University of California

Los Angeles

Application of machine learning techniques to recommendation system

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Statistics

by

Mei Gao

2015

Abstract of the Thesis

Application of machine learning techniques to recommendation system

by

Mei Gao

Master of Science in Statistics

University of California, Los Angeles, 2015

Professor Yingnian Wu, Chair

In this paper, I applied several machine learning techniques, including Latent Dirichlet allocation (LDA), deep Convolutional Neural Networks (CNN) as well as Ranking SVM to build a restaurant recommendation system using the Yelp Dataset. The idea is that, compared to algorithms that are previously commonly used in recommendation system, i.e. Pearson similarity and clustering algorithms, the application of machine learning techniques such as LDA, CNN and SVM to recommendation has been a new area and not systematically studied yet. Topic models such as LDA could allow us to learn the latent subtopics in review texts, which then can be used in personalized recommendation. Deep CNN could be used to extract the underlying features from food images and combined with Ranking SVM to provide users with the best food image of a restaurant. In this paper, I made several attempts to introduce these machine learning techniques to the construction of a restaurant recommender engine and results have shown that the efficiency of the recommendation system could be largely improved with these techniques.

The thesis of Mei Gao is approved.

Hongquan Xu

Qing Zhou

Yingnian Wu, Committee Chair

University of California, Los Angeles

2015

To my parents . . .

who have been with me every step of the way

through good times and bad

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction to recommendation system

The goal of a Recommender System is to generate meaningful recommendations to a collection of users for items or products that might interest them [[MS10]]. Real world examples include suggestions for books on Amazon, movies on Netflix, and news articles on Google News. As Chris Anderson mentioned in his book "The long tail" [[And13]], we are leaving the age of information and entering the age of recommendation. Information is severely overloaded at this time. People read around 10 MB worth of material a day, hear 400 MB a day, and see 1 MB of information every second. Rather than providing a static experience in which users search all these information for something they will potentially buy, recommender systems identify recommendations autonomously for individual users based on past purchases and searches, and on other users' behavior. As examples, 2/3 of the movies watched on Netflix are recommended, recommendations on Google News generate 38% more clickthrough and 35% sales on Amazon are from recommendations.

In a recommendation system, there are two classes of entities, which are users and items. Users have preferences for certain items. What we need to do with a recommendation system is to extract the preference information from the data. The data itself is represented as a utility matrix, as shown in table 1. Given each user−item pair, a value in the utility matrix represents what is known about the degree of preference of that user for that item. Values may come from an ordered set, e.g., integers 1−5 representing the number of stars that the user gave as a rating for that item. This utility matrix is typically sparse, as most users do not rate most items. The goal here is to predict what rating a user would give to a previously unrated item. Typically, ratings are predicted for all items that have not been observed by

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Table 1: A utility matrix representing ratings of movies on a 1−5 scale. Blanks represent the situation where the user has not rated the movie. The movie names are HP1, HP2, and HP3 for Harry Potter I, II, and III, TW for Twilight, and SW1, SW2, and SW3 for Star Wars episodes 1, 2, and 3. The users are represented by capital letters A through D.

a user, and the highest rated items would be recommended to the user.

Most recommender systems take three basic approaches: collaborative filtering, content-based filtering and hybrid method.

- **Collaborative Filtering (CF):** Items are recommended to users based on the past ratings of all users collectively.
- **Content-based recommending:** Items recommended to users are those that are similar in content to items the user has liked in the past, or matched to attributes of the user.
- **Hybrid approaches:** These methods combine both collaborative and content−based approaches.

1.1 Collaborative Filtering

In a collaborative filtering system, recommendation are made based on a model of prior user behavior. The model can either be constructed solely from a single user's behavior or from the behavior of other users who have similar traits. Usually the latter approach is more effective. When it takes other users' behavior into consideration, collaborative filtering uses group knowledge to form a recommendation based on users. In essence, recommendations are based on an automatic collaboration of multiple users and filtered on those who exhibit similar preferences or behaviors.

For example, suppose you're building a website to recommend movies. By using the information from many users who have rated different movies, you can group those users based on their preferences. For example, you can group together users who gave similar rating to the same movies. From this information, you identify the highest rated movies that are rated by that group. Then, for a particular user in the group, you can recommend the highest rated movies that he or she has never rated or watched before.

1.2 Content-based recommendation

Content-based approach is based on a user's behavior. For example, this approach might use historical information, such the ratings given by the user to the movies he or she previously watched. If a user gives high ratings to most of the action movies he watches, then content-based filtering can use this history to identify and recommend movies with similar content (action). This content can be manually defined or automatically extracted based on other similarity methods.

1.3 Hybrid

Hybrid approaches combine collaborative and content-based filtering and help to increase the efficiency (and complexity) of recommendation systems. Incorporating the results of collaborative and content-based filtering creates the potential for a more accurate recommendation. The hybrid approach could also be used to address collaborative filtering that starts with sparse data− known as cold start− by enabling the results to be weighted initially toward content-based filtering, then shifting the weight toward collaborative filtering as the available user dataset matures.

CHAPTER 2

Algorithms in recommendation systems

Many algorithmic approaches are available for recommendation engines. The design of such recommendation engines depends on the domain and the particular characteristics of the data available. Therefore results can differ based on the problem and the algorithm that is designed to find the relationships present in the data. In this chapter I will firstly discuss some of the previously commonly used algorithms in recommender engine, and then I will introduce some of the machine learning techniques that I applied to the restaurant recommendation system which will be discussed in details in next chapter.

2.1 Previously used algorithms

2.1.1 Pearson Similarity

This method computes the statistical correlation (Pearsons r) between two users common ratings to determine their similarity [[ERK11]]. GroupLens and BellCore both used this method [[HSR95], [RIS94]]. The correlation is computed by the following equation:

$$s(u,v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \overline{r}_u)(r_{v,i} - \overline{r}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{u,i} - \overline{r}_u)^2} \sqrt{\sum_{i \in I_u \cap I_v} (r_{v,i} - \overline{r}_v)^2}} \tag{2.1}$$

Pearson correlation suffers from computing high similarity between users with few ratings in common. This can be alleviated by setting a threshold on the number of co-rated items necessary for full agreement (correlation of 1) and scaling the similarity when the number of co-rated items falls below this threshold [[HKR02], [HKB99]].

2.1.2 Clustering algorithms

Clustering algorithms are a form of unsupervised learning that can find structure in a set of seemingly random (or unlabeled) data. In general, they work by identifying similarities among items, such as movies, by calculating their distance from other items in a feature space. The number of independent features defines the dimensionality of the space. If items are "close" together, they can be joined in a cluster.

Many clustering algorithms exist. The simplest one is k-means, which partitions items into k clusters. Initially, the items are randomly placed into clusters. Then, a centroid (or center) is calculated for each cluster as a function of its members. Each item's distance from the centroids is then checked. If an item is found to be closer to another cluster, it's moved to that cluster. Centroids are recalculated each time all item distances are checked. When stability is reached (that is, when no items move during an iteration), the set is properly clustered, and the algorithm ends. Calculating the distance between two objects can be difficult to visualize. One common method is to treat each item as a multidimensional vector and calculate the distance by using the Euclidean algorithm.

2.1.3 Other algorithms

Besides Pearson similarity and clustering algorithms, there exist many other algorithms for recommendation engines. Some that have been used successfully include:

