A RESTAURANT RECOMMENDATION SYSTEM USING DEEP NEURAL NETWORKS

Harika Konagala

A paper submitted to the
Computer Science & Electrical Engineering Department
in partial fulfillment of the requirements for the M.S. degree at
University of Maryland Baltimore County

05.08.2017

Certified by: Dr. Tim Oates

Advisor's Signature:_____Date_____

ABSTRACT

Every day we form opinions about things we like and dont like. Our tastes vary, but we generally follow patterns. People tend to like things that are similar to other things they like. That's where recommendation engines play a part.

Recommendation engines try to match our tastes to almost everything we like these days. For instance, Yelp provides a perfect recommendation service for people seeking restaurants with delicious food and good services. It does a fantastic job of suggesting appetizing restaurants. But, there seems to be room for improvement. Thus the goal of this project is to build a system that can identify a user's preferences and combine restaurant profiles using better collaborative and content based models. In particular, I implemented K-nearest neighbor and deep neural networks to make recommendations and then compare their performance. The models achieved a good RMSE and MRR score of 1.173 and 0.1315 respectively. Besides the results, I discuss interesting conclusions about the behavior of each model.

Table of Contents

List of Figures

Chapter 1

Introduction

Recommendations are quite popular. We have seen them in practice on sites such as Amazon, Netflix, Spotify or Last.FM based on browsing or historical records. The main goal is to produce a list of products that may be interesting to you. These techniques have been a topic of research in machine learning since the 1990s and have become more mainstream. Demand for them has increased, and supply of open-source implementations has as well. This, along with increasingly accessible and cost-effective computing power means that recommendation engines are becoming more widely used.

Recommendation engines can also be applied to estimate the strength of associations between many things or even estimate which customer might like a certain item the most. In social networks, one such example is Facebook that utilizes recommendation systems to suggest friends to users. Given this general theme, Yelp is a good place to start, considering the vast database of reviews, ratings, and general information provided by the community about businesses. And when it comes to algorithms like Nearest Neighbor, they provides an easy way to implement recommendations. But how accurate are those recommendations? In a competitive e-commerce market, a simple recommendation system wont serve your purpose. That's where deep learning comes into the picture. It has been shown that some problems can be

solved easily with shallow machine learning but deep learning fits better for other problems. The main difference between these two comes in the feature engineering part. The accuracy of machine learning algorithm heavily depends on the feature engineering done on the data set, while deep learning does this part by itself.

Thus, this project aims to build a recommendation system that will enable us to make sophisticated food recommendations for Yelp users by applying deep learning algorithms and compare their performance to shallow machine learning systems.

Chapter 2

Motivation & Related work

There has been a significant amount of research in the field of recommendation systems, much of it with the yelp dataset. The goal of these systems is to make sophisticated recommendations and there are multiple ways of doing this. The only difference is how efficient and user friendly the system is in helping users find what they like.

Two research works inspired this project. Collaborative deep learning for recommender systems [1] by the students of Hong Kong University built a hierarchical Bayesian model called collaborative deep learning (CDL), which jointly performs deep representation learning for the content information and collaborative filtering for the ratings. The authors of Wide & Deep Learning for Recommender Systems [2] worked on Wide & Deep learning- jointly training wide linear models and deep neural networks- to combine the benefits of memorization and generalization for recommendation systems. Similar to the work in [2], I have implemented the deep neural network and observed how well it performed over various other models.

The next few sections will briefly talk about the dataset used while creating the recommendation system, followed by the algorithms used, experiments and their performance evaluation, finally concluding with the outcomes and discussion of future work that could be explored.

Chapter 3

Dataset

Yelp provides a portion of its data through the Yelp Dataset Challenge event [3]. The dataset includes 144,072 businesses, 1,029,432 users, and 4,153,150 reviews, which include star ratings in the range of 1 to 5 and user's opinions in text along with 200,000 pictures from 85,901 businesses described in the main dataset. Each file is composed of a single object type, one json-object per-line. Some of the important columns in each data file are mentioned in Table 3.1 and basic statistics are mentioned in Table 3.2.

File	**Columns**
Business	business_id, name, review_count, attributes, categories, stars, city, state
Review	review_id, stars, text
User	user_id, name, review_count, average_stars
Checkin	time
Tip	text, likes

Table 3.1: Yelp data with important columns

Data	**Count**
Businesses	144072
Users	1029432
Reviews	4153150
Check-ins	125532
Tips	946600

Table 3.2: Basic statistics on Yelp data

Chapter 4

Recommendation system

Recommendation Engines are a subclass of information filtering systems that seek to predict the rating or preference that a user would give to an item. There are many approaches to build a recommendation system. Some of them are listed below.

4.1 Collaborative Filtering

Collaborative filtering [10] methods are based on collecting and analyzing a large amount of information on users behaviors, activities or preferences and predicting what users will like based on their similarity to other users. A key advantage of the collaborative filtering approach is that it does not rely on machine analyzable content and therefore it is capable of accurately recommending complex items such as movies without requiring an understanding of the item itself. Many algorithms have been used in measuring user similarity or item similarity in recommendation systems. For example, the k-nearest neighbor (k-NN) approach and the Pearson Correlation.

4.2 Content-based

Content-based filtering methods are based on a description of the item and a profile of the user's preference. In a content-based recommendation system, keywords are

used to describe the items and a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past (or is examining in the present). In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended. This approach has its roots in information retrieval and information filtering research.

4.3 Hybrid

Hybrid recommendation systems [14] combine two or more recommendation techniques to gain better performance. This can be done by adding content-based capabilities to a collaborative-based approach (and vice versa), or by unifying the approaches into one model. These methods can also be used to overcome some of the common problems in recommendation systems such as cold start and the sparsity problem.

Netflix is a good example of a hybrid system [14]. Netflix primarily uses your ratings, viewing history, and taste preferences to determine your recommendations (i.e., collaborative filtering) as well as by offering movies that share characteristics with films that a user has rated highly (content-based filtering). The hybrid recommendation engine is visually represented in Figure 4.1.

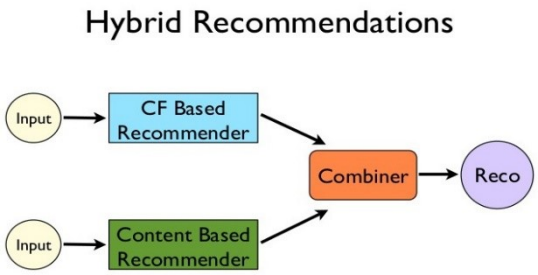

Figure 4.1: Hybrid Recommendation Engine

Chapter 5

Predictive Models

5.1 Neighborhood Collaborative Filtering

The general problem in recommendation systems can be cast as follows, given an $item_m$ and $user_i$, determine the most likely $rating_{mi}$. A common framework for generating such predictions is a neighborhood-based approach [16]. Here similar items that $user_i$ has rated are computed and a $rating_{mi}$ is estimated as some weighted average of the ratings on those items. Some of the commonly used neighborhood-based methods are K-Nearest Neighbor, k-Means, k-d Trees etc.

