Restaurant Recommendation System

Ashish Gandhe
ashigan@{stanford.edu, microsoft.com}

Abstract

There are many recommendation systems available for problems like shopping, online video entertainment, games etc. Restaurants & Dining is one area where there is a big opportunity to recommend dining options to users based on their preferences as well as historical data. Yelp is a very good source of such data with not only restaurant reviews, but also user-level information on their preferred restaurants. This report describes the work to learn to predict whether a given yelp user visiting a restaurant will like it or not. I explore the use of different machine learning techniques and also engineer features that perform well on this classification.

I. INTRODUCTION

Local business review websites such as Yelp and Urbanspoon are a very popular destination for a large number of people for deciding on their eat-outs. Being able to recommend local businesses to users is a functionality that would be a very valuable addition to these site's functionality. In this paper I aim to build a model that recommends restaurants to users. The way we will model this is by predicting whether a user will have a positive or a negative review for the business. We will restrict to restaurants segment within the business category as recommendation is a very good fit in that system. One way this model could be used in practice is by having an automatic 'Recommend: Yes/No' message when a user visits a restaurant's profile page. The way the problem is modeled is to be able to predict yes/no for any given restaurant and user.

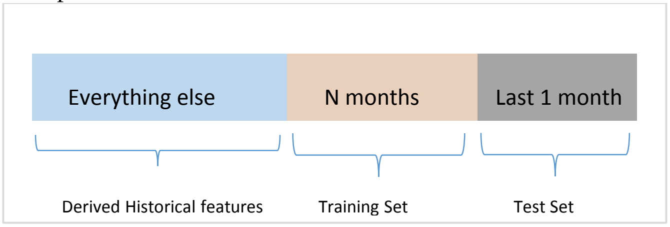
In this work, we will primarily explore the following directions: 1) Optimization algorithms to predict the desired label 2) Develop features that would help improve the accuracy of this model.

There are systems that exist today that recommend users restaurants, but none of them model the problem in this way to predict a yes/no given a user and a restaurant. To my knowledge, this is the first solution that attempts to recommend a Yes/No given a user and a local business. One assumption we make in this work is that the reviews data is not biased by the label. I.e. the majority of users are uniformly writing reviews for restaurants they visit, and not because of their good or bad experience.

II. DATA COLLECTION

The data that we used in this project was obtained from the Yelp Dataset challenge [1]. The dataset contains five different tables: User, Business, Review, Check-In and Tips. The data has 14092 restaurants, 252395 users, 402707 tips and 1124955 reviews. The reviews span over 10 years of data.

I hold out the latest 1 month of the reviews data as the test data, which contains 22375 reviews from 6/16/2014 to 7/16/2014. In addition, I keep N months of data from the period ending 6/15/2014 as training data, where N is a variable to our learning algorithm. The remaining data from the older period serves a source for generating historical features. Here's a depiction of how the data is split based on time period:



The way the problem is now modeled is to learn from current data to make predictions in the future.
Yelp users give ratings on a 5-point scale, which are mapped to a binary yes(4,5)/no(1,2,3) label. Hence, each example in our training/test data is a single review with a binary label. Roughly, about 65% of labels are yes and 35% are no which means that we can achieve a trivial baseline accuracy of 65% by predicting everything as yes.

III. FEATURES

I will first describe the features being used, and the features that I developed to solve this problem. Given that our input tuple is <user, restaurant> I have features of following categories:

a) User-level features
b) Restaurant-level features
c) User-Restaurant features

Summary of features

I also classify the features into the following buckets:

1. **Raw features**
   I have 407 raw features from the data itself, comprising of 5 user-level features and 402 restaurant-level features. User level features are such as number of days in yelp, number of fans, number of votes etc. Restaurant level features are such as binary features for attributes (parking, take out, etc) and categories (cuisine)

2. **Derived/Computed features**
   As described in the previous section, I hold out majority of the old reviews data for computation of features. This old period is prior and not overlapping to the periods from which I sample training and test data.
   I compute 16 derived features from this holdout 'historical' period that are described in more detail in this section. These consist of B) user-level features such as average user historical rating and business-level features such as average business-level rating, number of reviews. Also includes C) user-category features such as average rating from the user on that category given the current restaurant's category and D) features from user's social network with friends' preferences.

Significant amount of work went into engineering these features, trying different ways to compute them. A lot of improvements in the results came from the iterative creation of new features. I'll next go into the details of the features, and in the next chapter I'll summarize the results of adding these features.