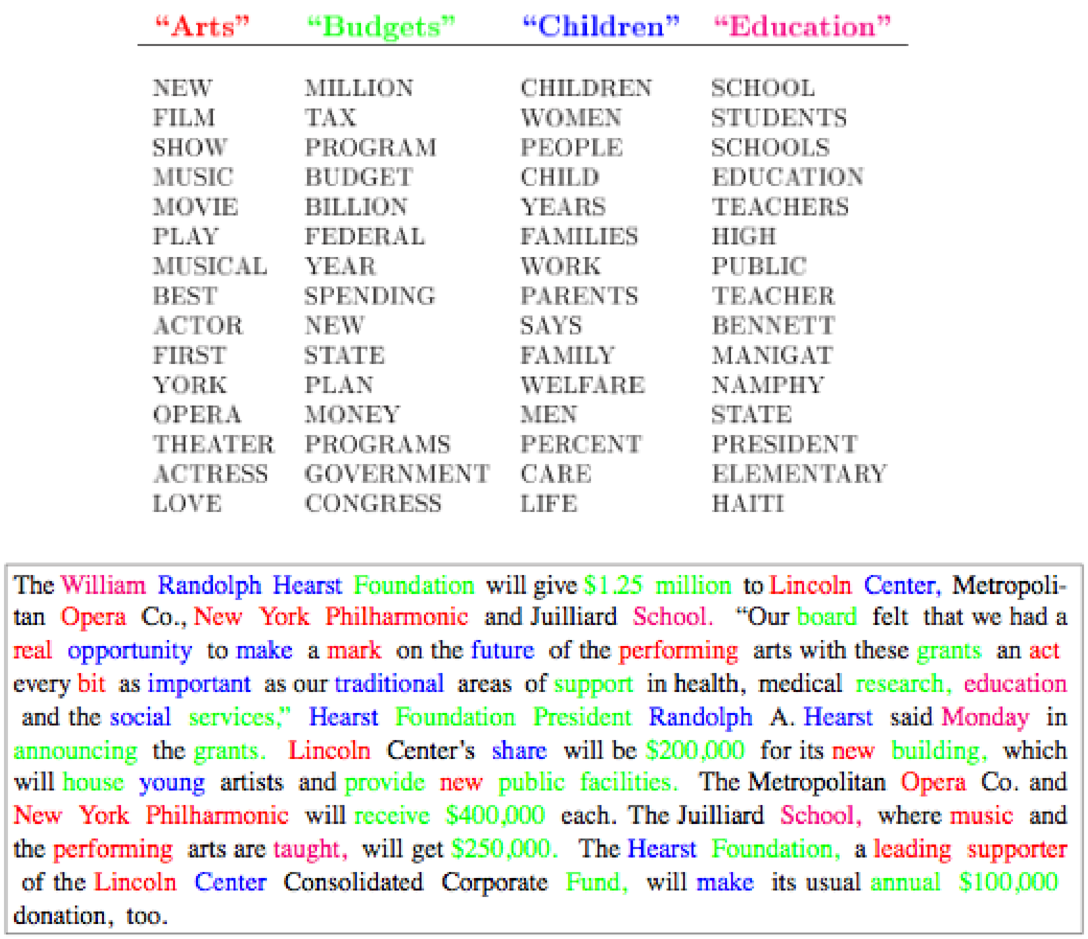
- **Bayesian Belief Nets,** which can be visualized as a directed acyclic graph, with arcs representing the associated probabilities among the variables [[Kri01]].
- **Markov chains,** which take a similar approach to Bayesian Belief Nets but treat the recommendation problem as sequential optimization instead of simply prediction.

- **Rocchio classification,** which exploits feedback of the item relevance to improve recommendation accuracy [[Joa96]].

2.2 Introduction to Topic Modeling using LDA

Probabilistic topic modeling is a suite of algorithms that aim to discover and annotate large archives of documents with thematic information [[Ble12]]. For example, consider using themes to explore the content of the New York Times. At a broad level, some of the themes might correspond to the sections of the newspaper − foreign policy, national affairs, sports. We could zoom in on a theme of interest, such as foreign policy, to reveal various aspects of it − Chinese foreign policy, the conflict in the Middle East, the U.S. relationship with Russia. In all of this exploration, we would be pointed to the original articles relevant to the themes. The thematic structure would be a new kind of window through which to explore and digest the collection. Topic modeling algorithms are statistical methods that analyze the words of the original texts to discover the themes that run through them and how those themes are connected to each other. Topic modeling algorithms do not require any prior annotations or labeling of the documents − the topics emerge from the analysis of the original texts. Topic modeling enables us to organize and summarize electronic archives at a scale that would be impossible by human annotation.

One of the simplest topic model is latent Dirichlet allocation (LDA), The intuition behind LDA is that documents exhibit multiple topics. An example is shown in figure 1. The bottom part is a document from the TREC AP corpus (`http://research.nii.ac.jp/ntcir/ntcir-ws2/OnlineProceedings/Keynote-Harman.pdf`). Words used in this article are highlighted in different color. Words about 'Arts' are highlighted in red, 'Budgets' in green, 'Children' in blue and 'Education' in magenta. Therefore we can see that this article blends 'Atrs','Budgets', 'Children' and 'Education' in different proportions. LDA is a statistical model of document collections that tries to capture this intuition. It is most



"Arts"	"Budgets"	"Children"	"Education"
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

Figure 1: An example article from the AP corpus. Each color codes a different factor from which the word is putatively generated.

easily described by its generative process, the imaginary random process by which the model assumes the documents arose.

We formally define a topic to be a distribution over a fixed vocabulary. We assume that these topics are specified before any data has been generated [[Ble12]]. Now for each document in the collection, we generate the words in a two-stage process.

- Randomly choose a distribution over topics.

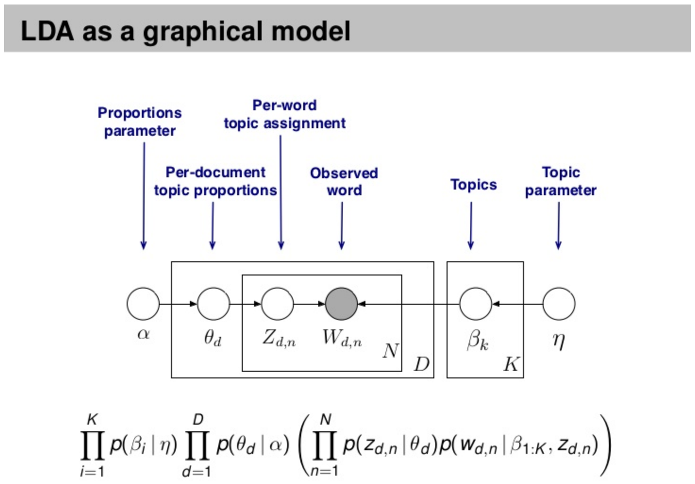
- For each word in the document

a. Randomly choose a topic from the distribution over topics in step #1.

b. Randomly choose a word from the corresponding distribution over the vocabulary.

This statistical model reflects the intuition that documents exhibit multiple topics. Each document exhibits the topics in different proportion (step #1); each word in each document is drawn from one of the topics (step #2b), where the selected topic is chosen from the per-document distribution over topics (step #2a). As we described previously, the goal of topic modeling is to automatically discover the topics from a collection of documents. The documents themselves are observed, while the topic structurethe topics, per-document topic distributions, and the per-document per-word topic assignmentsis hidden structure. The central computational problem for topic modeling is to use the observed documents to infer the hidden topic structure.

In LDA, let K be a specified number of topics, V be the size of the vocabulary, $\vec{\alpha}$ positive K-vector, and η a scalar. We let $Dir_V(\vec{\alpha})$ denote a V-dimensional Dirichlet with vector parameter $\vec{\alpha}$ and $Dir_K(\eta)$ denote a K dimensional symmetric Dirichlet with scalar parameter η.



$$\prod_{i=1}^{K} p(\beta_i \mid \eta) \prod_{d=1}^{D} p(\theta_d \mid \alpha) \left(\prod_{n=1}^{N} p(z_{d,n} \mid \theta_d) p(w_{d,n} \mid \beta_{1:K}, z_{d,n}) \right)$$

Figure 2: A graphical model representation of the latent Dirichlet allocation (LDA).

- For each topic, draw a distribution over words $\vec{\beta}_k \sim Dir_K(\eta)$
- For each document,

  (a) Draw a vector of topic proportions $\vec{\theta}_d \sim Dir_V(\vec{\alpha})$

  (b) For each word,

    (1) Draw a topic assignment $Z_{d,n} \sim Mult(\vec{\theta}_d)$, $Z_{d,n} \in \{1, ..., K\}$.

    (2) Draw a word $W_{d,n} \sim Mult(\vec{\beta}_{Z_{d,n}})$, $W_{d,n} \in \{1, ..., V\}$.

This is illustrated as a directed graphical model in figure 2. The joint distribution of the hidden and observed variables is also shown at the bottom of the figure.

2.3 Introduction to deep learning in image understanding

Since their introduction by [[LBD89]] in the early 1990s, Convolutional Networks (convnets) have demonstrated excellent performance at tasks such as hand-written digit classification, face detection as well as more challenging visual classification tasks [[KSH12], [SEZ13], [SZ14]]. Features extracted from images using convolutional neural networks are proved to be be far from random, uninterpretable patterns [[ZF14]]. Rather, they show many intuitively desirable properties such as compositionality, increasing invariance and class discrimination as we ascend the layers. Features extracted using convolutional neural networks increase the non-linear operation and thus can learn more detailed information and common structure of images. Therefore these features can be used to obtain better results in image classification.

