Wisdom Restaurant Recommendation System: An Application of Pre-Categorized Supervised Natural Language Processing and Ontology Machine Learning

S. Gazzaz; P. Shekhar; C. Wang; D. Wang[i]

Abstract

A recommendation system named *Wisdom* that recommends appropriate restaurants according to the user input is completely designed, with the application of natural language processing methods—e.g. TF-IDF, LDA, and ontology learning. A novel and interesting approach of *pentagonal analysis* that describes the quality of a restaurant from five different yet correlated facets ("food", "ambiance", "service", "cost performance", and "popularity") is developed. An efficient algorithm is designated to implement the entire recommendation system, which requires an $O(n)$ time complexity and $O(k)$ space complexity, which are proved to be the theoretically fastest designation, where n and k are the total number of restaurants in the dataset and the total number of restaurants recommended, respectively. The data training and recommendation results are studied as well in order to indicate a hard evidence of the accuracy and beauty of the system.

Key Words

Recommendation System; Restaurant; Natural Language Processing; Machine Learning

Introduction

Natural language processing can be done using Apache Spark services, which can be applied to many different kinds of web services, depending on different requests of users. Traditional recommendation systems do not contain a natural language data training part[1], in which the data training models are implemented as part of "feedbacks" from the previous users, or the "appearance" of user requests that targeted to specific entities. However, with the giant explosion of available data nowadays[2], people require recommendation system to be "smarter", not only does it smartly selects data that has an unneglectable probability to be what the user's preference, but also embraces a self-changing method to precisely calculate the degree of how the entity satisfies the user's request. Then, the challenge of *how to know the user's "taste" with incomplete searching information given by the user* arises, since research on complete information searching that points to only one entity amongst the "sea" of similar entities is meaningless. This question can be divided on two factors, the user's personal preference based on the user's behavior history, and the common preference or concepts to describe the entity based on the entity itself. Therefore, far from being some kinds of theoretical postulates, the smartness of a recommendation system is actually a realistic issue, which involves human natures and pre-classified concepts/regulations to the recommendation entities.

It is never exaggerated to claim the significance and popularity of the topic of eating. Moreover, "searching for a restaurant before going there" has become part of the routines of people nowadays. Therefore, a smart, user customized design of a restaurant recommendation system would be very beneficial in marketing, and it also contains many interesting topics worth to put research efforts on. First, there are millions of restaurants in the United States, and new restaurants open with old restaurants close occurs every single day. The data of restaurants' information is thus not only huge, but also changing frequently. Research on real-time big data that changes from time to time is one of the hottest research area in recent years[3][4]. Second, specifically in eating, everyone has a unique taste, which is why Asian food restaurant has to vary the presentation and the taste of its dishes to survive in America, "flattering" American's taste. How to imitate the user's preference in opting appropriate restaurants will be another challengeable area of research. The last but not least factor for us to select restaurant as our recommendation entity is because that, though multiple existing restaurant recommendation system have already become popular and done good job[5], none of them take the advantage of natural language processing to deal with their database of restaurants' information. Since mentioned before that the recommendation is essentially a realistic experiencing issue, it is inacceptable to put the actual dining experience of those real "experiencer", the customers who went to restaurants, away. Admittedly, most customers did not provide their text review of the restaurants they had gone to, but those customers who did give reviews tended to share their real thoughts of the dining places he/she went to in social media, an almost no-consequence place of speaking. Therefore, we believe those customer reviews are the "gold" to be the reference of whether or not the restaurant can be recommended to a future interested customer. Nevertheless, the "gold" is not already there, which means we need to delve and mine the gold from the rest redundant information. This will be the main effort of us in implementing our real restaurant recommendation system. To sum up, our main purpose of this project is to delve out a nice approach to assign a good "recommendation scoring" to each restaurant based principally on its customer reviews, which includes all kinds of text mining and information extracting methods.

Related Work

Obviously, it is not wise to collect data of restaurants' information and start processing every time when a user performs a searching with given information, because the system running will be very slow, and it is not necessary to process the same data (same restaurant) repeatedly.

Therefore, a wise designation of the recommendation system is to divide the whole system into two parts—the server part and the client part. The core of the server part is to manage the collected data. When the server is about to be turned on, it begins processing data before the server is completed turned on. Thus if we successfully turn on the server, it indicates that the data is finished processing and training which is ready to use. The client part is mainly designed for the user's input handling, and it is also the location where the real recommendation system resides. In short, the server part is *data science*, and the client part is *software engineering*. Several relate works can be found out based on the big data management and software designation.

1. Yelp Recommendation System[5]

The dataset that Yelp Recommendation System is using mainly comes from the Kaggle competition[6]. In order to properly train the dataset, Yelp uses the three different measurements: the mean value of the training set, a random sample from the training set, and a regression sample[5]. The evaluation metrics is represented as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{n}},$$

where n is the total number of review ratings to predict, p_i is the predict rating of review i, and a_i is the actually rating of review i. It divides the dataset into two parts: the *training set* and the *cross-validation set*. If the user gives an average 4.5-star rating to two restaurants, then if the 5-star appears in the training set (of course the user gives a 4-star and a 5-star for a 4.5-star average), then we come cross a review by the user into the validation set, we can know that it must be a 4-star review.