A. Raw features

From the raw data I had five user-level features: number of fans, number of days on yelp and three different vote counts.
There are 61 business attribute features such as binary information about ambience, diet and facilities. There are 233 binary features about cuisine, style of restaurant. There are 108 binary features for the city in which this business is.

B. Historical user and business aggregated features

The first set of features I implemented were based on intuition that a business is likely to receive ratings correspond to their historical ratings:
User-level: Average historical rating from this user, # of reviews
Business-level: Average historical rating for this business, # of reviews
Missing features – Since historical data for certain users/business can be missing, I circumvented this by using some variations of this feature with default values ranging from min to average to max. Using a default seems to have helped across the board as we give the algorithm a way to treat missing values differently than just zero.

C. User-business category based affinity

In order to improve the accuracy further, I decided to implement features that model each user's personal preference. These features are computed as follows:
Step 1: Compute each user's personal preference on each of the possible business categories and attributes. This is computed as the average rating a user gave to each of the business categories in the historical period. One such feature is avg_rating_for_thaicuisine_for_this_user.

Step 2: In the training and test data, I compute matching features comparing user's preference and the business categories. E.g. the best feature in this category was the average rating for this user averaged over all categories that matched the given business's categories.

The intuition behind these features is that user's personal preference on certain categories of restaurants should be a strong signal to whether a user would like a future restaurant.

D. Collaborative Features

The publicly available dataset also provided each user's social graph, i.e. the users' friends. Using the intuition that a user's friends' likings are good representatives of a user's likings, I developed the following feature: Given a business and user in the training/test set, average rating for this business from this user's friends in the historical period. As before, I used suitable variations for default values when the feature was missing a value.

IV. UNDERSTANDING THE FEATURES

Before delving into training a model, I want to analyze the features first. I used a simple measure such as F-Score to identify the top features in our data. Here are the top 10 features based on F-Score on a 1 month training set:

Index	Feature	FScore
1	business_averageBusinessRatingWithDefault	0.1011
2	business_averageBusinessRating	0.0328
3	userbus_averageRatingForAttrMatchWithDefault	0.0123
4	userbus_averageRatingForAttrMatchWithDefaultW	0.0121
5	user_averageUserRatingWithDefault	0.0120
6	userbus_averageRatingForCatMatchWithDefaultW	0.0120
7	userbus_averageRatingForCatMatchWithDefault	0.0113
8	avgfriendratingonthisbusinessD	0.0063
9	business_attributes.Parking.lot	0.0039
10	business_categories_Buffets	0.0035
11	business_attributes.Parking.garage	0.0033
12	business_attributes.Caters	0.0025
13	business_categories_Fast Food	0.0024
14	business_attributes.Parking.street	0.0024
others	…..	

Historical user and business aggregated features: The top feature business_averageBusinessRating is the computed average rating from the historic hold out period. business_averageBusinessRatingWithDefault was normalized to always have a default value even if historical data is missing for that business (the default value I use is the average rating from all reviews).

user_averageUserRatingWithDefault is the average rating for that user with default.

User-business category based affinity: The feature userbus_averageRatingForAttrMatchWithDefault measures the affinity of a user with a business based on historical ratings on its categories.

Collaborative Features: user_averageUserRatingWithDefault is the average rating for this restaurant from the user's friends.

The remaining features on the list are binary features on business categories and their names are self-explanatory.

The top 8 performing features are all derived features described in the previous section, with the top feature being historical average rating of the given restaurant.

V. MODELING THE PROBLEM

In this section I will describe different algorithms I used varying parameters such as size of training data, subset of features and evaluate their performance.

I experimented with a few different algorithms, variations in training data, as well as features. I'll describe the results from each of them separately. Before proceeding I'll define the feature set I used in the results:

Feature Set 1: Consists only of the raw features defined in III.A section

Feature Set 2: Consists of the raw and derived features defined in III.A and B sections, i.e. this includes historical average ratings per user and business and simple review stats.

Feature Set 3: Consists of all the raw and derived features defined in III.A, B, C and D sections. This includes all the features previously mentioned, including user category affinity and collaborative features.

A. Learning Algorithms Used

I experimented with the following algorithms to train the rating predictor classifier:

- SVM with RBF Kernel
- Linear SVM
- Logistic regression

A.1. SVM with RBF Kernel

My first approach was to train an SVM classifier using the radial basis function kernel. Since the amount of training data needed at this point is not clear, I vary the training data size with reviews period ranking from 1 week, 1 month, 2 month to 4 months (the test data remains unchanged).