A KNN approach involves finding the top K nearest neighbors for an item. Ratings from the list of nearest neighbors are combined to predict the unknown rating. This normally involves finding all user-user correlations or item-item correlations and poses the biggest challenge to this approach. Given that for most real-world systems, the number of users is much larger than the number of items, it becomes even more of a challenge to use a straightforward user-user similarity comparison for most practical scenarios.

Thus, most neighborhood based systems such as Amazon these days rely on item-item approaches [15]. In these methods, a rating is estimated by other ratings made by the user on "similar" or "nearby" items. Another problem for the KNN approach is that the notion of similarity is often arbitrary.

The Pearson correlation similarity of two users x, y is defined in Equation 5.1.

$$Similarity(x,y) = \frac{\sum_{i \epsilon I_{xy}}(r_{x,i} - r_x)(r_{y,i} - r_y)}{\sqrt{\sum_{i \epsilon I_{xy}}(r_{x,i} - r_x)^2 \sum_{i \epsilon I_{xy}}(r_{y,i} - r_y)^2}} \tag{5.1}$$

where I_{xy} is the set of items rated by both user x and user y.

Although there are several measures like Pearson correlation, mean squared error based similarity and vector cosine based similarity, sometimes the large number of dimensions in real-world data reduces their effectiveness.

5.2 Deep Neural network

Neural Networks [17] take a different approach to problem solving than that of conventional algorithms and for better understanding we can define them as a network with many layers made up of a number of interconnected 'nodes' which contain an 'activation function'. During training, patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer'.

Neural Networks excel in the region where the relationships between the data may be quite dynamic or non-linear. They provide an analytical alternative to conventional techniques which are often limited by strict assumptions of normality, linearity, variable independence, etc.

While a feed-forward neural network is the simplest neural network without any loops, there are other neural networks which have more complex structures, like

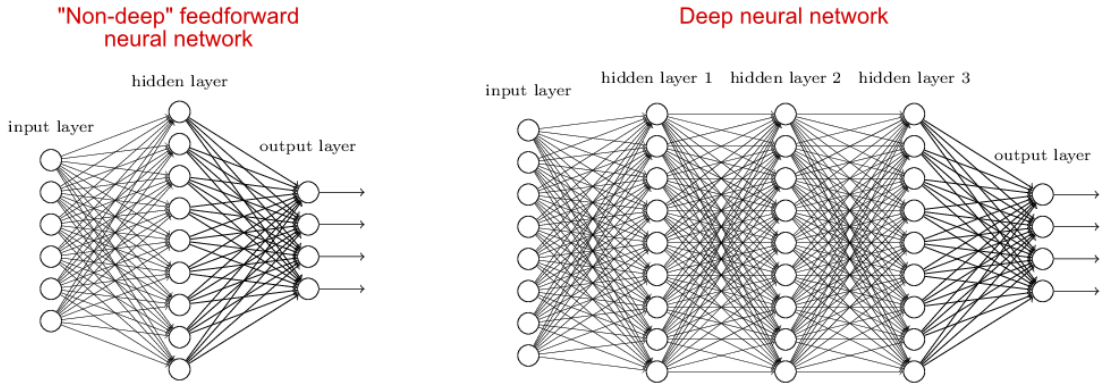

Figure 5.1: Neural Network vs. Deep Neural Network

RNN's (Recurrent Neural Networks) and DNN's (Deep Neural Networks).

A DNN is a Multi-Layer Perceptron (MLP) with many hidden layers. Back-propagation (BP) [5] is employed to learn DNN parameters. The success of DNNs is due to two techniques: a larger number of hidden units and better parameter initialization. A DNN with a large number of hidden units can have better modeling power. Even though the learned parameters of the DNN is a local optimum, the DNN can perform much better than those with fewer hidden units. However, in order to converge to a local optimum, a DNN with a large number of units also requires more training data and more computational power. This also explains why DNNs became popular only recently. Learning a DNN is a highly non-convex problem. It is no doubt that better parameter initialization can lead to better performance. Researchers [6] found that parameters of DNNs can be initialized with the learned parameters of a DBN (Deep Belief Network) with the same architecture. A graphical representation of the neural and deep neural networks is shown in Figure 5.1.

Chapter 6

Approach

6.1 Exploratory Analysis

While the Yelp dataset is huge and has lot of ground for exploring, it is important to understand the data first. When you know what you are dealing with, it becomes easy to build any kind of system using the data. Therefore, the following section has some of the interesting facts from the Yelp dataset.

- Since the goal is to build a restaurant based recommendation system, it is interesting to find that restaurants are the most popular business category in Yelp with a count of 48,485 unique restaurants. Not just that, it also has the highest number of reviews among all businesses.

- Arizona and Nevada are the two most competitively popular states with the highest number of restaurants (Figure 6.1 and Figure 6.2).

- Regarding check-ins, Fridays are the busiest day of the week with the highest number of check-ins. While on any day from 11 AM to 1 PM and 7 PM are the rush hours, early morning 4 AM is the worst time to go to a restaurant (Figure 6.3).

- The term 'service' is the most used word in reviews, occurring in almost 9% of them.