However, this recommendation system does not contain any kinds of natural language processing, which means that the user can give the overall rating of the dining place, but the machine cannot read or understand the text review accompanying the star rating. As we all know, Yelp is a very popular restaurant recommendation system, which has a very precise and advanced data training model embedded. Nevertheless, it does not do anything to the customer review texts. The possible reasons are: 1) natural language process is slow, it contains several steps in the ML pipeline, e.g. Spark NLP, TF-IDF, Feature Vector, Ontology Learning, etc., where each model may train the data many time in order to reach a certain degree of precision. 2) It is cheap to monitor the data change in star rating, but very expensive to monitor the data change in natural language. Since every time a new star rating is added/modified, the system will pass a single number to re-determine the weighted scoring of the restaurant to the recommendation list, but if a review text is added which should be interpreted immediately, the system has to take a long time to decide how the added text will positively/adversely affect the overall rating as well as the weighted scoring of the restaurant before it can make real change to its own database. This is essentially something that cannot be beautifully solved yet all over the world.

According to these issues, in our Wisdom Restaurant Recommendation System, our system attempts to avoid such kind of real time data. Instead, we have a fixed size of our database, in order to experiment on the speed of passing data in Spark NLP.

2. Learning of Imbalanced Data[7]

The biggest issue in the data processing of restaurant recommendation is that the data (text reviews) is tremendously imbalanced, which means a restaurant may have thousands number of customer text reviews while another restaurant only has three reviews. The problem is not just the difference in popularity of the two restaurants, but more like how to make the rating of the restaurants *comparable*? This problem is pervasive and causing trouble to the large size of data mining, it always occurs in a classification problem. If there are many instances of some problem than others, standard class tend to overwhelmed by the large one and ignore the small one, this is the class imbalance problem. The existing method to optimize this problem is to charting out the progress when machine learning using the imbalanced data sets by outlining the trends since the AAAI 2000 workshop, which is balancing and processing the class in different ways. This method can solve this problem in a certain context, but it hasn't been solved from the root. With the improving of the machine learning, this problem will have a better solution.

In our restaurant recommendation system, we applied a very simple way to make the different results comparable, which is *normalization*. The specific details will be introduced in the implementation section. How to make the results from trained data comparable with each other is an important but not yet solved question. Normalization is not the perfect solution, but at least it provides some reasonable results based on statistics.

3. Deals Effect on Recommendation System[8]

Restaurant is considered as one type of commercial products, which means the user has to pay to obtain food services. However, when it comes to issues related to money, money will significantly change the current rating of our recommendation entities. Therefore, a very good research of how the general recommendation ratings would be changed based on the existence of a business deal should be proposed.

Groupon is more and more popular nowadays, it became the largest deal for a lot of merchants. But there is a Groupon effect occur with this trend, which is the evaluation for the merchants of the Groupon customers always lower than the others. With this phenomenon, the evaluation on the Yelp became much lower when they use the Groupon.

An interesting research have been done for this sharp decline. There are some supposes of this phenomenon. The first one is the merchants always actively positive reviews for their business, but the customers who use the Groupon will be influenced more slightly than the others. The second one is there are some false comments for the reviews of the merchants without Groupon which means the merchants maybe using some people or auto

comments to enhance their evaluation. The third one is the merchants will give a worst service to the Groupon customers than others. And some other supposes also be put forward such as the Groupon customer will give an evaluation with the commodity connection slightly.

Based on the review filter of the Yelp, the spam reviews should be prevented to affect its ratings. So the research supposes all the reviews is true and effective. They summarize the effects of the Groupon for the rating of merchants which include intrinsic decline, critical reviewers, bad businesses, unprepared businesses, discriminatory businesses, experimentation and artificial reviews. Then doing the detailed research and analysis using these hypothesizes. And the result showed us that this phenomenon of sharp decline is effected by a lot of reasons include the poor business behavior, Groupon user experimentation and artificially high baseline, all of them play a role in this decline.

The Groupon rating sharp decline phenomenon exactly reflected the deficiency in the commercial operation. So as the reaction of this declined rating, the merchants should summary their behavior and improving their service for the Groupon customers, and treat a new customer as the regular one. And the customers also should be evaluating the commodity as usual based on related the actual service. With these deficiencies being removed, the rating decline based on the Groupon will be more and slighter.

4. Ontology Learning—Current Issues[9]

Ontology is a basement of the semantic Web and machine learning. The ontology can help the information sharing and knowledge management more effectively and efficiently. Ontology can provide a semantic foundation of the digital content for the machine. The ontology aims to provide the knowledge about for the developer and computer which include the domain and relation. And it can define the attribute, function, constraints, and axioms for an item, then it can help the computer to manage and learn the knowledge easily.

With the ontology method develop rapidly, the tools for develop and manage the ontology became requirement. The fundamental and needful function for those tools is knowledge elicitation, ontology retrieval, ontology editing, ontology validation, collaborative development and ontology transformation and presentation.

With those tools, ontology learning always has a framework, which include the information extraction, discovery of ontology, and ontology organization. The ontology extraction is pre-processing and recognize information from all the kinds of data, the ontology discovery is using the extracted information to discovery the concepts and relations, and the ontology organization means the seeking the concepts and relation via the clustering synonymous terms and their relation, deriving inverse relation, discovering local centers of concepts and building higher-level ontologies.

The domains in the ontology also have a lot of classes, there are some popular classification scheme for the domains and it's useful for the ontology learning such as established vs. under-developed domains, emerging vs. conventional domains, technology-heavy vs. technology-light domains, and self-contained vs. interdisciplinary domains.

With those steps, we can learn and use the ontology, then it can help us to transform information from the Webpages, papers, reports, books and articles to the knowledge we can use. And the ontology is advantageous to searching and filter the related information, and it's also useful for the machine learning.