In this study, I used the convolutional networks model developed by the Visual Geometry Group from the university of Oxford (`http://www.robots.ox.ac.uk/~vgg/`), which is the basis of their ImageNet Challenge 2014 submission. Their team secured the first and the second places in the localization and classification tracks respectively. They found that a significant improvement on the prior-art configurations can be achieved by increasing the depth to 16-19 weight layers, which is substantially deeper than what has been used in the

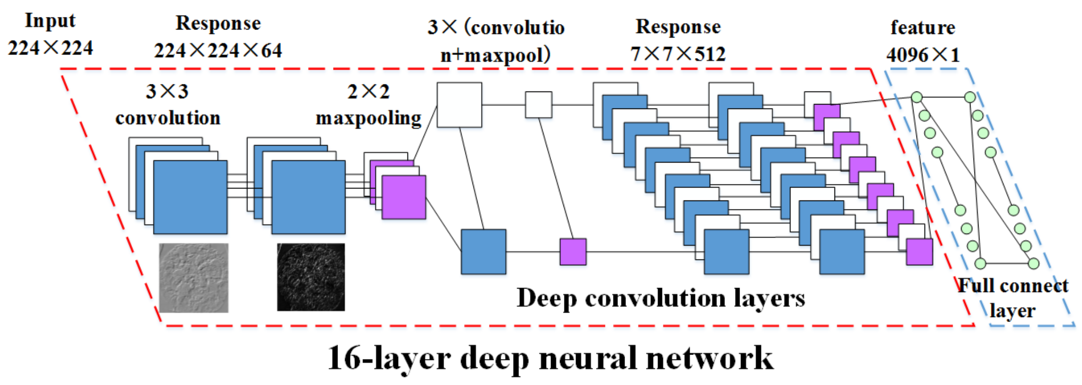


Figure 3: Architecture of the 16 layer convnet model.

prior art. To reduce the number of parameters in such very deep networks, they use very small 3×3 filters in all convolutional layers (the convolution stride is set to 1). The following figure 3 shows the architecture of the model.

First of all, the training images are resized into 224×224 and subtracted by mean value so that the mean of image intensity is zero. At each layer, the filter size is fixed as 3×3. For layer 1, there are 64 filters. To keep response map having the same resolution as 224×224, one pixel is added to response map to keep 224×224. Thus the response map is 224×224×64. For layer 2, there are also 64 filters, and 224×224×64 response map. Layer 3 is max-pooling layer. So 2×2 sub-sampling (max response in a 2×2 area is picked) is applied to response map. After max-pooling, response map is 112×112×64. Both Layer 4 and layer 5 have 128 filters and response map is 112×112×128. Layer 6 is max-pooling and response map is 56×56×128. Both layer 7 and layer 8 have 256 filters. So the response map is 112×112×256. Layer 9 is a little different, it has 256 1×1 filters. Layer 10 is max-pooling layer and response map is 28×28×256. Layer 11 and layer 12 have 512 filters and response map is 28×28×256 and response map is 28×28×512. Layer 13 has 512×1×1 filters. Layer 14 is max-pooling layer and response map is 14×14×512. Layer 15 and layer 16 have 512 filters and response map is 14×14×512. Filter 17 has 512×1×1 filters. Layer 18 is max-pooling layer and response map is 7×7×512. Finally the 7×7×512 response is connected to two layers of 4096 nodes to generate 4096-dimensional features.

It's important to understand what we can learn by applying the deep Convolutional Neural Networks (CNN). Suppose all natural images are distributed in an unknown high-dimension manifold space. Here we want to learn the underling representation of images with different categories. It is well known that each category of images lies on a very thin spike of manifold (You can imagine we have a finite number of categories but represented by 4096 dimension space). Learning such a manifold is very difficult. You can also imagine the possible distribution of observed images with unknown structure is like a haystack, and each spike of underling distribution is like a small needle. As a metaphor, learning representation of images is as to look for a needle in the haystack. However, with deep hierarchical abstraction and non-linear transformation, deep CNN can learn such a representation by grouping shared structure. Therefore deep learning is usually defined as representation learning. The learned underling representation could provide more discriminative features for ranking to provide better recommendation. This will be further explained in the following sections.

2.4 Introduction to ranking SVM

A Ranking SVM is an application of Support Vector Machine, which is used to solve certain ranking problems. The algorithm of ranking SVM was published by [[Joa02]]. In ranking SVM, the data points generated by the training data are in the feature space, which also carry the rank information (the labels). These labeled points can be used to find the boundary (classifier) that specifies the order of them. In the linear case, such boundary (classifier) is a vector. Suppose c_i and c_j are two elements in the database and denote $(c_i, c_j) \in r$ if the rank of c_i is higher than c_j in certain ranking method r. Let vector $\vec{w}$ be the linear classifier candidate in the feature space. Then the ranking problem can be translated to the following SVM classification problem.

minimize:

$$V\left(\vec{w},\vec{\xi}\right)=\frac{1}{2}\vec{w}\cdot\vec{w}+C\sum\xi_{i,j} \tag{2.2}$$

s.t.

$$\forall\xi_{i,j}\geq 0,\forall c_i>c_j \tag{2.3}$$

$$\vec{w}\left(x_i-x_j\right)\geq 1-\xi_{i,j} \tag{2.4}$$

C is a parameter that allows trading-off margin size against training error. Geometrically, the margin δ is the distance between the closest two projections within all target rankings. The algorithm is illustrated in the following figure 4. It shows an example of how the weight vector $\vec{w}$ determines the ordering of four points in a two dimensional space. For any weight vector $\vec{w}$, the points are ordered by their projection onto $\vec{w}$ (or, equivalently, by their signed distance to a hyperplane with normal vector $\vec{w}$). This means that for $\vec{w}_1$ the points are ordered (1, 2, 3, 4), while $\vec{w}_2$ implies the ordering (2, 3, 1, 4). The optimization problem is very similar to the SVM model, except that the slack variable now is associated with both point i and point j. And the constrain is that the difference in score calculated for i and j should be larger than 1 minus the slack variable. Slack variable is used to relax the constrain that the data has to be linearly separable.

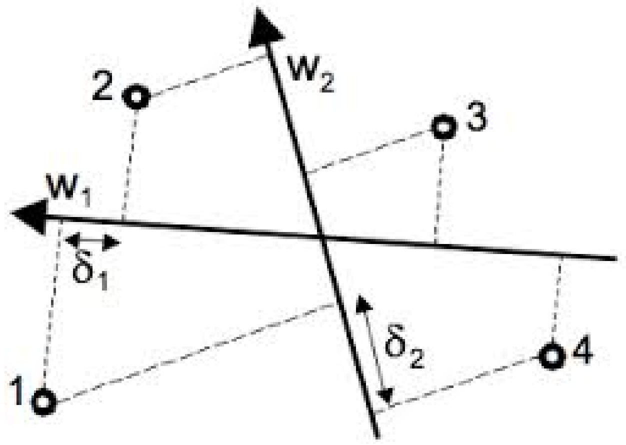


Figure 4: Example of how two weight vectors $\vec{w}1$ and $\vec{w}2$ rank four points.

CHAPTER 3

Application of machine learning to Yelp restaurant recommendation

3.1 Introduction to the restaurant recommendation system

One of the problems that bothers a lot of people including myself on a daily basis is that, very often we don't know what and where to eat. Even with the Yelp App, we simply don't know what to look for. Most of the recommendation systems make suggestions based on users historical behavior, such as restaurants they have been to or have written reviews for. However, since people's preferences for food change all the time, it is sometimes very tricky to build a restaurant recommender based on historical data. What a user whats to eat today may not be what he or she wants to eat tomorrow. Therefore I'd like to build a recommender system which is based on whatever that's going on in the user's mind at the moment when he or she tries to make a decision, instead of on his or her historical behavior. I built a website called ”imfeelinghungryy” (`imfeelinghungryy.com`), which is a Python based web application that provides personalized restaurant recommendations without requiring historical data from user. Since the monthly cost to maintain and deploy the website on AWS (Amazon Web Services, `https://aws.amazon.com/`) is not trivial, the web application is not active currently. But in this paper, I made screen shots of the website to show readers the appearance and functions of this on-line restaurant recommendation application.

