a slight change to the second feature matrix, adding 100 columns to represent how many times we found on the review the most common 50 negative and 50 positive words (as described on 2.3). It is worth to notice that this would not be useful if we want to predict the rating that a user might give to a restaurant that he has not been reviewed. As expected, after training the data, we proved that the addition of words found on the reviews of that specific restaurant, did not provide more information than the one the scores alone gave. This is because in case a restaurant had a high average rating, we were to find more positive words counts, and conversely in a low rated restaurant.

The fact that training the feature matrix took more time than expected, we decided to compress the feature vectors. First, we started by reducing the 300 locations clusters into 100. Then, we decided to review the restaurant categories and we decreased them from 363 to 200 by merging categories with different names and misspellings, as described on the first part of section 2. Considering this change, we also decreased the number of columns assigned to the users average rating per category. This changes helped to reduce the number of features for each review from 1028 to 502 in total. This is the feature matrix we ended up using to train and test the data on stochastic gradient descent, linear regression least-squares and l-bfgs.

Table 4 summarizes the feature vector structure for each of the 1.6 million reviews. It shows the number of bits used to represent each feature, as a combination of data from the restaurant and the reviewer.

Table 4: Feature Vector Summary

1	100	200	1	200
offset	location	categories	average rating	average rating given by users for each category

On all mentioned experiments, we randomized the position of the reviews and we took 70% of the data as a training set and 30% as a test set.

4. THE PREDICTION

4.1 Method Used

In order the train the feature vectors over labels, we used linear regression. By defining the feature vectors as a matrix $X \in \mathbb{R}^{S \times F}$ and labels as $y \in \mathbb{R}^{S}$, where S is the amount of samples and F is the amount of feature values on each vector, we would like to compute θ such as

$$X \cdot \theta = y + \epsilon$$

s.t. epsilon (error value from from perfect prediction) is minimal.

There are many approaches to solve the linear regression problem, but in order to explain them we defined three terms:

- $MSE(\hat{y}, y; \theta)$ - the mean square error between the predicted labels and the real labels, and defined as the squared Euclidean distance of prediction from real values.

$$MSE = \left\| \hat{y} - y \right\|_2^2$$

- $FVU(\hat{y}, y; \theta)$ - the coefficient of determination, where $FVU = 1$ means a trivial predictor (always take the average) and $FVU = 0$ means a perfect predictor. It can help us to understand how accurate is our predictor.

$$FVU = \frac{MSE(\hat{y}, y; \theta)}{Var(y; \theta)}$$

and since the variance is defined as

$$Var(y) = MSE(avg(y), y; \theta)$$

$$FVU = \frac{MSE(\hat{y}, y; \theta)}{MSE(avg(y), y; \theta)}$$

.

- $\hat{y} = Predictor(X;\theta)$ - is the prediction of labels, given features matrix of samples, and the training model θ. Since our feature vectors are simply real numbers, defining θ as a vector of real values (weights) is sufficient. From now on, $Predictor(X;\theta)$ is simply defined as

$$\sum_i X_i \times \theta_i^T$$

- $L1$ and $L2$ regularization - in order to give good results in the general case, and not overfit to a specific input, regularization is required. Regularization means bounding the values of weights; they cannot be too higher or too small, thus cannot overfit a very specific case and will perform better in the general case.

$$L_1 = \left\|\theta\right\|_2 = \sqrt{\sum_i \theta_i^2}$$

The "Tikhonov regularization", expects minimal error in overall, and does not necessarily tries to minimize big difference more than small ones (good in case we do want to allow more flexibility in variation). Now consider L_2

$$L_2 = \left\|\theta\right\|_2^2 = \sum_i \theta_i^2$$

L_2 puts more emphasize on this overfitting cases. Since we do want to allow flexibility but avoid overfitting, to get best results we used elasticnet regularization which is a balance between the two:

$$\alpha \cdot L_1 + (1-\alpha) \cdot L2$$

.

4.2 Linear regression least-squares

This approach uses arithmetic to compute θ. Since that $X \cdot \theta = y$ in the ideal case,

$$\theta = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$

. By definition, such an approach overfits the given samples, and if X is square and of full rank, the θ is the "exact" solution of the equation (perfect prediction to train samples), therefore it is good as a comparison tool, but not as a general input predictor.