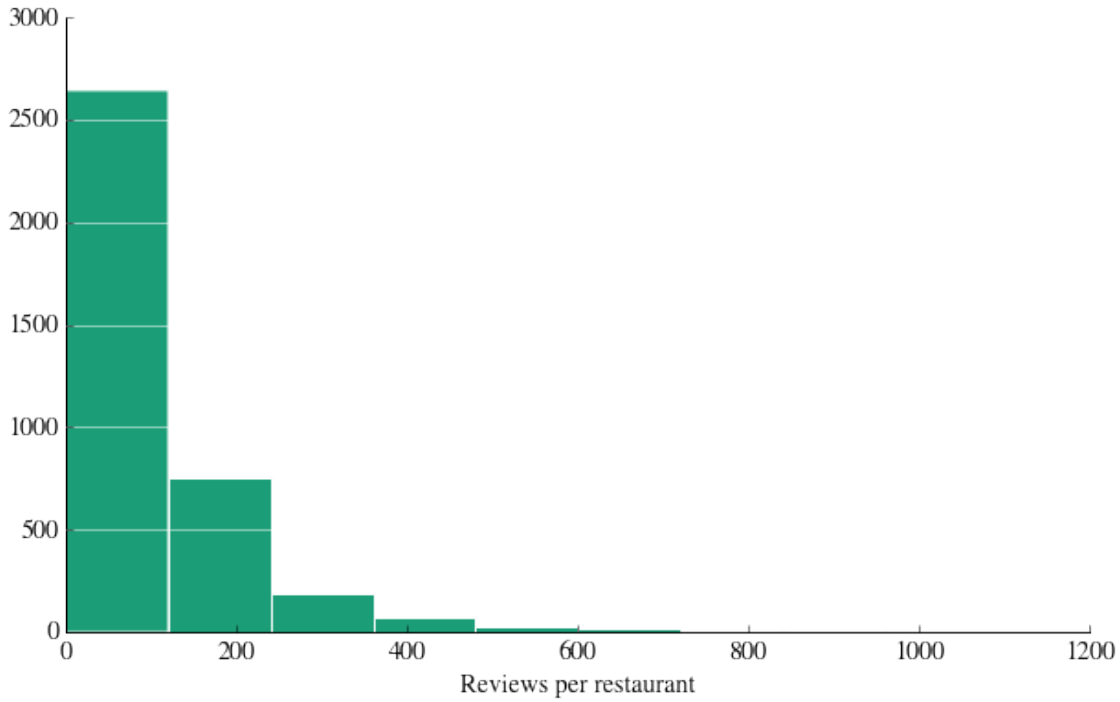

Figure 6.1: Reviews per restaurant

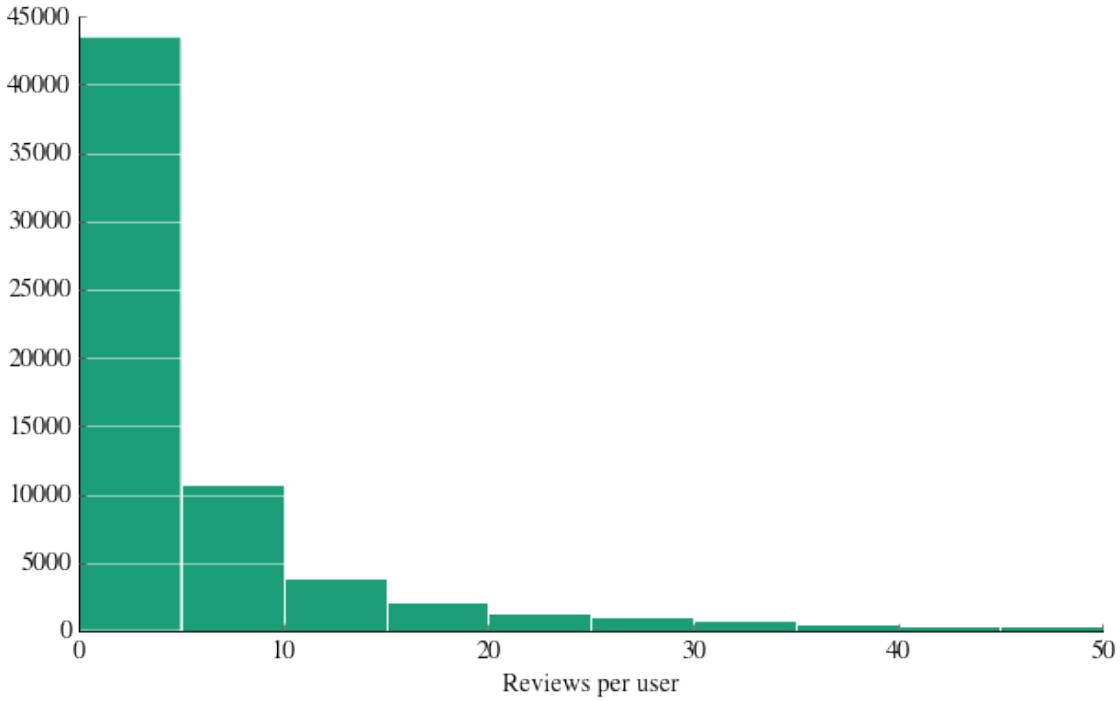

Figure 6.2: Reviews per user

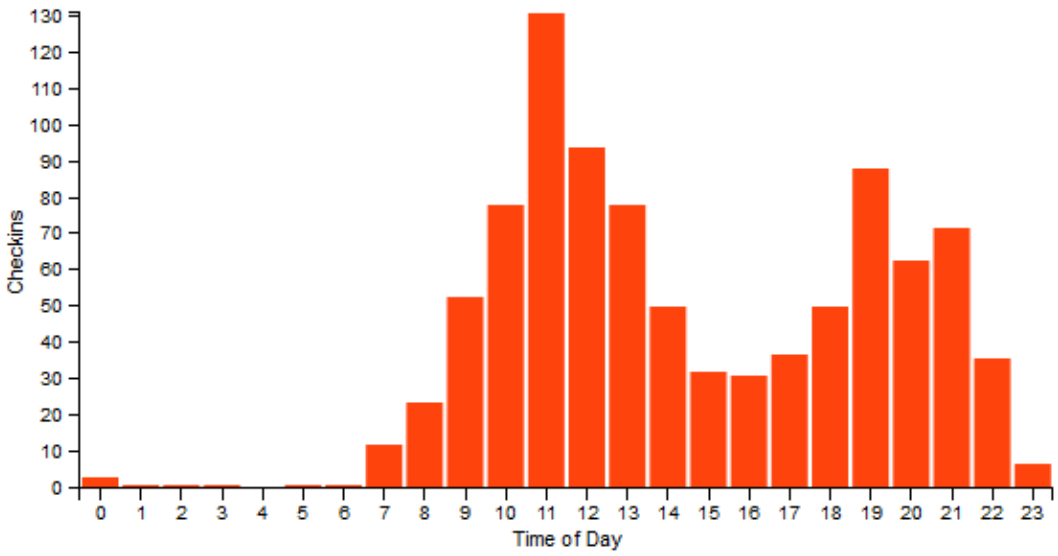

Figure 6.3: Check-ins vs Time of Day

- Only 18% of the restaurants have a user review count greater than 60 and business review count greater than 150, and among them Asian restaurants are voted as the most popular restaurant category by a majority of reviewers.

- While this is just the tip, there's much more we can explore with this data. Fig. 6.4 shows the relation between the reviews and the months. Apparently, the graph tells that July and August have comparatively seen more reviews throughout the year.

- Fig. 6.5, Fig. 6.6, Fig. 6.7 show the correlations between the star ratings over all reviews, average rating by restaurant and users respectively.