With the ontology studding, we not only learned how to use the ontology to seek and extract the information we needed, but also know that the human and machine and understand each other, and the relations and domains are very important for an item. It can help us and the machine cognize and learn one thing clearly and accurately. And we can use ontology to build the knowledge framework and database.

5. Discussion of Algorithms

For the restaurant recommendation system, we use the Pentagonal method to process this system, which means that we always use the popularity, food, ambiance, service and cost performance to represent the different aspects of a restaurant. We extract the reviews of the restaurant and process them with TF-IDF and feature vector. The number of reviews use as the popularity, and the other aspects distinguish with each other based on the words processing, then we based on the reviews to give a grade for each aspects of the restaurant, then we use the grade to evaluate the restaurant.

For the data training algorithm, the tournament tree is a choice to implement the system. tournament tree is a form of min/max heap, every external node represents a value and internal node represents winner of their children. Then the root of the tournament tree will be the min/max of the tree. Figure 1 shows the tournament tree selection algorithm.

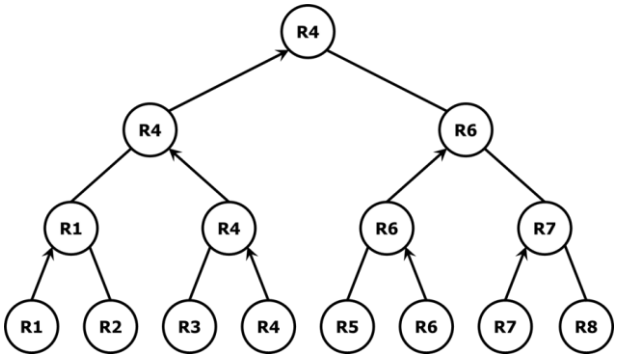

Figure 1. A *tournament tree selection algorithm* for restaurant recommendation system. R1 - R8 are restaurants stored on the leaf level. And the better restaurant will be pushed up as the parent, the best restaurant will be the root.

If we use the tournament tree to implement the recommendation system, the time complexity to select the k best restaurant list is $O(n + klogn)$, because at the first time it need $O(n)$ to build the tournament tree, and for

each time wo choose another best restaurant, it need $O(logn)$ to find it. This algorithm provides an efficient method on time complexity, but it will waste some extra space to store the data. As we know, if we have n restaurants in the dataset, it will take $2n$ spaces to store it. That means, if we need to get a huge number of recommendations, this algorithm will be not a very good choice.

Then we choose the *heap* as the data structure for the final implementation. Because it can save a lot of spaces for the system, there is no extra space in the heap. In the heap, we always using the nodes to represent the best restaurant list, and push up the worse restaurant to the top, then the root will be the worst restaurant. Then when a new restaurant coming in, we compare it with the root, if it is worse than the root, we ignore it because it is worse than all the restaurant in this Heap. But if the restaurant is better than the root, we replace the root with it, then re-Heap it. Then the Heap will be a new best list of restaurants in the dataset. This process will be done again and again. When all the dataset has been finished, the last Heap will be the best list of the restaurants. The Figure 2 showed the details in this process.

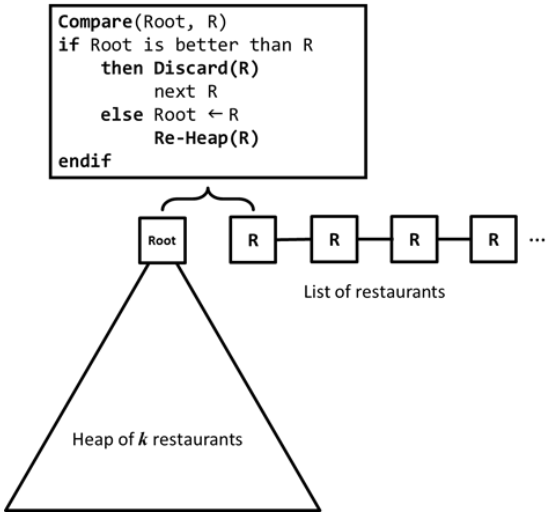

Figure 2. A heaping algorithm to select the top 8 restaurants amongst the entire list of trained restaurant data.

To sum up, the time complexity of the best case is $O(n)$, because if the 8 restaurant we choose at the first time is the best 8 restaurants, it just needs to compare all the other restaurants to the worst one in the 8 restaurants, then give up it. So it just takes n times in comparison. For the worst case, we should do the re-heap for each new restaurant come in, as we know, the time complexity of re-heap is $O(logn)$, which means the height of the tree. We always recommend 8 restaurants in this system, so the height of the tree is always 3, then the time complexity should be $n + n + 3n$. The first n means we build the initial heap, the second n means n-time comparison, and the $3n$ means n times re-heap. But it's also the $O(n)$ time complexity based on the Big-O. Accordingly, the time complexity of this algorithm is $O(n)$ for the average case. And the space the heap used is always 8 because the system always recommends 8 restaurants.

Even though for our restaurant recommendation system we always recommend 8 restaurants, but this algorithm can be used to recommend k restaurants. In general condition, if the recommended restaurants become k, the best case of this algorithm is still $O(n)$, because it still just need to do n times comparison for the final work. But for the worst case, it become $O(n + nlogk)$, because we need do a re-heap which have an $O(logk)$ time complexity for each new restaurant come in. For the average case, the time complexity is also $O(n + nlogk)$. But for the space it used it need $O(k)$ for this algorithm, so it will save a lot of space for a large dataset.