4.3 L-BFGS

L-BFGS algorithm is a limited-memory version of BFGS, builds and refines quadratic model of a function being optimized. Algorithm stores last M value or gradient pairs, and uses them to build positive definite Hessian *approximation*. This approximate Hessian matrix is used to make quasi-Newton step. If quasi-Newton step does not lead to sufficient decrease of the value or gradient, we make line search along direction of this step.

The quasi-Newton step is the change in weights given this formula:

$$\left(I - \frac{y_k \cdot \Delta X_k^T}{y_k^T \cdot \Delta X_k}\right)^T \cdot H_k \cdot \left(I - \frac{y_k \cdot \Delta X_k^T}{y_k^T \cdot \Delta X_k}\right) + \frac{\Delta X_k \cdot \Delta X_k^T}{y_k^T \cdot \Delta X_k}$$

.

Essential feature of the algorithm is positive definiteness of the approximate Hessian. Independently of function curvature (positive or negative) we will always get "symmetric positive definite matrix", and quasi-Newton direction will always be descent direction, meaning, after enough iterations we will converge to an optimal result.

Another essential property is that only last M function or gradient pairs are used, where M is moderate number smaller than problem size S, often as small as 3-10. It gives us very cheap iterations, which cost just $O(F \cdot M)$ operations.

As mentioned before, the hessian matrix (second derivative) is an estimation, but the goal function to minimize (and usually its derivative for performance and accuracy boost) are accurate. The function we are minimizing over θ is

$$f(\theta) = \frac{1}{N} \cdot \left\|y - X \cdot \theta\right\|_2^2 + \lambda \cdot \left\|\theta\right\|_2^2$$

where the second term is L_2 regularization factor. It is clear that $f(\theta)$ returns a single real number, and forces regularization on MSE. By simply computing the gradient of f, we get:

$$f'(\theta) = \frac{\partial f(\theta)}{\partial \theta} = \frac{2}{N} \cdot X^T \cdot (X \cdot \theta - y) + 2 \cdot \lambda \cdot \theta$$

Notice that $L_1 regularization$ would have been canceled out on the derivative.

The biggest problem with L-BFGS is its performance. Increasing the amount of features reduces performance exponentially since more iterations are required and the hessian matrix becomes more complicated to estimate. We needed some other strategy that does not compute the whole matrix at a time, but improves it iteratively per sample. For that we used stochastic gradient descent.

4.4 Stochastic gradient descent

The biggest difference between BFGS and SGD is that BFGS estimates the hessian (2-dimensional) over all given samples, whereas SGD computes the gradient for each sample locally. That means the computation and memory complexity required for each epoch is smaller, and the amount of epochs (full stack of samples iteration) required is smaller, and usually converges pretty well after 5-15 epochs.

Given a function $f(\vec{\theta})$ *Gradient Descent* finds a stationary point of that function, where the gradient is 0. Meaning, that the weight vector $\vec{\theta}$ has an optimal value. Gradient descent does that by iteratively updating the parameters with the current derivative and hyperparameter λ.

$$\vec{\theta} := \vec{\theta} \pm \mu \frac{\partial f(\vec{\theta})}{\partial \vec{\theta}}$$

Explanation of its correctness can be found in relevant papers. The $(+)$ sign is used for gradient ascent (finding the maximum) and $(-)$ sign for gradient descent (finding the minimum). *SGD* algorithm uses the same technique, only that randomizes the input dataset order, and use $f(\theta) = MSE(;\theta) + \lambda \cdot L_2$ as the function to minimize. Since f provides a numerical value for the correspondence of $\vec{\theta}$ to the training set, maximizing this values results in a maximum correspondence.

Since we defined

$$f(\theta) = \frac{1}{N} \cdot \left\|y - X \cdot \theta\right\|_2^2 + \lambda \cdot \left\|\theta\right\|_2^2$$

and

$$f'(\theta) = \frac{\partial f(\theta)}{\partial \theta} = \frac{2}{N} \cdot X^T \cdot (X \cdot \theta - y) + 2 \cdot \lambda \cdot \theta$$

it is obvious that there is at least one global minimum, and probably many local minimums, but we do not want to overshoot it and stop on a local minimum. Because of that we used an adaptive step size technique, or "Grid Search".