- More than 60% of tips are given for categories like 'Restaurants' and 'Food'. From all the above analysis, we can infer that either users provide more feedback for food and restaurants or Yelp is primarily used for searching for this

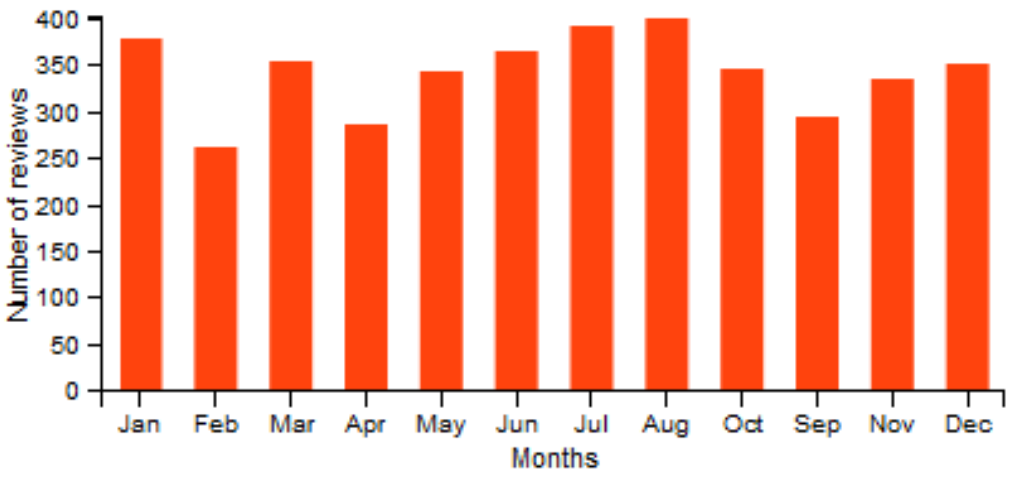

Figure 6.4: Number of reviews vs. Months

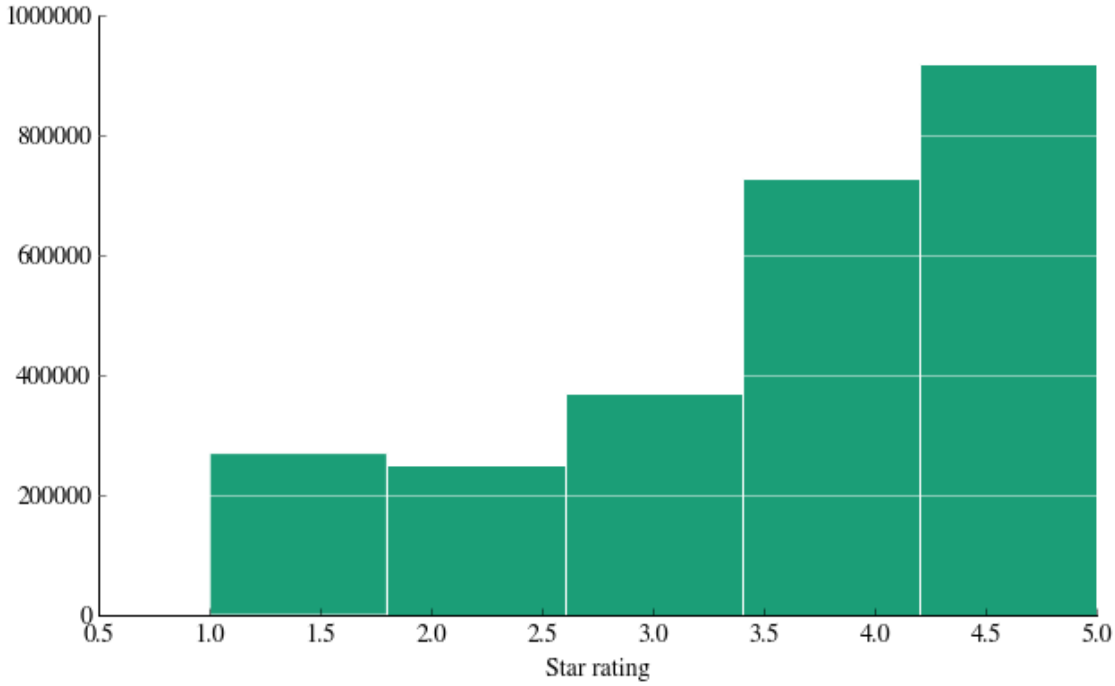

Figure 6.5: Star ratings over all reviews

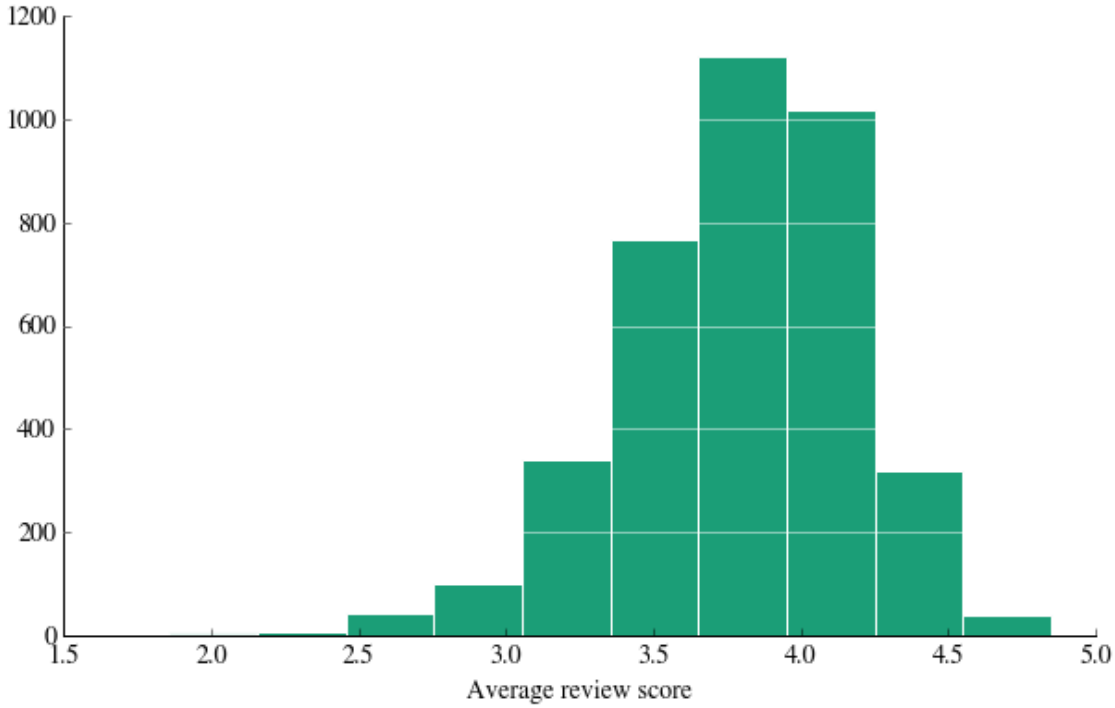

Figure 6.6: Average restaurant rating

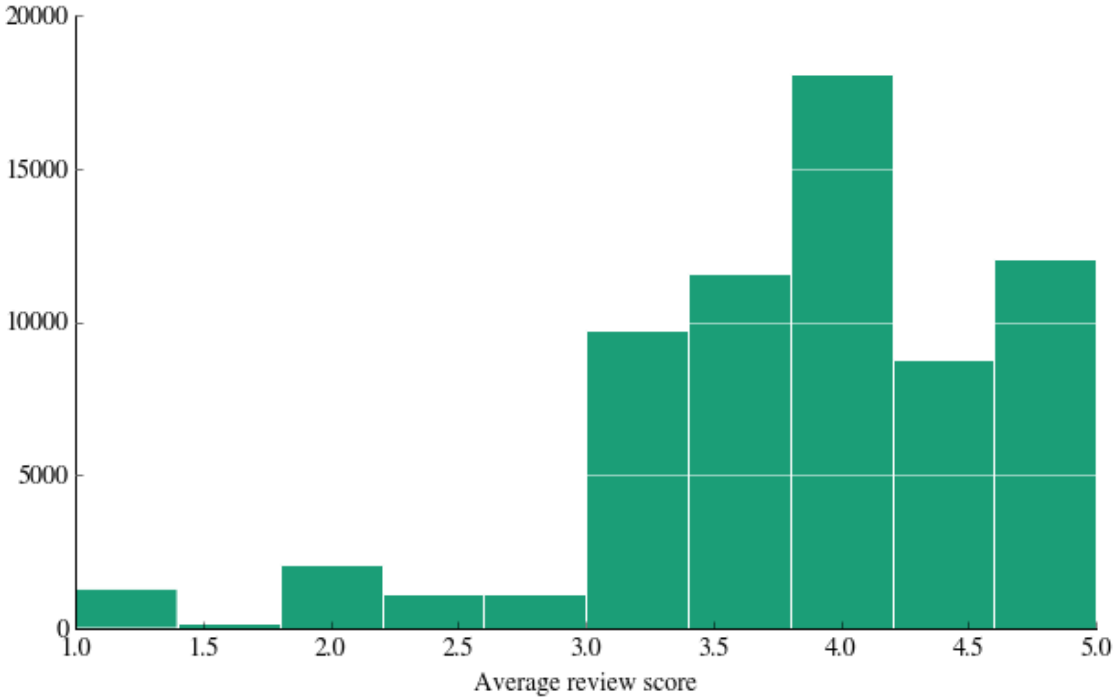

Figure 6.7: Average user rating

purpose.

6.2 Techniques used/ Experiments

The Yelp dataset includes businesses other than restaurants, which is not what we want. After filtering the data for food-related businesses and reviews, there remain 34,219 businesses from 22 states and 674 cities with 2,540,795 reviews, and 75,022 users who gave food reviews. In addition, only a handful of users and restaurants have review counts more than 50. Thus, for convenience, I simplified the dataset further to a single state with review counts greater than 50. After all the trimming, the dataset size has reduced to approximately 4000 restaurants, 66,000 users, and 470,000 reviews.

6.2.1 Evaluation

6.2.1.1 Root Mean Squared Error Metric (RMSE)

After researching the kinds of evaluation metrics prevalently used to evaluate recommendation systems, I decided to evaluate the system based on the root mean square error (RMSE) through k-fold cross validation. As the Microsoft Research paper [7] on evaluating recommendation systems argues, root mean squared error is one of the popular and highly accurate metrics historically used to measure the performance of a recommendation system that aims to predict the rating (1 to 5 stars in Yelp) that a particular user would give to an item.

If $p_{i,j}$ is the predicted rating for user i over item j, and $v_{i,j}$ is the true rating, and

$K \subset \{(i,j)\}$ is the set of hidden user-item ratings, then the RMSE is determined by formula 6.1.

$$RMSE_k = \sqrt{\frac{\sum_{(i,j)\epsilon K}(p_{i,j} - v_{i,j})^2}{n}} \tag{6.1}$$

To be more precise, I have applied the RMSE metric over k-fold cross validation where I split the dataset into train and test sets and calculated the overall RMSE using formula 6.2.

$$RMSE_{overall} = \sqrt{\frac{{RMSE_1}^2 + {RMSE_2}^2 + ... + {RMSE_K}^2}{k}} \tag{6.2}$$

Average RMSE adjusts for unbalanced test sets. For example, if the test set has an unbalanced distribution of items, the RMSE obtained from it might be heavily influenced by the error on a few very frequent items. If we need a measure that is representative of the prediction error on any item, it is preferable to compute RMSE separately for each item and then take the average over all items. Similarly, one can compute a per-user average RMSE if the test set has an unbalanced user distribution.