Proposed Results

The results of our Wisdom Restaurant Recommendation System is divided into two parts: the data training results and the system running results. The novel design of our recommendation system is that we never use an overall rating to represent a restaurant in the recommendation list[10]. Instead, we manually figured out five different facets of a restaurant: food, ambiance, service, cost performance, and popularity (see Figure 3). These five facets have five different scoring for each. Far from combining everything together, the five facets will describe a restaurant to the user in a pentagonal shape, where each corner represents a facet. Actually, these kinds of idea is not new, since in almost all Sport TV Channels, they use similar way to represent a player's ability, e.g. speed, height, strength, and so on. They put all these factors in to a two-dimensional shape and use each corner to represent one examining factor. The audience thus receives a very straightforward and clear picture of the style and/or class of the current player. Also, as early as 1950s, scientist begin using four different *quantum numbers* to describe an electron, main quantum number, angular momentum, magnetic quantum number, and the spin. These facts opened our mind to believe that anything is essentially determined by not only one factor, and a demonstration of a few facets that belong to one entity is even more clear and understandable to the user. Therefore, we decide to build the very first restaurant recommendation system all over the world which has five weighted, normalized recommendation ratings for one restaurant. When dealing with the user's request, all five different ratings will be all taking into consideration. Based on the human nature that the more parameters taken into account, the more accuracy we could obtain, we believe that our restaurant recommendation system will show better results or better precisions that those systems with only one overall rating considered. Also, this method is good for classifying different kinds of restaurants, which is better to do some social research on different restaurants as well.

In the pentagonal graph, the outside regular pentagon is the framework with its center displayed. The inside irregular pentagon is shaded in order to represent the actual values of the restaurant. Text labels appear in the corners of both outside and inside pentagons, which performs a very clear way to the user. The user can have a very tactile feeling of the restaurant by just looking at this novel pentagonal description of the restaurant.

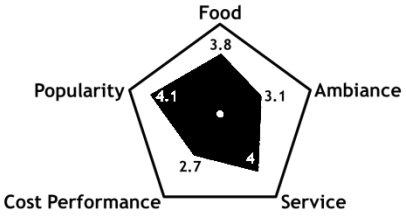

Figure 3. Pentagonal representation of a restaurant with five manually classified categories located at the five corners. The internal irregular pentagon represents the actual value of the five factors for the restaurant, where each value is represented as the distance between the vertex and the center of outside regular pentagon.

For the data training part, besides the Spark natural language processing, TF-IDF top words removing, feature vector generation, and sentiment analysis. Our main goal is to develop a graph of relations with ontology learning. The reason that relation plays as the key factor in this era is because that a review may never mention "food" or "service", but if it mentions "lobster" or "cockroaches", the system should know it is talking about "food" or "ambiance". In this case of restaurant recommendation, our ontology learning is even easier, since the only relationship we need to extract is `isRelatedTo`, we do not care about deeper relation, since it is useless in our restaurant recommendation system.

The sentiment analysis in the five fields of a restaurant is based on the text reviews. We quantified the analyzing result to have five levels: very negative (1), negative (2), neutral (3), positive (4) and very positive (5), where the number in the bracket is the `int` representation of the sentimental result. When we add the sentimental results together, we use the formula below to determine the final results of the restaurant in the training dataset,

$$F = 3 + \int_{1}^{n-1} \left(\frac{s_x}{3(n-x)} \right) dx$$

for *continuous model*, which means every piece of reviews counts as a part of validated sentimental documents. Other the other hand that will be the realistic case, the *discrete model*, the formula will look like:

$$F = 3 + \sum_{x=1}^{n-1} \left(\frac{s_x}{3(n-x)} \right)$$

where F is the value to be calculated, n is the total number of review for current restaurant, and s_x is the sentimental analysis of current review x.

To sum up, the proposed result for the training data is a JSON format of a list of trained restaurants' information, where each line in the file represents a specific restaurant and is looked like the following format, in which an empty string (`""`) means that the website of the restaurant does not exist in our database.


```
{
  "Name": "Iron Sushi",
  "Type": "Japanese, Sushi",
  "Address": "355 E 78th St",
  "City_State_Postal": "New York, NY 10021",
  "Phone": "212-772-7680",
  "Zip": "10021",
  "Website": "",
  "Food_SA": "0.550958254",
  "Food": "3.429078957",
  "Ambiance_SA": "0.100415343",
  "Ambiance": "1.533283867",
  "Service_SA": "0.240971032",
  "Service": "2.108538822",
  "Cost_SA": "0.718642639",
  "Cost": "4.211603911",
  "Wifi": "FALSE",
  "Healthy": "FALSE"
}
```


In the result, "SA" represents the sentimental analysis result, which is a value between 0 and 1. Based on these values, the star-rated results can be generated (between 1 star and 5 stars). The last two terms "Wifi" and "Healthy" are based on customer reviews. If no reviews mentioned "Wifi" or "Healthy" food, then they are considered negative. The way of implementation would have an issue of accuracy, since that no review talking about Wi-Fi does not mean that the restaurant does not have complimentary Wi-Fi service. Nevertheless, in our implementation, if the user chooses to include complimentary Wi-Fi service, then the system will pop out recommendation restaurants whose "Wifi" feature is "TRUE", which means previous customer reviews mentioned that the restaurant contains complimentary Wi-Fi service, based on the sentimental analysis and information extraction. On the other hand, if the user does not mention that he/she needs the complimentary Wi-Fi service, we assume that he/she does not care about whether the restaurant has or does not have the service. Therefore, the system ignores the "Wifi" feature from all restaurants when it is generating the recommendation list.