Grid Search is a technique where we run the experiment on different values of θ and μ and find the optimal result. That way, we avoid getting stuck in a local minimum and even improve overall performance for future queries. Notice that the step size μ hyperparameter is also not a fixed one. We decrease its value on each epoch. By doing that, the algorithm achieves a region of optimum faster, yet more accurate and closer to the ideal optimum.

Convergence is achieved either after a fixed amount of iterations, or when the total change in gradient is smaller than than μ value.

5. RESULTS

5.1 Least-squares

As described earlier this method doesn't take into account regularization and it is one of the simplest methods to train our data with. Table 5 shows the MSE, Variance and FVU of when we train our initial feature matrix. As expected, when we calculate the MSE over the test data we can see that the error is unacceptable. The reason for this error is that the weights learned by this predictor are over fitted with the training data. This causes some weights to be extremely positive or negative, predicting ratings a lot greater than 5 and lower than 1.

Table 5: Prediction error training with linear regression least-squares and the 1028 features dataset

	Training set	Test set
MSE	0.831968	2.412797e+19
Variance	1.39926	1.40084
FVU	0.594579	1.72561e+19

When we apply the same method with the reduced feature matrix (502 features), we don't get the same problem of overfitting for two reasons: First, the training and test sets for the two feature matrices are different and, second, the fact that some of the features are collapsed results in less room for overfitting an specific weight with a sample that is not common. Table 6 summarizes the results for this method when applied the feature matrix of 502 features.

Table 6: Prediction error training with linear regression least-squares and the 502 features matrix

	Training set	Test set
MSE	0.832650	0.834614
Variance	1.39814	1.40084
FVU	0.595541	0.595797

5.2 Stochastic Gradient Descent

For the Stochastic Gradient descent method, we first provide the MSE and FVU when we use the 1028 features matrix on Table 7. As expected, the introduction of a method that handles regularization solves the problem of overfitting. Moreover, the nature of SGD prevent the prediction to become extremely off compared with the real values.

Table 7: Prediction error training with Stochastic Gradient Descent and the 1028 features matrix

	Training set	Test set
MSE	0.825505	0.825708
Variance	1.39926	1.39823
FVU	0.589959	0.590537

Table **??** summarizes the results obtained when training the 502 features matrix. The fact that the MSE and FVU of Tables 7 and **??** are similar indicates that the merge of categories of restaurants was done in such a way that we didn't lose important information about the restaurants. Also, it indicates that the use of less locations (100 instead of 300) doesn't compromise the result. This was anticipated considering that, as we mention earlier, the weights for each one of the location doesn't have a strong significance on the result.

Table 8: Prediction error training with Stochastic Gradient Descent and the 502 features matrix

	Training set	Test set
MSE	0.825508	0.826955
Variance	1.39814	1.40084
FVU	0.590433	0.590329

In Figure 8 we can see how training with different μ values changed the accuracy of our predictor (FLV value). In order to avoid overfitting the table shows the FVU on the training data rather than the testing data.