Before users use the website, they've no idea what to eat and where to go. After they click on the site, ten words would show up (figure 5). Each of the word is associated with

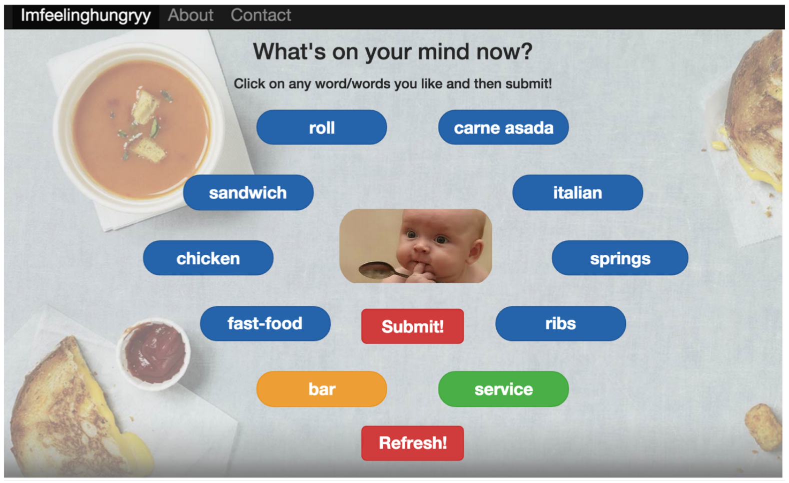
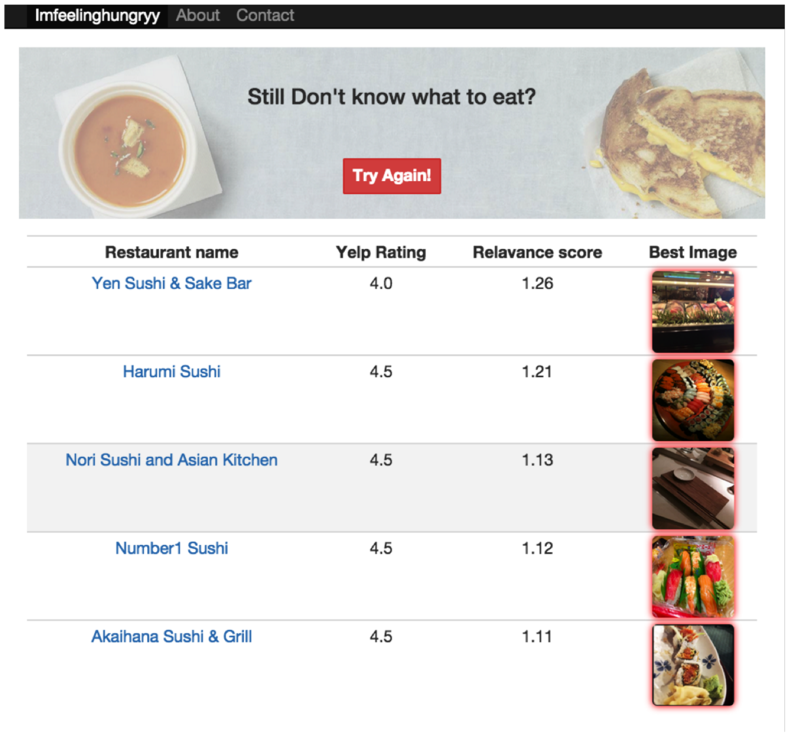


Figure 5: Users can choose whatever topics they are interested in or combine multiple topics together

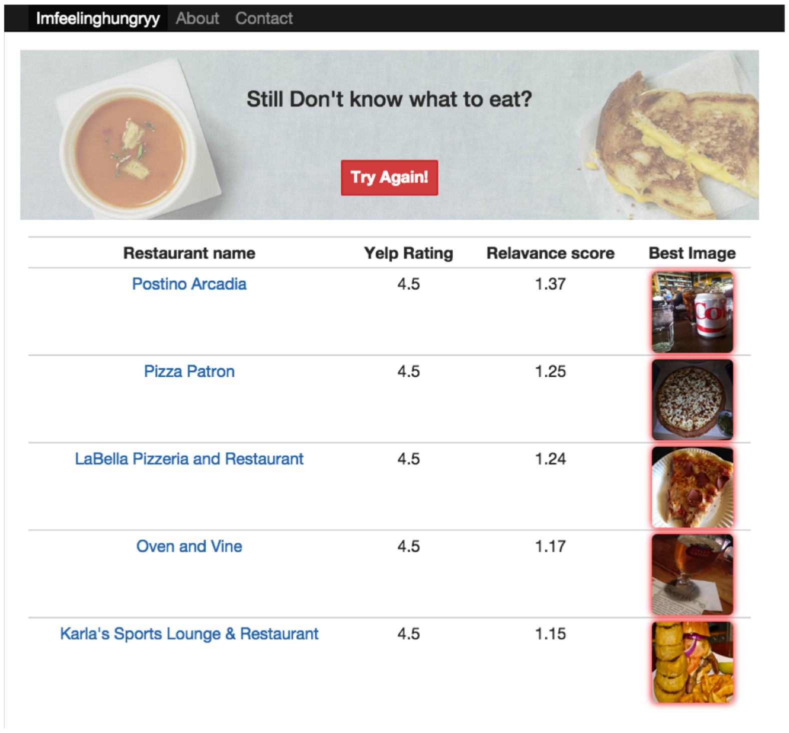


Restaurant name	Yelp Rating	Relavance score	Best Image
Yen Sushi & Sake Bar	4.0	1.26	
Harumi Sushi	4.5	1.21	
Nori Sushi and Asian Kitchen	4.5	1.13	
Number1 Sushi	4.5	1.12	
Akaihana Sushi & Grill	4.5	1.11	

Figure 6: Recommendation system will return a list of most relevant restaurants as well as the best image for each restaurant

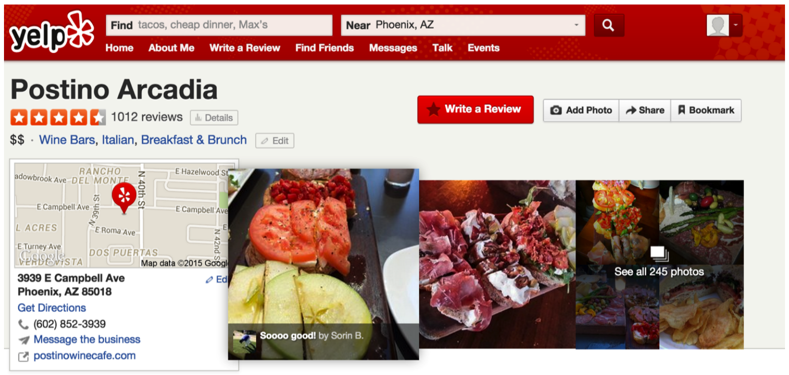
a specific topic generated from all the reviews using the LDA topic model. After they see these words, they may find, for instance, yeah sushi rolls sounds great today, then they can click on "rolls", and then they'll get a list of recommended restaurants, together with the Yelp rating, a relevance score and a best image for the restaurant chosen from all the images on Yelp (figure 6). For each restaurant, there could be hundreds of food images uploaded by Yelp users. The quality of these images very a lot. Helping the user to select the best food image for each restaurant significantly improves the visualization of the recommendation. It will not only help users to make a better decision but also improve the user experience of the web application. If we take a closer look at the recommended restaurants, we can easily find that all the five restaurants listed on the websites are Japanese restaurants with great sushi, which are exactly what the users are looking for. If they didn't like the recommendation, they can refresh the page and get a different group of words. They can also combine food categories with either drinks or quality of service. For example, they may want to eat some pizza and then go to a bar. They can then click on both of these topics. The recommendation system will then return a list of restaurants which could provide users with both categories. As shown in figure 7, if we take a detailed look at the first restaurant in the recommended list, Postino Arcadia (`http://www.yelp.com/biz/postino-arcadia-phoenix`), it is actually an Italian restaurant with great wine bars (figure 8).

As we can see from these results, the web application provides us with very reasonable restaurant recommendations to users. The following sections are structured as follows. First of all, I will briefly introduce the dataset used in building this web application. Then I will explain in details about how the different machine learning techniques are applied in the recommendation system as well as provide evidence and measures for the accuracy and efficiency of the recommender engine.



Restaurant name	Yelp Rating	Relavance score	Best Image
Postino Arcadia	4.5	1.37	
Pizza Patron	4.5	1.25	
LaBella Pizzeria and Restaurant	4.5	1.24	
Oven and Vine	4.5	1.17	
Karla's Sports Lounge & Restaurant	4.5	1.15	

Figure 7: An example of recommended restaurants when users choose topic "Roll"



Postino Arcadia

1012 reviews Details

$$ · Wine Bars, Italian, Breakfast & Brunch Edit

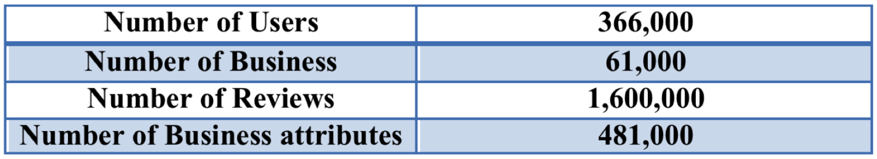
3939 E Campbell Ave
Phoenix, AZ 85018
Get Directions
(602) 852-3939
Message the business
postinowinecafe.com

Soooo good! by Sorin B.

Figure 8: An example of recommended restaurants when users combine topics "Italian" with "bar"

3.2 Introduction to Yelp Dataset

In this project I use the data from the Yelp dataset released by Yelp Inc. for the Yelp Dataset Challenge (`http://www.yelp.com/dataset_challenge`). It is a deep dataset introduced by Yelp for research-minded academics. It is richer and a lot larger than the previous released Academic Dataset. Table.2 summarizes the properties of the dataset. Here business includes a variety of different kinds of business types including restaurants, hospitals, retailer stores etc. In this paper I only focus on restaurants. Business attributes include open hours, parking availability, noise levels etc. for each business.

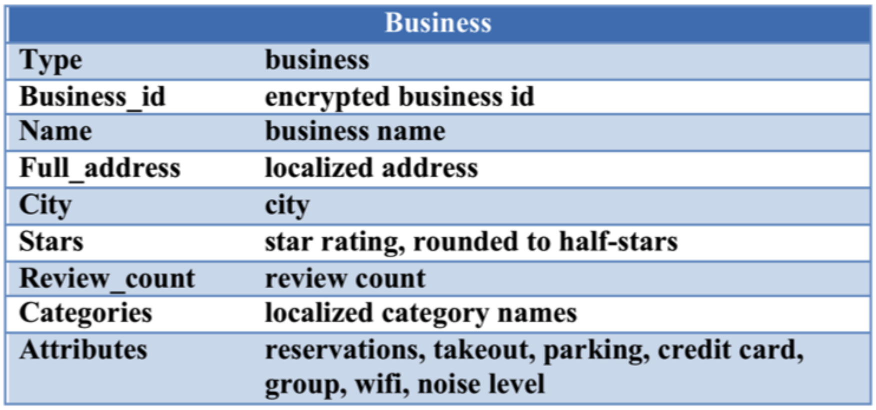


Number of Users	**366,000**
Number of Business	**61,000**
Number of Reviews	**1,600,000**
Number of Business attributes	**481,000**

Table 2: Properties of Yelp dataset.

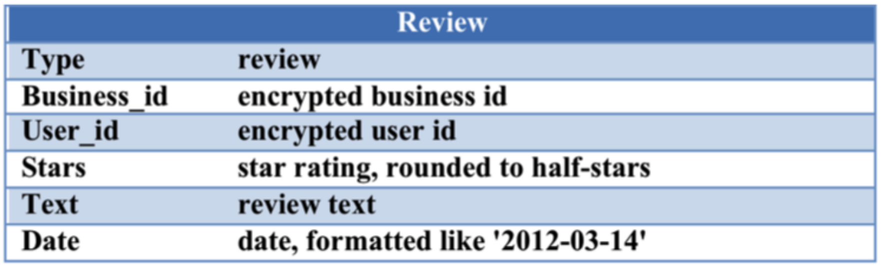
The data is stored in Jason format, and contains five different objects, which are business, review, user, tip and check-in. In this paper, only two objects that are most relevant are used, Business and Review. The following tables summarize the structure for the two objects. In the business object, I extract basic information of each restaurant, such as location, averaging stars and number of total review counts. In the review object, I extract information for each review, including who wrote the review (user id) and which restaurant the review is written for (business id) as well as the text information of the review.

I developed Python code to read the data from different objects. Due to the large size of the data, I stored the data in SQL database. I created two tables, called business and reviews, respectively. Business table stores the basic attributes (rating star) for each restaurant. Reviews table stores the reviews information for each restaurant. Because of time limit and the limit in the computational efficiency of my PC, I chose a subset of restaurants from the whole data set. I filtered the data by choosing only ”Open” restaurants (getting rid of the



Business	
Type	business
Business_id	encrypted business id
Name	business name
Full_address	localized address
City	city
Stars	star rating, rounded to half-stars
Review_count	review count
Categories	localized category names
Attributes	reservations, takeout, parking, credit card, group, wifi, noise level

Table 3: Structure of the business object.



Review	
Type	review
Business_id	encrypted business id
User_id	encrypted user id
Stars	star rating, rounded to half-stars
Text	review text
Date	date, formatted like '2012-03-14'

Table 4: Structure of the review object.

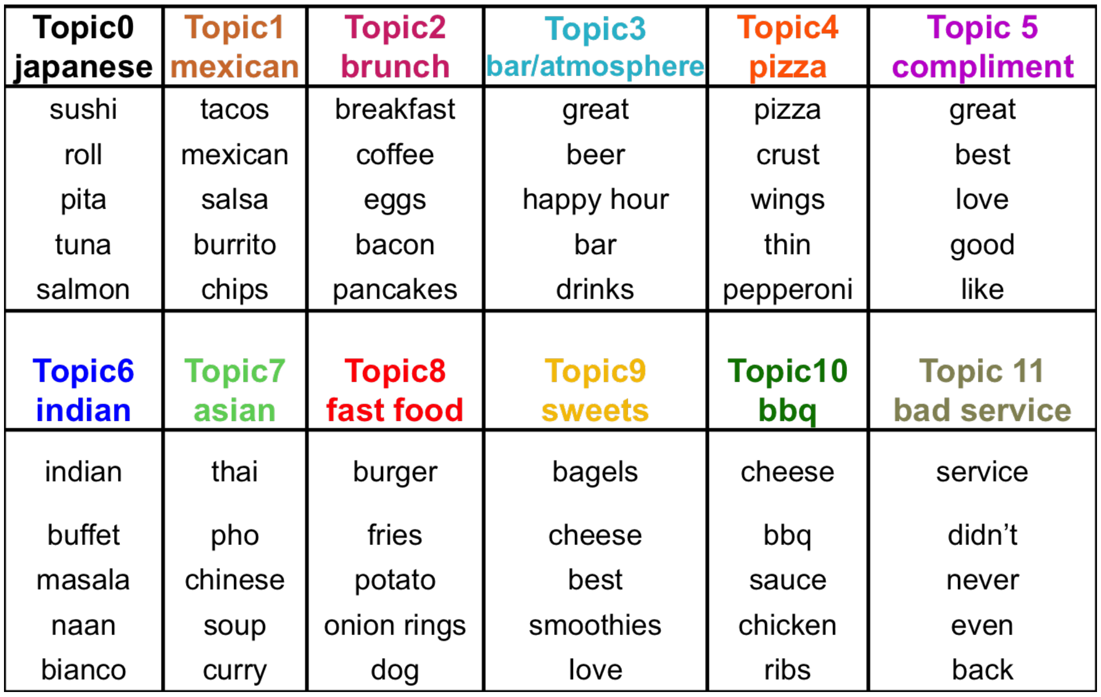
"Closed" ones) and choosing only restaurant with reviews (getting rid of restaurants with zero number of reviews). This yields to 1966 restaurants in the city of Phoenix, Arizona.

3.3 Finding latent topics in restaurant review texts using LDA

I first created a corpus using the reviews from all the 1966 restaurants in the city of Phoenix. The corpus was generated according to the following procedure. First of all, each review for each of the 1966 restaurants is tokenized, meaning after lowercasing each words, all the common words, numbers, as well as words that only appear once in all the reviews are removed. Then I converted each review to vectors. I use a document representation called bag-of-words. In this representation, each document is represented by one vector where each vector element represents a question-answer pair, in the style of: How many times does the word system appear in the document? Once. Thus I can create the corpus based on how frequent each word in the dictionary appear in each review for the 1966 restaurants. Finally I calculated the tf-idf (short for term frequency − inverse document frequency) value for the corpus. The tf-idf value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general. It can be calculated by multiplying a local component (term frequency) with a global component (inverse document frequency), and normalizing the resulting documents to unit length. Formula for unnormalized weight of term i in document j in a corpus of D documents:

$$\mathrm{weight_{i,j}} = \mathrm{frequency_{i,j}} * \mathrm{log\_2(D/document\_freq_i)} \tag{3.1}$$

Then I could fit the corpus_tfidf constructed from these reviews to the LDA model using the gensim python package (`https://radimrehurek.com/gensim/models/ldamodel.html`). The model would estimate LDA model parameters based on the corpus_tfidf and would return the word distribution for each topic. The following figure 9 shows the topics generated using the LDA model from the corpus constructed from restaurant reviews as well as the most frequent words that appear in each topic. When fitting the LDA model, it is important to decide the number of topics that generate the corpus. I experimented with different $n_{topic} \in \{1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 100, 200\}$. The fitness of each model could be evaluated by comparing the perplexity for each of them. Perplexity is a measurement of how well a probability distribution or probability model predicts a sample. It may be used to compare probability models. What I found is that after I increase the number of topics to more than 10, the perplexity of different models stay pretty much the same without large variations. I also found when the number of topics are too small, words that are describing different topics may be grouped to the same topic. While if the number of topics are set to be too large, the results are sometimes very hard to interpret and does not make much sense. In this case here, I decided to choose $n_{topic} = 15$.