The second part of the result is the real recommendation results, with the pentagonal graphical representation. Since the system is prepared under a linear algorithm with smaller linear space trade-off, the running time of our real system is proposed to be less than ten seconds when using a dataset of size around 2,000.

Another key issue in our restaurant recommendation system is that the five facets at the corners of a pentagon are *correlated*, not independent, with each other. The reason is that based on our human nature, it does make sense that if a restaurant's food quality is very good, then it has a very high probability that the ambiance of the restaurant is also very good. On the other hand, if a restaurant's food is good but it has cockroaches everywhere in the dining room, then the restaurant will not be in our recommendation list, since our system does take the *standard deviation* amongst the five facets into consideration, which means if the restaurant is kind of an extreme case, then it will not be considered as a

good restaurant with high rating scores in our recommendation list.

Then, the next issue will be "how to represent an exact correlation between each two elements in the pentagon"? In other words, how the system can mathematically manage, and adjust the correlation values by itself based on the input dataset? This part is the key machine learning part in our restaurant recommendation system. First of all, we assume that the food is always normalized to unit value, since we believe the customer text reviews about the food have the highest degree of trustworthy, since restaurant is a place that serves food and all people take most care of the food quality for a restaurant. Therefore, the food review text can be treated as a "standard value" to link with all other facets in the correlation map. In our machine learning model about the correlations, the numerical values of correlative numbers are stored as below.

```
[
  {
    "Food": "1",
    "Service": "0.8-1.2",
    "Ambiance": "0.9-1.8",
    "Cost": "0.5-2"
  }
]
```

Here, the value of "Food" is always set to 1, the initial values of other features are manually input based on human nature. For example, the value of "Service" feature is "0.8-1.2", which means that if the food quality is 1, then the service quality of the same restaurant would like to reside in the region of [0.8, 1.2], and the diameter of the region is 0.4. If from the trained data, we obtain that one restaurant's food is 4-star, but the service is only 3-star, since $^3/_4$ is 0.75 which is less than the bottom limit of 0.8 for service-food correlation, than we have every reason to believe that the service quality of the restaurant is not that bad. The bad scoring probably comes from the lack of related text reviews. In this case, our system will compensate the discrepancy of

$$0.8 \times 4 - 3 = 0.2$$

by half, which means that we will report $3 + 0.1 = 3.1$ as the value of service quality of the restaurant. At the same time, the fact of the current restaurant proves that our correlative region of service quality may be too high based on the data set. Therefore, the system will lower its bottom limit of service correlation by 0.05, where the "0.05" is pre-set to be the self-study increment. On the other hand, if the restaurant's food is 3-star, but its service quality is 4.5-star based on the initial trained data, then the system will lower the service quality of the restaurant from 4.5-star to

$$\left(\frac{1}{2}\right) \times (4.5 - 3 \times 1.2) = 4.05.$$

And at the meantime, the up limited of the service quality in our machine learning model will be increased to 1.25.

However, there is a problem if our machine learning model stops being designated here, because if more and more data is loaded into the model, it is not hard to imagine that the diameter of the region will become wider and wider. Therefore, we will have the following limit if the data size is very big.

$$\lim_{n \to \infty} D = 5 - 0.2 = 4.8$$

In other words, at this moment, our machine learning model will completely lose its ability of learning. What is more, if the diameter of the region is wider, then the ability of learning will be weaker.

In order to solve this problem, a self-checking function has to be build inside the machine learning model, in order to keep the learning ability high at all times. When the diameter of a correlation range is wider than a certain number, in our model the "certain number" is pre-set to be 2, then the model will re-locate the range with a shorter diameter. In our case, when a correlative range is wider than 2, then the system finds the mid-point of the region, and place the new center of the region at the mid-point. After that, the system sets the region with diameter 1 centered at the new center point as the new region of the correlative values. This approach is simple but very effective, since we can assume that the distribution of correlations is similar to a quadratic distribution, which means it is reasonable to assume that most restaurants' correlation values are around the center of the region.

Besides the automatic diameter checking method in our machine learning model, the system lowers its bottom limit and up limit concurrently when a change is made. If the bottom limit is being lowered by 0.05, then the up limit will be lowered by 0.99×0.05 as well. Therefore, though the diameter of the region does increase after a change, the amount of changing in the diameter of a correlative region is very small (equal to $0.01 \times 0.05 = 0.0005$). In sum, the above mentioned two approaches can keep our machine learning model to be powerful in learning the input dataset.

Therefore, a constant changing in the correlative regions in our machine learning model is another result to be expected.

Implementation

The user interface of our restaurant recommendation system is developed by web page, HTML/CSS/Java Script. It includes a textbox which user can type the requirement of the restaurant he/she would like to go to, and several check boxes that the user can select whether he/she needs complimentary Wi-Fi service provided inside the restaurant, or whether he/she needs to see an entrée that is less than 600 calories. All the input and selections will become an object called `UserInput` and all values will be parsed into our core recommendation system when the user clicks "Submit" on the web page.

As we mentioned before, when the server is turned on, it means that all the data of restaurants' information is completely trained and ready to generate recommendation list. Therefore, after we obtain the user input from the web page, we can pass all the restaurants in the trained dataset

one-by-one, and each restaurant is compared with the user input to determine *how much the restaurant may satisfy the user* based on the user input. The algorithm of comparison is listed in Table 4.