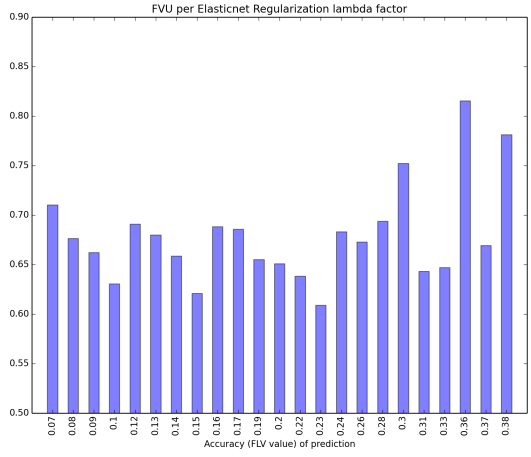

Figure 8: FVU for different regularization factors

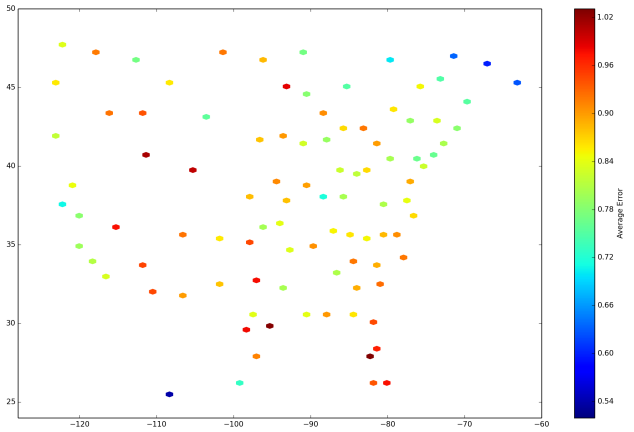

Figure 9: Average Error per location cluster

5.3 l-bfgs

When we train our first feature matrix for predicting the reviews of the restaurants by a user, we first improve the performance of the minimizing function and the computation of the derivative, but even with this changes the 1028 features for each of the 1.6 million reviews made the algorithm run for more than 20 hours without reaching conversion. Once we create the new feature matrix with less location clusters and less categories, we were able to converge in 6 minutes. We expected this behavior since we realize that the time that the algorithm takes to converge grows linearly with the number of samples but exponentially with the number of features.

Table 9: Prediction error training with l-bfgs and the 502 features matrix

	Training set	Test set
MSE	0.856863	0.857289
Variance	1.398141	1.400836
FVU	0.612859	0.611984

Another important result is the Mean Square Error (MSE) for each of the location clusters that is presented in Figure 9. From this figure we can appreciate that the highest errors are mainly on the center of the US. The reason for these becomes apparent if we compared it with Figure 2. From there we can see that the places with less number of ratings have lower accuracy and therefore higher mean square error.

5.4 Comparison between methods

Table 10 summarizes the results of the three methods used with the 502 features matrix. This table also shows a comparison of the performance of each one of the methods used. Least squares shouldnâĂŹt be used since it doesnâĂŹt take into account regularization and will produce overfitting as shown in Table 5. SGD is useful since it can converge quicker than other methods with comparable levels of accuracy. This method is specially useful with big feature matrices since instead of calculating the derivative of the minimize formula, it estimates it. L-BFGS represents a good method for learning the weights but for our case it was exponential on the number of features given.

Table 10: Comparison of the methods used using the 502 features matrix

	l-bfgs	Stochastic Gradient	Stochastic Gradient	Least squares
Max # of iterations	15000	500	5	n/a
MSE	0.857289	0.826955	0.829287	0.834614
Variance	1.40084	1.40084	1.40084	1.40084
FVU	0.611984	0.590329	0.591994	0.595797
Convergence Time (s)	324	982	11	445

6. CONCLUSIONS

After pre-processing the Google locations data and selecting the features that we considered as helpful for predicting the users rating for a restaurant, we provided three ways to train the weight vector needed for the prediction. From the results presented we conclude that the least squares method should not be used since it overfits the data. SGD and l-bfgs are good candidates as learning algorithms, as long as regularization is used. The use of the locations of a restaurant does not affect the final rating in the same proportion as the categories of the restaurant do. For future improvements of our prediction task we could consider the time at which the review was created, as well as using a predictor for any possible location around the globe, in case we consider the entire world data set.

7. REFERENCES

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2007.

[2] C. Elkan. Maximum likelihood, logistic regression, and stochastic gradient training.

[3] P. Flach. *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012.

[4] K. P. Murphy. *Machine Learning: A Probabilistic Perspective (Adaptive Computation and Machine Learning series)*. The MIT Press; 1 edition, 2012.

[5] Wikipedia. Broyden-fletcher-goldfarb-shanno algorithm — wikipedia, the free encyclopedia, 2015. [Online; accessed February-2015].

[6] Wikipedia. Stochastic gradient descent — wikipedia, the free encyclopedia, 2015. [Online; accessed February-2015].