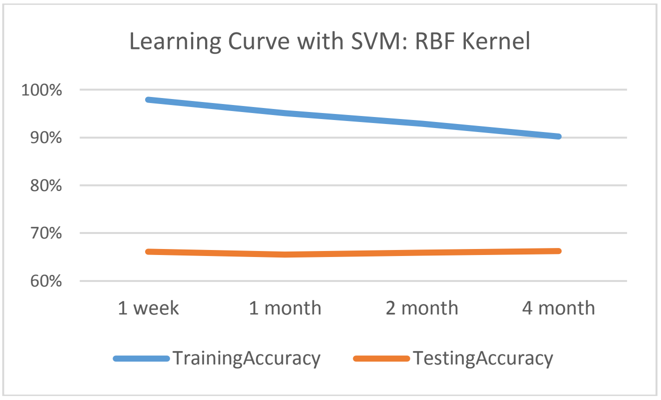
I measure accuracy defined as the percentage of reviews where I predicted a positive or negative review correctly.

The table below shows the change in performance of the model with varying training data size with **Feature Set 3**.

Table 1: SVM with RBF Kernel with Feature Set 3

# training data	Training Data Size	Training Accuracy	Test Accuracy
1 week	5000	97.88%	66.12%
1 month	20000	95.10%	65.50%
2 month	40000	92.88%	65.88%
4 month	80000	90.18%	66.22%

The learning curve looks like follows:



**x-axis corresponds to size of training set*

From the learning curve, it's clear that some over-fitting is happening with lesser training data, and adding more training data helps that. However, even with 4 months' worth of training data I see that the testing accuracy only marginally improves.

Reducing over-fitting:

The effect of high over-fitting is likely arising from the high dimensional feature mapping from the RBF kernel. I iterated on some regularization methods to achieve significantly better results (optimized gamma and C using a parameter sweep with cross validation).

Table 2: Regularized SVM with RBF Kernel w/ Feature Set 3

# training data	Training Data Size	Training Accuracy	Test Accuracy
1 week	5000	73.10%	68.59%
1 month	20000	70.78%	69.41%
2 month	40000	70.38%	69.33%
4 month	80000	70.16%	69.54%

This clearly has lesser over-fitting and better test accuracy.

A.2. Logistic regression

I applied the same training and test data with Feature set 3 with logistic regression, and the results were as follows:

Table 3: Logistic regression w/ Feature Set 3

# training data	Training Data Size	Training Accuracy	Test Accuracy
1 week	5000	71.97%	69.10%
1 month	20000	69.93%	69.55%
2 month	40000	69.43%	69.68%
4 month	80000	69.40%	69.28%

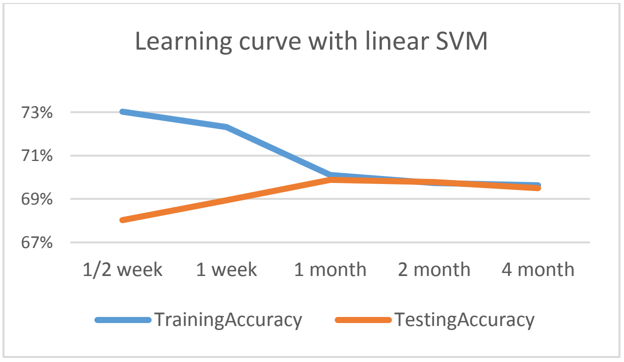
A.3. SVM with Linear Kernel

I observed that given the vast feature set we have high dimensional kernel for SVM did not add a lot of value. In fact, there was over-fitting in the high dimensional space until significant regularization was added. I experimented with SVM with a linear kernel which reduced over-fitting and produced comparable and even slightly better results as shown in the table below.

Table 4: SVM with linear kernel

# training data	Training Data Size	Training Accuracy	Test Accuracy
1/2 week	3000	73.02%	68.01%
1 week	5000	72.31%	68.94%
1 month	20000	70.10%	69.89%
2 month	40000	69.73%	69.77%
4 month	80000	69.63%	69.49%

Here's the learning curve for linear SVM:



As we see, we achieve comparable training and test accuracy with large enough training data looking at periods of 1 month or more. There's little to no over-fitting happening at this stage, and this is the maximum we can learn from the given set of features and training data.

B. Impact of derived feature sets

The following table compares the testing accuracies with different feature sets with varying training data.