Table 4. Algorithm of determination of how much a restaurant may satisfy a certain user input.

```
Algorithm Satisfy-Degree(Restaurant r, UserInput u)
      // Input: a restaurant in the list of trained data.
      // Input: a user input set grasped from the web page.
      // The algorithm returns a value between 0 and 1.
 1    return_value ← 0
 2    if u.text contains r.city or u.text contains r.postal
 3      then return 1
 4      else
 5        if u.text contains r.name or u.text contains r.type
 6          then return_value ← return_value + 0.4
 7          else
 8            if u.wifi == r.wifi == true
 9              then return_value ← return_value + 0.2
10              else
11                if u.healthy == r.healthy == true
12                  then return_value ← return_value + 0.2
13                  else
14                    if u.price == true and r.cost ≥ 4
15                      then return_value ← return_value + 0.2
16                    endif
17                endif
18            endif
19        endif
20    endif
21    return return_value
22    End
```

From the algorithm articulated above, we can see that if the user specifies any location information in the searching text box, then the system will always return the highest satisfaction degree (equal to 1) for the restaurants that matches the user's requirement. If the restaurant's type (e.g. American food, pizza, sushi bar, etc.) or name matches the user's input, then it gives an additional 0.4 in satisfactory degree. Besides, the price expectation, Wi-Fi service, and healthy food will give 0.2 additional in satisfaction degree for each.

After we obtain all the satisfaction degrees for all the restaurants, our next task is to find out a number of k restaurants that best match the taste of the user input. In our model, we set $k = 8$, but our implementation supports almost all k such that $k \leq n$, where n is the total number of restaurants in the training data.

Finding out best k restaurants amongst the sea of n restaurants is very difficult. However, we have implemented in a smart way that transform the problem into *binary comparison* problem, which means that if we only compare two restaurants at a time and are able to find out the better one, then we can use the following method to generate k best restaurants. The algorithm of this kind of *restaurant selection* is demonstrated in Table 5, in which the basic operation of a binary min-heap is omitted since it is kind of classic methods and is not belong to our designation.

Table 5. Algorithm of the selection of k restaurants amongst the "sea" of n restaurants (trained data available).

```
Algorithm Restaurant-Selection(Restaurant r[1..n], k)
      // The input r[1..n] is the list of n restaurants.
      // k is the number recommendation restaurants.
 1    Build-Min-Heap(r[1..k], 1)
 2    for i = k + 1 to n
 3      if r[i] is-better-than r[1]
 4        then r[1] ← r[i]
 5             Heapify(r[1...k], 1)
 6      endif
 7    endfor
 8    for j = k downto 1
 9      output(r[k])
10    endfor
11    End
```

From the algorithm shown in Table 5, we can see how simple and efficient the system is implemented. If we want to recommendation a number of k restaurants, we build a min-heap for the first k restaurants appeared in the list, which means the restaurant located at the root of the heap is considered to be the worst restaurant in the heap. Then we pass more restaurants in the list one-by-one. For each time, the algorithm compares the current restaurant with the root restaurant. If the current restaurant is worse than the root restaurant, then we can just discard it, since if it is worse than the root of a min-heap, it must be worse than all the restaurants in the heap. If the current restaurant is better than the root restaurant, then we need to replace the root restaurant with the current restaurant. After that, we perform the `Heapify` method to move the root restaurant to an appropriate place in the heap, which requires at most $logk$ operations. After completing the comparison of all the restaurants in the list, the restaurants in the heap will be considered as the best k restaurants in the training dataset, and the heap is the recommendation list in reverse order. We can simply output the heap in reverse order to generate the actual restaurant recommendation list.

The smartness of this implementation algorithm is that it does not require any extra space. All kinds of comparisons are *in-place* comparisons. Also, in worst case scenario, the time complexity of this algorithm is $O(nlogk)$. Admittedly, it may not be the theoretical best algorithm in time. If we use *tournament tree* as our data structure to store the restaurants' information, we might get an $O(n + klogn)$ time complexity. However, tournament tree requires at least $2n$ of space to store the data, since all data is initially stored in the leaf nodes. Until now, we have not yet found another algorithm which can perform better than ours.

Since we have already transformed the selection problem into a binary comparison problem, our next question will be: how to compare between two restaurants and get the better one? Depending on different restaurant recommendation systems, the solution varies dramatically. In our *Wisdom Restaurant Recommendation System*, we set the following rules in our comparison model.

1) The better restaurant has a larger satisfaction degree.
2) The better restaurant does not contain any < 3 values.

3) The better restaurant has a higher average values.
4) The better restaurant has a lower standard deviation.
5) The better restaurant has a higher food quality value.
6) We can randomly return a restaurant in the two.

The sequence of these six levels matters. When our model compares two restaurants, it compares the two restaurants from level 1 down to level 6. If there is no tie at any level, the "wining" restaurant will be returned immediately.

The output of our actual recommendation system is a response web page, in which pentagons with related values are demonstrated. Instead of using some kinds of complex web drawing APIs (e.g. HighCharts), our system just used the Scalable Vector Graph (SVG) tag in HTML to draw two pentagons overlapping with each other, and load numerical values in. Table 6 demonstrates an example of using `<svg>` tag to draw resulting pentagonal representation of a restaurant.

Table 6. Example of `<svg>` drawing in HTML to draw the representative pentagonal analysis of a restaurant.

```
<svg width = "380px" height = "290px">
  <!-- Outside Frame Pentagon -->
  <polygon points = "100,0 4.89,69.1
    41.22,180.9 158.78,180.9 195.11,69.1">
  </polygon>
  <!-- Internal Representative Pentagon --
>
  <polygon points =
LOADED_RESTAURANT_VALUES>
  </polygon>
</svg>
```

The GitHub repository URL of our project and the YouTube video URL of the introduction of our project is listed in Table 7.