6.2.1.2 Mean Reciprocal Rank (MRR)

MRR is one of the other metrics best used for recommendation systems [18]. MRR evaluates recommendation systems that produce a list of ranked items for queries based on formula 6.3. The reciprocal rank is the multiplicative inverse of the rank of the first correct item. For calculating MRR the items don't have to be rated. MRR

doesn't apply if there are multiple correct responses (hits) in the resulting list.

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{6.3}$$

6.2.2 Baseline System

Since the goal is to build better predictive models, I started with a basic version of both content and collaborative filtering models as part of the baseline system.

In content-based filtering, the recommendations are made based on the similarity of a restaurant's characteristics to a user's profile. Each restaurant is characterized by its average review score, categories and attributes which were represented by one-hot encoding to construct feature vectors.

A user's profile is a weighted average of the features of the restaurants he/she reviewed, weighted by their rating of the restaurant. Finally, the algorithm recommends the restaurants that are closest in cosine distance to the user's profile.

In collaborative filtering [11], the goal is to look at neighborhood based user-user, item-item and k-nearest neighbor models. However, most CF models start off with a simple baseline model.

For this model, I have used only the mean ratings given by users and the mean of the restaurants to remove any possible bias. To better understand this model, lets walk through an example [15].

Assume that in Las Vegas the average rating of restaurants by all the users is 3.5. Imagine one particular restaurant is better than an average restaurant, so it tends

to be rated 0.5 stars above the average (over all the users). However, you are a harsh food critic who tends to rate 0.2 stars below the average. Then a baseline estimate for the recommendation for the restaurant, for you, is 3.5+0.5-0.2=3.8.

These baseline estimates adjust the data by accounting for the systematic tendencies for some users who give higher ratings than others, and for some restaurants to receive higher ratings than others [15]. We can calculate the baseline estimate $Y_{um}^{baseline}$ for an unknown rating Y_{um} for user u and restaurant or business m as mentioned in formula 6.4.

$$Y_{um}^{baseline} = \hat{\mu} + \hat{\theta_{u0}} + \hat{\gamma_{u0}} \tag{6.4}$$

The unknown parameters $\hat{\theta_{u0}}$ and $\hat{\gamma_{u0}}$ indicate the deviations, or biases, of user u and item m, respectively, from some intercept parameter μ.