Topic0 japanese	Topic1 mexican	Topic2 brunch	Topic3 bar/atmosphere	Topic4 pizza	Topic 5 compliment
sushi	tacos	breakfast	great	pizza	great
roll	mexican	coffee	beer	crust	best
pita	salsa	eggs	happy hour	wings	love
tuna	burrito	bacon	bar	thin	good
salmon	chips	pancakes	drinks	pepperoni	like
Topic6 indian	**Topic7 asian**	**Topic8 fast food**	**Topic9 sweets**	**Topic10 bbq**	**Topic 11 bad service**
indian	thai	burger	bagels	cheese	service
buffet	pho	fries	cheese	bbq	didn't
masala	chinese	potato	best	sauce	never
naan	soup	onion rings	smoothies	chicken	even
bianco	curry	dog	love	ribs	back

Figure 9: Topics generated from LDA model and the most frequent words in each topics.

Then I observed the results from the fitted model, which outputs the distribution of words in each topic. I manually gave a general category to each of the topic based on the most frequent words in it and deleted three topics that I could not interpret very well so as to define a general category. This left me with 12 topics in total, representing categories including Japanese, Mexican, brunch, bar/atmosphere, pizza, compliment, Indian, Asian, fast food, sweets, BBQ and services. The most commonly discussed topics are services and compliments, followed by BBQ and Mexican good. Intuitively, when people write reviews for restaurants, it is very likely that they would spend a lot of words commenting on the quality of service of the restaurants, or compliment on what they like about the restaurants. Considering that all the 1966 restaurants in our dataset are located in city of Phoenix, Arizona, the fact that topics such as BBQ and mexican food are commonly discussed is also easy to understand. Figure 10 shows the percentage distribution of these different topics, in both forms of a pie graph and a word cloud. Users should then choose among these topics the one that they are most interested in, or combine several topics together to make a choice.

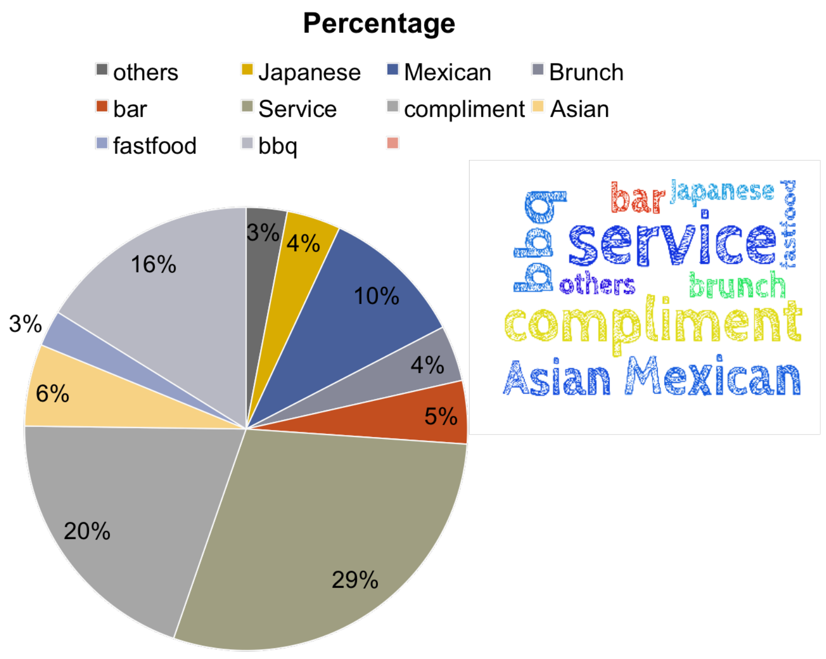
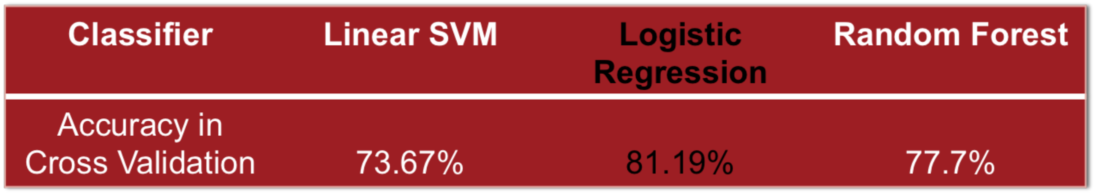


Figure 10: Percentage distribution of topics.

The next step is to infer topic distributions for each restaurants. To do this I collected all the reviews for each restaurants, and then apply the trained LDA model from previous step to the reviews. This would give me the distribution of topics in the reviews for each restaurant, which is exactly what I need in my recommendation system. For example, if a user choose category Japanese, then the system would search among all the 1966 restaurants, and return the ones where the percentage of topic "Japanese" is the highest among all the other topics in the reviews of each specific restaurant. Meaning that, words which belong to topic "Japanese" constitute most part of the reviews for these selected restaurants. Therefore these restaurants would be recommended to the user. In cases when user chooses multiple topics, I came up with a way to give a different weight to each topic. First of all I classified restaurants into "good" and "bad" ones based on their Yelp rating. If the averaging star is higher than 3.5, then it would be classified as a "good" restaurants. While if it is lower than 3.5, it will be classified as a "bad" restaurants. Then I trained a classification model to classify the 1966 restaurants. The features used in the model is a 12 dimension vector which represents the distribution of the 12 topics in the reviews of each restaurant. As for the algorithms used in the classification model, I tried linear SVM, logistic regression as well as random forest. After tuning the parameters in different cases, it turned out that results from logistic regression yield the highest classification accuracy in the validation set. The following table shows the cross validation accuracy for each model.



Classifier	Linear SVM	Logistic Regression	Random Forest
Accuracy in Cross Validation	73.67%	81.19%	77.7%

Table 5: Classification accuracy for linear SVM, logistic regression, random forest.

Among different topics, I also want to understand the relative importance of each topic in determining the quality of each restaurants. Figure 11 below shows the relative importance for the 12 topics. As we can find from the figure, compliments have the strongest relative importance contributing to a "good" restaurant. Intuitively, compliment words in reviews have very positive impact on the ratings of restaurants. It is followed by topics such as bar/atmosphere, BBQ and Japanese. This is indicating that for people in Arizona, when they talk more about bars and atmospheres, or BBQ and Japanese food in their reviews, it is more likely that they would give a high rating to the restaurant they commented on. While complains about services yield the largest negative impact on ratings because those words in reviews may lead people to give very low rating for the restaurants. Therefore, in cases when user chooses multiple topics, the weight I put on each topic is calculated from their relative importance to ensure that restaurants recommended are with the highest quality.

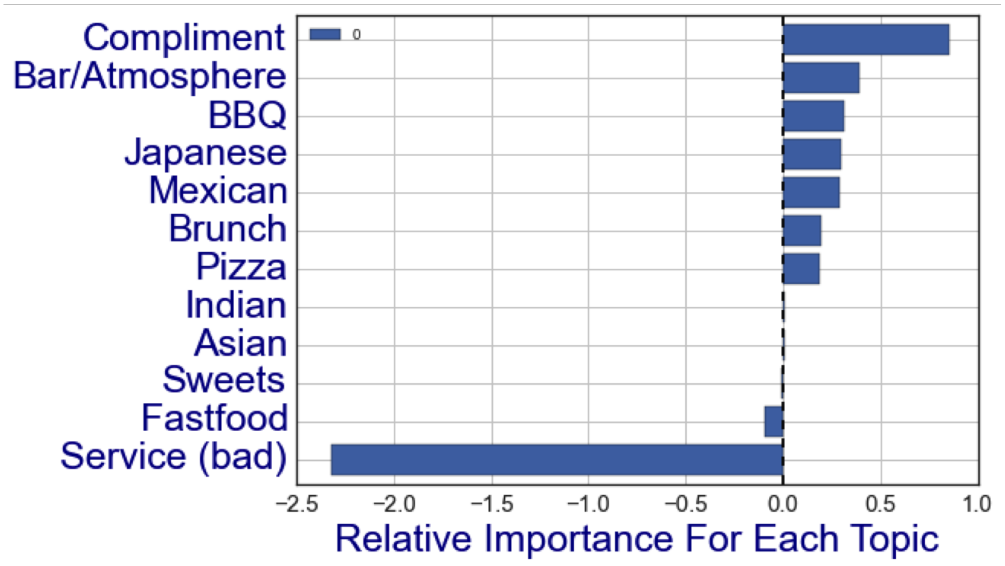


Figure 11: Relative importance of each topic in determining the quality of restaurants.

3.4 Extracting image features using deep CNN and ranking images with Ranking SVM

For each of the recommended restaurant, I would also like to select the best food photos of that restaurant from Yelp. This is a very hard task because the quality of food photos on Yelp vary a lot and it is also difficult to determine what would be the best criteria for a "good" image. This is exactly where machine learning techniques such as convolutional neural network and ranking SVM come into play in the recommendation system.

As described in chapter 2, the food images are originally lying in a 224×224 high dimensional spaces. In order to reduce dimension as well as to learn the underlying structures of these images, I applied the very deep (16 layers) convolutional networks model (pre-trained using the ImageNet data) to extract image features. The following figure 12 helps to visualize the logic we describe above and in previous sessions.

The next question is how to determine whether an image is good or not for recommenda-

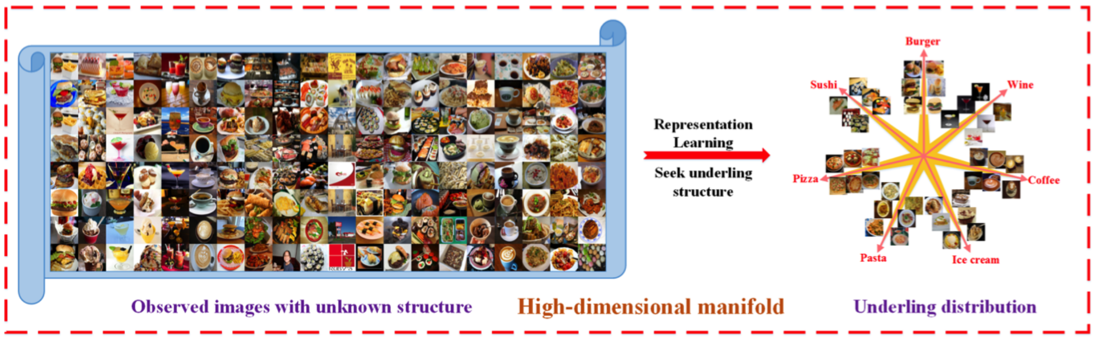


Figure 12: Application of deep hierarchical abstraction to learn the structure of images.

tion. One of the criteria that could directly reflect an image's popularity among users is by looking at the number of "likes" or the number of "votes" for each image. However, the problem here is that users on Yelp are not actively voting for or liking the food photos. Therefore most of the photo has zero or only a few number of "likes", which is not suitable to be used as label to train the model and could also lead to biased model results. In order to solve for this problem, I decided to use the food images from Flickr (`https://www.flickr.com/`) instead. On Flickr, users are very active in "liking" photos, and therefore could provide us with a very good data source to train the Ranking SVM model. I collected around 28,000 images including sushi, coffee, burger, barbecue, ice cream and so forth from Flickr and use the number of likes of each photo on Flickr as the label to rank them when I trained the Ranking SVM model. After I obtained the trained model using Flickr images, I applied the trained model to food images on Yelp for each of the 1966 restaurants in my dataset. Again the features of the Yelp images are extracted using the 16 layer very deep CNN model. As a result, for each Yelp restaurant, the image with the highest rank is returned and provided to the user. The following flow chart (figure 13) provides a visual description of the above steps.

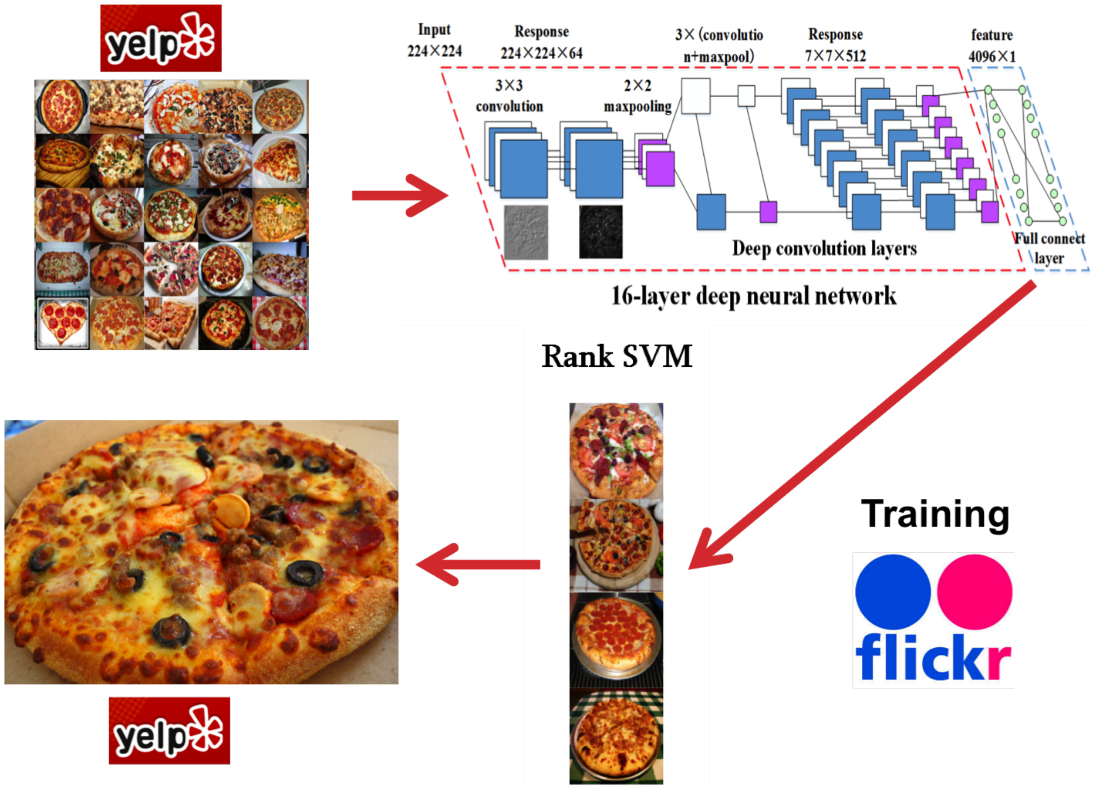


Figure 13: Ranking of best images for each restaurant on Yelp.

CHAPTER 4

Model validation

Traditionally most recommendation systems have been evaluated and ranked on their prediction power, i.e. their ability to accurately predict the user's choices. The easiest way is to perform off line experiments using existing data sets and a protocol that models user behavior to estimate recommender performance measures [[SG11]]. A more expensive option is a user study, where a small set of users is asked to perform a set of tasks using the system, typically answering questions afterwards about their experience. In this paper, since I do not have the resource to conduct user study experiments, I will use off line experiment to validate the efficiency of the recommendation system. In the paper [[SG11]], they provided an overview of a large set of properties that can be used to evaluate the success of recommendation system, and explained how candidate recommenders can be ranked with respect to these properties. The properties include root mean squared error (RMSE), precision and recall, area under the ROC curve (AUC), etc. In this paper, I used a metric which is specifically designed to measure the performance of a ranking system, the Normalized Distance-based Performance Measure (NDPM). Let r_i and r_j be the system ranking of item i and j, and $\tilde{r_i}$, $\tilde{r_j}$ be the reference ranking of item i and j (the ground truth), then NDPM score could be calculated using the following equations:

$$\begin{aligned}\text{NDPM} = \frac{1}{2 \times \sum_{ij} \operatorname{sgn}^2 (r_i - r_j)} \Bigg(\sum_{ij} \operatorname{sgn}^2 (r_i - r_j) - \\ \sum_{ij} \operatorname{sgn}\,(r_i - r_j) \operatorname{sgn}\,(\tilde{r_i} - \tilde{r_j}) - \sum_{ij} \operatorname{sgn}\,(r_i - r_j) \operatorname{sgn}\,(\tilde{r_j} - \tilde{r_i}) \Bigg)\end{aligned} \tag{4.1}$$

The NDPM measure gives a perfect score of 0 to systems that correctly predicts every preference relation asserted by the reference. The worst score of 1 is assigned to systems that contradict every reference preference relation. Not predicting a reference preference relation is penalized only half as much as contradicting it. Predicting preferences that the reference does not order (i.e. when we do not know the users true preference) is not penalized.

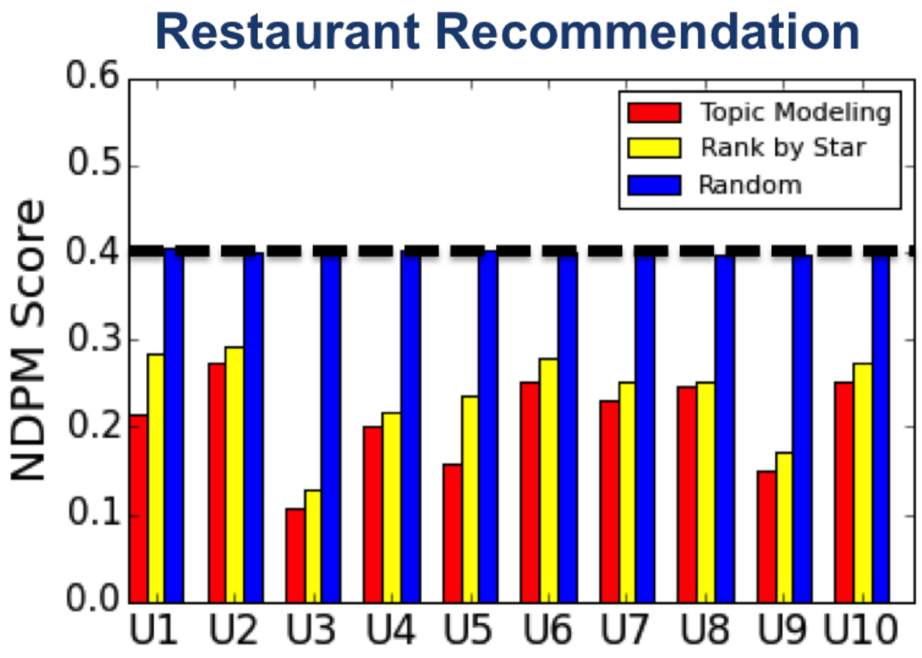


Figure 14: NDPM score for ten selected users for restaurant recommendation.

I used NDPM to evaluate the performance of both restaurant recommendation and the image recommendation. For the restaurant recommendation, I selected ten representative users from my dataset and collect the reviews from these ten users. I applied the LDA model to these reviews. This would output the most frequently discussed topics for each user. Then I assume each of the ten user would choose these topics when they use the web application. The ranking of the recommended restaurants was then compared to the ground truth, which is the actual ranking of the restaurants by each of the ten users according to the ratings they gave to them. Eventually I could calculate the NDPM score for each user and the results are shown in figure 14. The goal here is really to minimize the NDPM score. In figure 14, red columns are scores for the recommendation system built with topic modeling techniques, yellow columns are scores from a system that purely recommends restaurants according to their average rating, while blue columns are scores for random recommendation. As we can see from the figure, NDPM scores are much lower for the topic modeling recommendation system, compared to both ranking by ratings and random recommendation, meaning that the application of topic modeling to building recommendation system could efficiently improve the prediction accuracy of user choices.

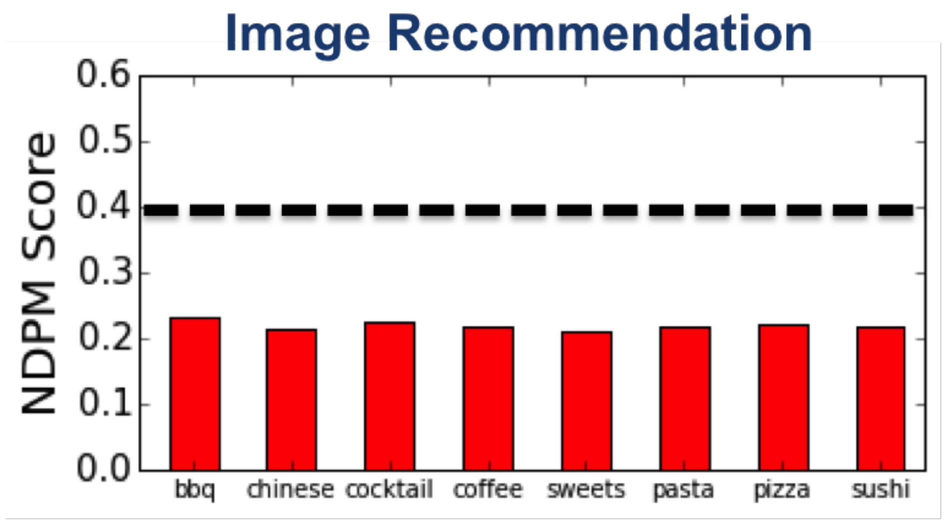


Figure 15: NDPM score for ten selected users for restaurant recommendation.

Similarly, I also calculated the NDPM score for the image recommendation. After training the ranking SVM model on the training set, I calculated the NDPM score on the validation set. The actual ranking (ground truth) is the ranking of the images by the number of likes Flicker. Since I trained a different SVM model for each food categories, the above figure 15 shows the NDPM score for each category. Again results are compared with random recommendation. As shown in figure 15, the performance of image recommendation with the deep learning and ranking algorithm has superiorly exceeded that of random recommendation.

CHAPTER 5

Conclusion

This paper explores the application of different machine learning techniques to recommendation system. LDA, CNN and ranking SVM have not been widely used in building recommender engine, nor have they been systematically studied to improve the efficiency of such recommender. This paper made several attempts to be the first to apply these techniques to the construction of a restaurant recommendation. All these techniques allow us to make full use of the information in the review texts and food images for each restaurant and hence efficiently improve the recommendation results. Further studies on how to apply these techniques to dataset of a larger scale should be necessary.

References

[And13] Chris Anderson. *The long tail.* Wereldbibliotheek, 2013.

[Ble12] David M Blei. “Probabilistic topic models.” *Communications of the ACM*, **55**(4):77–84, 2012.

[ERK11] Michael D Ekstrand, John T Riedl, and Joseph A Konstan. “Collaborative filtering recommender systems.” *Foundations and Trends in Human-Computer Interaction*, **4**(2):81–173, 2011.

[HKB99] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. “An algorithmic framework for performing collaborative filtering.” In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 230–237. ACM, 1999.

[HKR02] Jon Herlocker, Joseph A Konstan, and John Riedl. “An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms.” *Information retrieval*, **5**(4):287–310, 2002.

[HSR95] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. “Recommending and evaluating choices in a virtual community of use.” In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 194–201. ACM Press/Addison-Wesley Publishing Co., 1995.

[Joa96] Thorsten Joachims. “A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.” Technical report, DTIC Document, 1996.

[Joa02] Thorsten Joachims. “Optimizing search engines using clickthrough data.” In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 133–142. ACM, 2002.

[Kri01] Mark L Krieg. “A tutorial on Bayesian belief networks.” 2001.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks.” In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[LBD89] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. “Backpropagation applied to handwritten zip code recognition.” *Neural computation*, **1**(4):541–551, 1989.

[MS10] Prem Melville and Vikas Sindhwani. “Recommender Systems.” In *Encyclopedia of Machine Learning*, pp. 829–838. 2010.

[RIS94] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. "GroupLens: an open architecture for collaborative filtering of netnews." In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pp. 175–186. ACM, 1994.

[SEZ13] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. "Overfeat: Integrated recognition, localization and detection using convolutional networks." *arXiv preprint arXiv:1312.6229*, 2013.

[SG11] Guy Shani and Asela Gunawardana. "Evaluating recommendation systems." In *Recommender systems handbook*, pp. 257–297. Springer, 2011.

[SZ14] Karen Simonyan and Andrew Zisserman. "Two-stream convolutional networks for action recognition in videos." In *Advances in Neural Information Processing Systems*, pp. 568–576, 2014.

[ZF14] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks." In *Computer Vision–ECCV 2014*, pp. 818–833. Springer, 2014.