Table 7. GitHub repository and YouTube video of the project of *Wisdom Restaurant Recommendation System*.

GitHub Repository
https://github.com/SamaaG/WisdomRecSys
YouTube Video
https://youtu.be/SoIbxlTXtVg

Results and Evaluation

In order to see the performance of our *Wisdom Restaurant Recommendation System*, several data sets with different sizes are prepared and applied. Table 8 shows the properties of each data set. For each data set, we computed the data size (the number of restaurants in the data set), and the average values of the food, ambiance, service, and cost for the restaurants in the data set. First of all, when we attempted to use data set 8, the system crashed, probably because if the JSON file is too big, it has a high probability that an error was made somewhere inside the file. Since our system implementation does not contain a JSON correction model in it, so we cannot fix the issue at this moment. The following results demonstration are all based on data set 1 to 7.

Table 8. Prepared data sets used to test the implemented restaurant recommendation system.

#	Data Size	Food	Ambiance	Service	Cost
1	388	3.78	4.12	3.23	4.12
2	650	4.60	4.56	2.99	3.22
3	700	4.44	4.12	3.57	2.88
4	1,250	4.12	3.98	3.88	2.89
5	1,499	3.99	4.12	4.00	3.56
6	1,599	4.32	4.53	3.52	2.99
7	1,909	4.56	3.23	3.28	3.97
8	199,674	3.87	3.23	3.78	3.34

The food, ambiance, service, and cost are the *average values* of the restaurants in the data set. After we collected the run time performance, the run time in seconds is plotted in Figure 9.

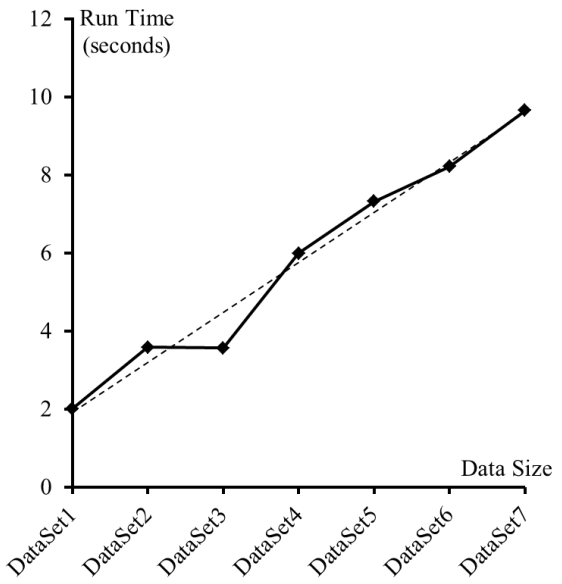

Figure 9. Run time performance of *Wisdom Restaurant Recommendation System* using data sets from 1 to 7. The dashed line is the trend line of the graph.

From Figure 9 we can see that it is obvious that the run time obeys the linear designation of the recommendation system. Although a small offset occurs at data sets 2 and 3, it is still within the acceptable range. Figure 9 proves that our actual implementation is consistent with our initial designation of the recommendation system.

Next, we would like to present the actual recommendation result list of restaurants, with an empty input, which means the user does not input anything or giving any leading information to the system. In this case, the system will generate results based on its own training method of the data set containing restaurants' information. This way of analysis can show the power of our training method. The result is demonstrated in Figure 10.

Also, after we ran the recommendation system, we checked the machine learning model to see if it really learned in our designed way, since this is model considered an important component in our system architecture. The result JSON file is demonstrated in Table 11 (after running).

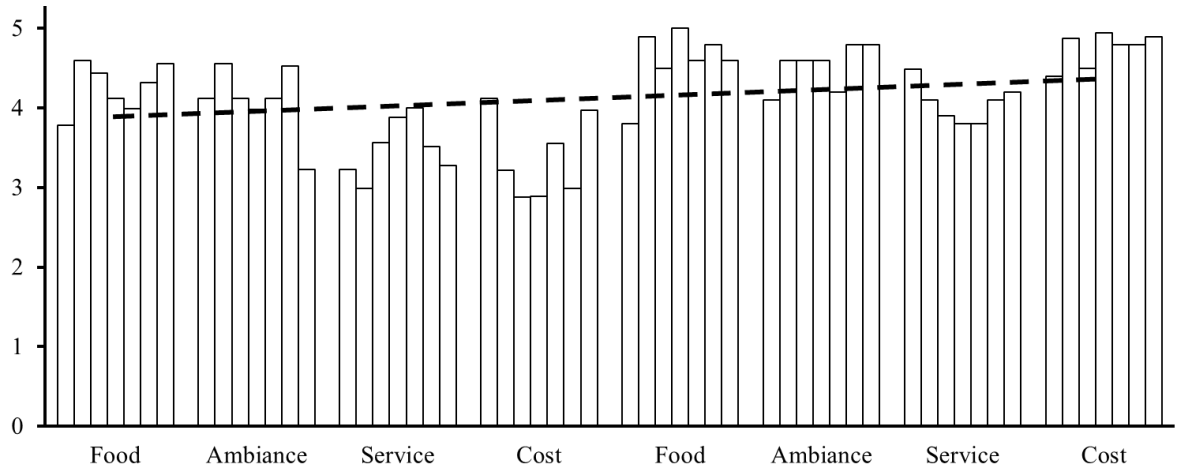

Figure 10. Results that compares the raw average values with the recommended average values for different data sets. In each entity, bars from left to right are representing data sets 1 to 7. The left four entities describe the raw average values in the trained data sets; the right four entities describe the average values in the recommendation lists. The dashed line is the trend line, which is increasing that means our recommendation system did find out restaurants that is better than average. However, the slope may be too small, which indicates that the recommendation system is not that powerful, still have room to improve the algorithm/data training method.

Table 11. Machine learning model after running the recommendation system.


```
[
  {
    "Food": "1",
    "Service": "0.5379999999971123-1.9594999999970906",
    "Ambiance": "0.31899999999710443-2.281499999996962",
    "Cost": "0.5404999999974259-1.7604999999974225"
  }
]
```


Compared with the initial values we gave, we can see the our machine learning model did learn by itself based on the incoming data sets, which is treated as another success of our implementation of the restaurant recommendation system.

To sum up, based on the power analysis, run time analysis, and accuracy analysis, it is obvious that our restaurant recommendation system have been successfully implemented with a satisfactory results in accuracy and runtime, as well as an acceptable result in the power of recommendation.

Conclusion

A novel restaurant recommendation system, the *Wisdom Restaurant Recommendation System*, was successfully developed based on the Spark natural language processing with topic supervised ontology learning. During the development cycle of the system, the novel idea of applying five different yet correlated factors to describe a specific restaurant in a pentagon. The related machine learning model was thus developed.

The results of the our implementation of the *Wisdom Restaurant Recommendation System* was proved to be accurate and consistent with the initial designation of the system. However, some weaknesses were also discovered during the test. First, the system does not include a component to rectify the defects in the input JSON files. Therefore, even though a very small recognizable error was made somewhere in the file, the system will crash when it is parsing the input data. Second, though the system successfully selected the restaurants that are better than the average level in the data set, no evidence can be acquired to prove the system really did select the best restaurant upon user's requirements. The slope of the trend line in Figure 10 is small, which might indicate that we need to think more to make the recommendation system to be more powerful.

Future Work

A product needs to beneficial in marketing. Advertisement plays a crucial role in marketing. Therefore, we are considering adding a feature of "AD Mode" to our current restaurant recommendation system. If the "AD Mode' is turned on, than a new list of *AD Restaurants* will be parse into the system. If a restaurant matches the user's request and is also in the AD list, then the restaurant will be chosen to display at the top of the final recommendation list, with a logo "AD".

The algorithm of our current restaurant recommendation system is a sequential algorithm. Although the linear algorithm is considered to be the best algorithm, the "best" is

in the field of *sequential algorithm*. Like Apache Spark or Watson, many famous web services have an advanced implementation in a parallel manner. If we can consider applying parallel algorithms to implement our recommendation system, we can plenty of room to improve the running time of the system. For example, the tournament tree method is not very efficient in sequential algorithm, but if parallel method is imported, then the running time can be decreased to $O(logn)$ for a constant number of recommended restaurants, with at least of $n/logn$ processors. Furthermore, if we can use the CRCW model (concurrent read concurrent write), then the run time of the system could be further decreased to constant time, using $n^{1+\varepsilon}$ $(\varepsilon > 0)$ processors.

The data we used in the current restaurant recommendation system is static data trained with topic supervised ontology learning. However, it would be better if the data we use for the recommendation system is real-time data. Obviously, it is too late to collect data when the use is searching something. Therefore, we need a more powerful data collecting and training server that always keeps an eye on the change of the current data set. Once a change is monitor by the server, it defines a region of "which portion of the data set should be updated". Then the server trained the related data, while the data is always available to the client side.

Also, we are still working on how find a nice way to remove the supervision of the data set. The problem is that if we pursue more concrete, the data appears less precise; if we pursue more precise, the data appears less concrete. Therefore, it is a very interesting topic that worth to put research efforts on, which will lead machine learning and knowledge management into another level.

References

[1] Chen, Ting, *et al*. "Content Recommendation System Based on Private Dynamic User Profile." *2007 International Conference on Machine Learning and Cybernetics*. Vol. 4. IEEE, 2007.

[2] Farhoomand, Ali F., and Don H. Drury. "Managerial Information Overload." *Communications of the ACM (2002)*.

[3] Marz, Nathan, and James Warren. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems.* Manning Publications Co., 2015.

[4] Fan, Wei, and Albert Bifet. "Mining Big Data: Current Status, and Forecast to the Future." *ACM sIGKDD Explorations Newsletter* 14.2 (2013): 1-5.

[5] Sawant, Sumedh, and Gina Pai. "Yelp Food Recommendation System."

[6] Recsys Challenge 2013: Yelp Business Rating Prediction. https://www.kaggle.com/c/yelp-recsys-2013. Accessed: 2014-12-12.

[7] Chawla, Nitesh V., Nathalie Japkowicz, and Aleksander Kotcz. "Editorial: Special Issue on Learning from Imbalanced Data Sets." *ACM Sigkdd Explorations Newsletter* 6.1 (2004): 1-6.

[8] Byers, John W., Michael Mitzenmacher, and Georgios Zervas. "The Groupon Effect on Yelp Ratings: A Root Cause Analysis." *Proceedings of the 13th ACM conference on electronic commerce*. ACM, 2012.

[9] Zhou, Lina. "Ontology Learning: State of the Art and Open Issues." *Information Technology and Management* 8.3 (2007): 241-252.

[10] Walter, Frank Edward, Stefano Battiston, and Frank Schweitzer. "A Model of a Trust-Based Recommendation System on a Social Network." *Autonomous Agents and Multi-Agent Systems* 16.1 (2008): 57-74.

[11] Mintz, Mike, *et al*. "Distant Supervision for Relation Extraction Without Labeled Data." *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2*. Association for Computational Linguistics, 2009.