Notice that the $\hat{\theta_{u0}}$ and $\hat{\gamma_{u0}}$ are parameters which need to be fit. The simplest thing to start with is to replace them by their mean estimates from the data which leads to calculation of $Y_{um}^{baseline}$ using formula 6.5.

$$\hat{Y}_{um}^{baseline} = \bar{Y} + (\bar{Y_u} - \bar{Y}) + (\bar{Y_m} - \bar{Y}) \tag{6.5}$$

Where $\bar{Y_u}$ = user_avg, the average of all user u's ratings and $\bar{Y_m}$ = business_avg, the average of all ratings for restaurant m. $\bar{Y}$ is the average rating over all reviews. The final two terms correspond to the user-specific and item-specific bias in ratings, that is, how their ratings tend to systematically diverge from the global average. This is the simplest possible way to predict a rating, based only on information about this

user and this restaurant.

6.2.3 User-User k-Neighborhood model

Now that we have the baseline system ready, we can make changes to the CF model and see if it performed better than the previous model or not. For this, I merged all the filtered restaurant data, review data, and user data into one single dataframe based on their business_id and user_id. I particularly used a few relevant columns like business_review_count, user_review_count and the average star ratings of restaurants and users to build models.

In order to implement this initial method, for any given user u, we begin by calculating the similarity between this user and all other users. Then we select its k-nearest neighbors based on this similarity measure and finish by computing the predicted rating for u based on a weighted combination of the k-nearest neighbor ratings as mentioned in formula 6.6.

$$P_{uj} = \bar{r_u} + \frac{\sum_{\bar{u}\epsilon N} S(u,u')(r_{u',i} - \bar{r_{u'}})}{\sum_{\bar{u}\epsilon N} |S(u,u')|} \tag{6.6}$$

where

$P_{u,i}$: user u's prediction for restaurant j

$\bar{r_u}$: mean prediction for user u

$S(u,u')$: similarity between user u and user u'

$r_{u',i}$: rating of user u' for restaurant i

N: neighborhood of user u

In the next part, we will focus on using item-item neighborhood [15]. To do this, we compute a similarity measure S_{mj} between the m^{th} and j^{th} items. This similarity is measured using the Pearson coefficient. This measures the tendency of users to rate items similarly. Since most ratings are unknown, it is computed on the "common user support" (n_common), which is the set of common raters of both items.

6.2.4 Item-Item k-Neighborhood model

For this model, we calculate the similarity between items using their entire common user support, and rank the nearest neighbors of an item by this similarity. Later on, we can modify it to pool information based on which items seem the most similar to this user. This recommendation system does have the advantage of dealing with the possible sparsity of the user's rated items, but also the disadvantage of recommending the same restaurants for all users, without taking the user's preferences into account. This is a classic case of bias-variance trade off.

Therefore, it is important to check for any kind of data sparsity. Once the data is less sparse we can re-calculate the common user support.

The common support calculates the common users for each pair of restaurants. It will be used to modify similarity between restaurants. If the common support is low, the similarity is less believable.

While computing the similarity, even though each reviewer uses the same 5-star scale when rating restaurants, comparing two users by comparing their raw user

ratings can be problematic. To make the ratings comparable we must subtract the average rating of the user from the actual rating of the restaurants. Then we can create a database of pair wise restaurant similarities.

But there seems to be a slight problem. Consider two cases where there is just one common reviewer, and where there are multiple common reviewers. In both the cases, the similarity can get biased, so to control the effect of small common supports, we can shrink the Pearson co-efficient by using the "regularization" parameter reg:

$$S_{mj} = \frac{N_{common}\rho_{mj}}{N_{common} + reg} \tag{6.7}$$

where N_{common} is the common reviewer support and ρ_{mj} is the Pearson co-relation coefficient.

Once it is all set up, we can use the k-nearest neighbor algorithm to get top matches and the top recommendations for users.

The above system has a few limitations. For example, it is hard to judge if the recommendations are any good if we split the data into train and test sets for the usual testing. The best I could do is to compare the average rating of restaurant B in the training set to the rating of restaurant B in the test set.

So to tackle this issue we can refine the algorithm by shifting the focus to more fine-grained predictions about each user, and try to predict what rating a user would give to a restaurant they have never tried before. To do this, we can only pool information from restaurants that the user has rated. This can create other problems like a new-item problem or a cold start problem, or in the case when there

are very few reviewers of a restaurant or very few reviews by a user, respectively. Thus we can modify the k-nearest neighbor algorithm to find the nearest k neighbors to a given restaurant from the restaurants that the user has already rated, find predicted ratings for users and items, and then check the top recommendations again just to verify the results.

6.2.5 Deep Neural Network

Since we have already implemented a baseline content based model. We can use the previously generated one hot encoding feature vectors to train the model. In order to make use of the review text, use the Word2Vec model [19].

6.2.5.1 Word2Vec

Due to further advances in Natural Language Processing with the help of Neural Networks, additional methods to represent text such as Word2Vec and Paragraph Vectors have been created. These representations perform better than previous techniques such as Bag of Words (BoW) and Term Frequency Inverse Document Frequence (TF-IDF) at many semantic relatedness tasks.

Word2Vec is a word embedding technique, where the words from a large corpus are represented as feature vectors. This utilizes Neural Networks with a single hidden layer. The task of the network is to predict a word, given the context of the word. The input layer of this neural network is the one-hot encoding of the context words (the words that surround the current word), and the output layer is the vector for

the word under consideration.

During training, the algorithm parses through a large corpus of text, and for each occurrence of a word the weights of the neural network are updated using gradient descent. After multiple iterations through the corpus, the training converges, and the hidden layers after convergence represent the feature vectors for the words in the output layer.

The advantage of this technique compared to BoW is that similar words have feature vectors which are closer to each other in the vector space. This technique can also be used for answering analogy questions, which demonstrates the ability of the feature vectors in capturing the semantics of the words.

6.2.5.2 Keyword Extraction

Once all the documents in the corpus are represented as feature vectors, keyword extraction is done using document similarity. Consider document corpus D, the list of all tags in the corpus K, and a query document q. Let F be the Paragraph Vector representations of all the documents from D obtained after training the network.

We obtain the keyword suggestions for query document q as follows:

- We first perform the inference step on the network and obtain the paragraph vector representation for q.
- We iterate through all the documents in the corpus, and find the similarity of the query document's vector with the paragraph vector of each document in the corpus.

- For each document d' in the corpus, the similarity score with the query vector doc-sim(q,d) is added to the list Lk of similarities for all the keywords for the corresponding document.

- After we iterate through the complete corpus, we have a list of similarity scores for each keyword k' in the keyword list K. For each keyword k' we consider the maximum (/sum/avg) of all the entries in the list Lk' of similarities for k'.

- To finally obtain the keyword suggestions, we consider the top keywords based on the similarity score from step 4.

After training the deep neural network with doc2vec, a list of strings is generated with the highest weighted words which then is used as a feature vector. Then we refine the users with the highest review count and create weighted profiles for each refined user. This profile uses 80% of the restaurants they reviewed.

To better understand, consider 1000 businesses in total. And each user has reviewed 100 businesses. Then for each user, the profile uses only 80% of the businesses he/she reviewed. After building the profile matrix, the cosine similarities of the constructed user profiles with the remaining (other than the 80%) businesses plus the 20% of the businesses left out during profile construction are computed and the businesses are ordered in descending order of cosine similarity. The cosine similarity values obtained explain how similar a restaurant's characteristics are to a particular user from which we can infer the relationship between recommendations and cosine similarity i.e., higher cosine similarities lead to better recommendations.

After computing the cosine similarities, the remaining businesses reviewed by the

user that were not considered in constructing the profiles are ranked. From this ranking, we can understand that there is an inverse relation between rank and cosine similarities.

Once all the feature vectors are set, profiles created and similarities and ranks calculated, the model can be trained to get recommendations and top matches.

Chapter 7

Results

7.1 Discussion

7.1.1 Collaborative Filtering Results

One significant thing to observe in the CF models is that all of them gave good predictions. But knowing how well the model the models performed is essential, so evaluating their performance with an error metric is important. For CF, I used the RMSE error metric and as seen in Table 7.1, out of all the three models, the baseline performed the worst (as expected). From the definition of RMSE the lower the RMSE value the better the model. Both the user-user and item-item neighborhood models performed well.

While the results are promising, it is important to know why and how the model performed that way and what those values actually mean?

We got an RMSE of 1.259 for the baseline model. Since I used only mean ratings, one of the reasons for such bad RMSE could be that the data used for the baseline system was sparse, and notably had outliers which brought down the averages. One possible solution to this problem would be to assign weights to the ratings instead of just taking the mean.

Now for the user-user k-neighborhood method, this model gave a RMSE value of

1.194 which is slightly better than the baseline model but not the best performer. As discussed earlier, some of the issues with this model are at times there are users with no ratings history to predict, or sometimes users whose neighbors had never rated the restaurant for which we want to predict. It also suffers from scalability problems as the number of users increases.

As expected, the item-item k-neighborhood model performed better than the other models. We got an RMSE value of 1.173 which is a marked improvement to the user-user k-neighborhood model.

This is because the item-item model takes care of several crucial problems. Namely, the search for similar users can be computationally more expensive which is taken care of in the item-item neighborhood because it considers the problem of new user profiles having comparatively fewer ratings for businesses. However, sparseness of the matrix is still a recurring challenge.

Algorithm	**RMSE**
Baseline Collaborative Filtering	1.259
User-User KNN	1.194
Item-Item KNN	1.173

Table 7.1: Collaborative Filtering Results

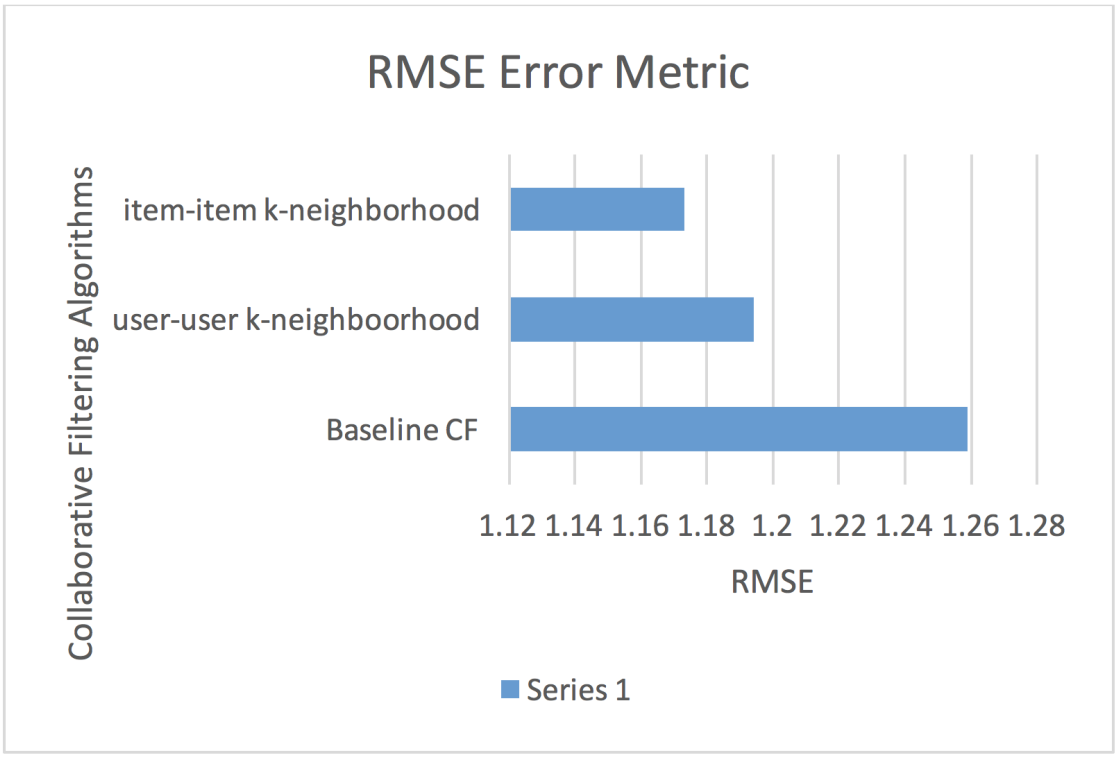

Figure 7.1: RMSE Error Metric

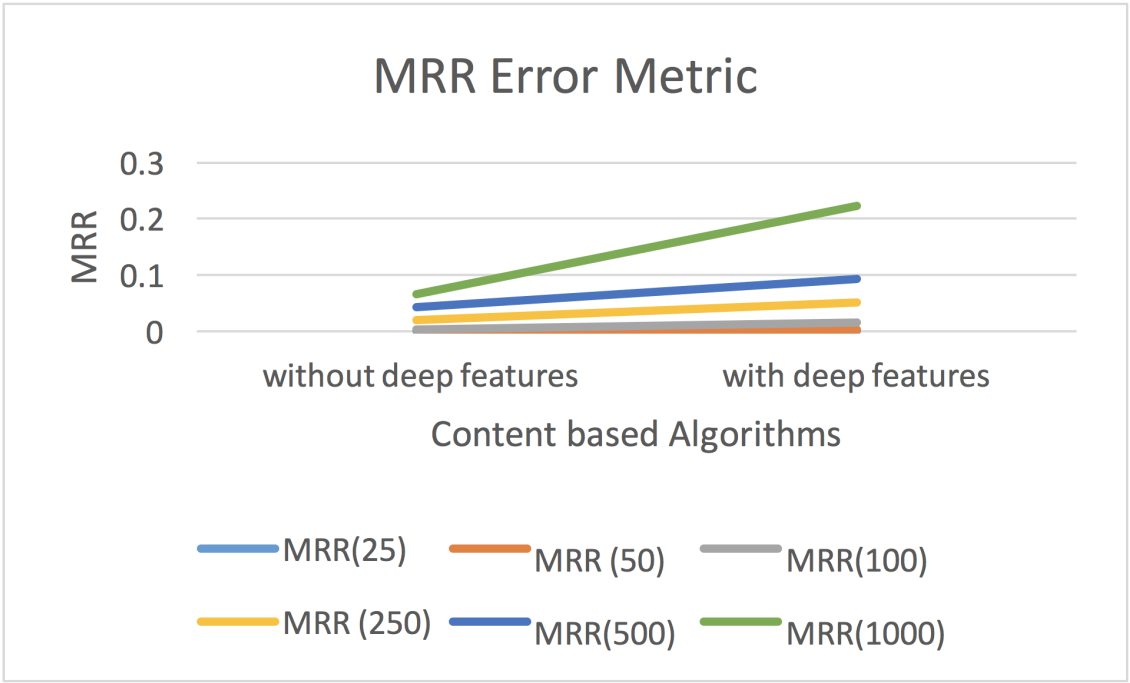

Figure 7.2: MRR Error Metric

Average MRR for number of users:	**25**	**50**	**100**	**250**	**500**	**1000**
Without deep features	0.0013368	0.0015962	0.01569	0.01674	0.0212	0.02376
With deep features	0.0013476	0.001724	0.01193	0.03641	0.04112	0.1315

Table 7.2: Content Based Results

7.1.2 Content Based Results

We have two content based models implemented, one with the deep features and the other without any deep features which we considered as our baseline.

Since we have been successful in getting good recommendations using both the models, they need to be evaluated to test how good the recommendations are. We can use the MRR metric to test the performance of each user and then take the average.

As seen in Formula 6.3, the reciprocal value of the mean reciprocal rank corresponds to the harmonic mean of the ranks and is one of the most commonly used error metrics for recommendation systems.

As expected, the deep neural network outperformed the shallow machine learning model by a good MRR score. All the observations listed in Table 7.2 display the comparison of average MRR score for linearly increasing users in the dataset. By looking at these values we can see that the DNN performed better. The possible reasons for such a good MRR score are as follows. One is better feature engineering using all the review data when integrated with the baseline. Whereas the baseline system completely ignored the reviews and focused just on average review score.

One of the issues for why the machine learning model did not work well is sparsity of the data. Apparently the sparsity of the data can affect any kind of model whether its a collaborative filtering based or content based. Due to this, the ratio of the number of users who gave a 5 star rating to the total number of users would definitely be low and finding such users can be an expensive task leading to a significant drop in cosine similarities for the model. Since the similarities are inversely related to ranks, they resulted in a higher rank for their recommendations. The model can therefore be improved by including more features. Whereas the deep neural network utilized the features of review text to identify the most common users with higher review count, and eliminated the problem of sparsity.

Figure 7.2 shows the performance of each model with increase in the number of users.

Chapter 8

Conclusion

In this project, I present several collaborative filtering and content based algorithms for recommendation system and tested the performance of each algorithm on part of the Yelp data. Finally, I have achieved a good RMSE score of 1.173 for the item-item k-neighborhood method which is a significant improvement over the baseline model.

Also, the DNN has achieved exceptional results as expected over the shallow machine learning model with a MRR score of 0.1315. The model performs well for a small subset of data from a single state with high review counts. And by looking at the results, we can infer that the DNN can perform better than the baseline systems as the number of users increases.

Although there is no common ground to evaluate content and collaborative models together, all the algorithms evaluated were able to make decent recommendations individually. Besides the performance of the models, the entire project has been a huge learning experience for me.

Chapter 9

Future work

As to the future work, it would be interesting to augment the current analysis with the 200,000 pictures and user rating evaluations (whether other users thought a particular users review was funny, useful, or helpful) as features in the prediction model. Also exploring CDL (collaborative deep learning) approaches (similar to the work in [1]) can be an improvement to the current recommendation system.

Bibliography

[1] Hao Wang, Naiyan Wang, Dit-Yan Yeung *Collaborative Deep Learning for Recommender Systems* (18 Jun 2015).

[2] Heng-Tze Cheng et al., *Wide & Deep Learning for Recommender Systems* (24 Jun 2016).

[3] Yelp Dataset Challenge, *https://www.yelp.com/dataset_challenge/* (2014).

[4] Yelp Business Insights, *http://people.ischool.berkeley.edu/ sayantan.satpati/yelp/#portfolioModal0.*

[5] Yves Chauvin and David E Rumelhart *Backpropagation: theory, architectures, and applications.* (Psychology Press, 1995).

[6] Geoffrey Hinton et al., *Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups.* (Processing Magazine, IEEE, 29 (6):8297, 2012).

[7] Gunawardana A., Shani G *Evaluating Recommendation Systems.*

[8] GOPAL RANDER, *Business Categorization using Yelp Reviews and Tips.*

[9] *https://github.com/lchiaying/yelp.*

[10] Chee Hoon Ha, *Yelp Recommendation System Using Advanced Collaborative Filtering.*

[11] Rahul Makhijani, Saleh Samaneh, Megh Mehta, *Collaborative Filtering Recommender Systems.*

[12] Sumedh Sawant and Gina Pai, *Yelp Food Recommendation System.*

[13] Zheng Wen, *Recommendation System Based on Collaborative Filtering* (December 12, 2008).

[14] *https://en.wikipedia.org/wiki/Recommender_system#Hybrid_recommender_systems.*

[15] *https://gist.github.com/henryachen/a5fb0281c1715e83aaef.*

[16] Ramesh Dommeti *Neighborhood based methods for Collaborative Filtering.*

[17] *https://www.doc.ic.ac.uk/ nd/surprise_96/journal/vol4/cs11/report.html #What%20is%20a%20Neural%20Network.*

[18] *https://en.wikipedia.org/wiki/Mean_reciprocal_rank.*

[19] *https://web.stanford.edu/ jurafsky/slp3/16.pdf.*