Table 5: Testing Accuracy with different feature sets with linear SVM:

# training data	Feature Set 1 (raw only)	Feature Set 2	Feature Set 3 (all derived)
1/2 week	64.90%	67.91%	68.03%
1 week	65.68%	68.81%	68.91%
1 month	66.97%	69.79%	69.89%
2 month	66.97%	69.77%	69.82%
4 month	67.09%	69.45%	69.50%
8 month	-	69.30%	69.42%

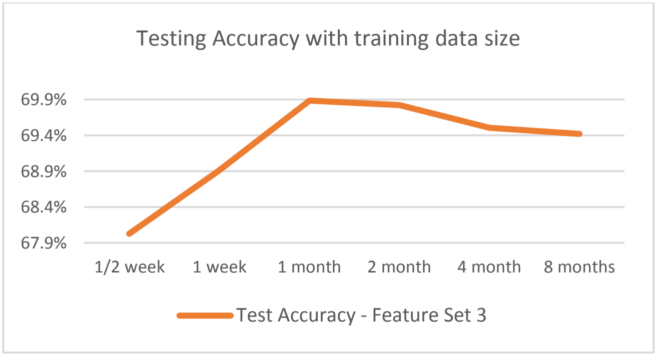
The table clearly shows the superior results from using the derived features. Feature Set 3 is the set of all raw and derived features and it consistently performs better than Feature set 2 for all varying training data, although only marginally better.

C. Impact of varying training data size

We see interesting results with varying training data size in Table 5.

Specifically, we see training accuracy go down with increase in training data size. This indicates a good reduction in variance in that the over-fitting problem is fixed with increased training data.

To analyze testing accuracy better, let's present a zoomed-in learning curve plot only for testing accuracy on feature set 3 with linear SVM:



We see test accuracy increase with increased training data until about 1 month of training data, but it reduces with significantly increased training data such as 4 or 8 months. We explain this behavior with the following hypotheses. *The way we sample training data size is not random, rather increase in training data is done through going back more in time and holding out more old data for training*. This also means that the historical hold out period from which derived features are computed also gets older with increase in training set size. I summarize the hypothesis here:

- Recent training data is more representative of reviews in the upcoming period than older training data.
- Derived features from recent period are stronger signals for predicting reviews in the upcoming period.

Thus it is important in such machine learning algorithms when using past data to predict the future results to optimize on varying holding out data for features and training data. We probably want to experiment with weighing training data based on its age.

VI. SUMMARY OF RESULTS

I summarize the results from the previous section as follows:

1. We see comparable results from the different algorithms that were used although linear SVM was least susceptible to over-fitting and performed marginally better. We achieved a testing accuracy of 69.89% with linear SVM,

feature set 3 and using 1 month of reviews as training data.
2. We see a significant improvement from derived features, specifically from using the following:
   a. Historical average ratings for the business
   b. Affinity of user to a specific business category
   c. Collaborative features
3. Increased training data reduced over-fitting, but there's value in weighing training data based on the age of the label. Recent data is more useful in learning than older data.
4. It was important to treat missing feature values differently than zero by providing variations to the model to learn from.
5. At the end, we perform about +5% in accuracy better than the trivial baseline of always predicting yes.

VII. FUTURE WORK

I acknowledge that the problem being solved is hard, specifically because of the following reasons:
1. Unclear Predictability of reviews: Any supervised learning problem aims to learn from the labels, given the provided features. The underlying assumption I made here is that the features I have access to are sufficient to predict a positive/negative review. However, one can imagine that a future review can depend highly on the experience the user has at the restaurant that is not captured anywhere in the features. This could cause correlation between the features and the label to be lesser than what would be ideal for a supervised learning problem. At this stage, it's not clear as I have not yet explored all the possible features, but it is a concern to me.
2. Unclear bias in reviews used for training and evaluation: One assumption we make is that a user's decision to provide reviews to a restaurant is random, and not biased by an unusually good or bad experience one has.

Future work would involve trying to identify stronger features beyond what is available in the datasets, as well as investing in an approach to gather training and evaluation data from alternate means (such as explicit human judgment systems).

ACKNOWLEDGMENT

The LIBSVM library was quite helpful in this project. In addition, I would like to thank Prof Andrew Ng for the lecture on insights when applying machine learning. I also want to thank the CS229 teaching assistants and my peers for fruitful discussions on the student forum.

REFERENCES

[1] Yelp dataset: https://www.yelp.com/academic_dataset

[2] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm