COMP-SCI 5560 (SS16): Knowledge Discovery and Management

Second project report

RESTAURANTS RECOMMENDATION SYSTEM

Team #5 - Wisdom

Team Members

Samaa Gazzaz (9), Pooja Shekhar (38), Chen Wang (44), Dayu Wang (45)

Revision of Problem Statement

Motivation

The general idea of the motivation of our restaurant recommendation system was completely introduced in the last project report. In this report, we would like to further delve the facts that were observed which could positively or negatively affect our designation of the system. What is more, the further discussion in depth will be presented in this report to show our progress in the entire project.

The part of our ideas and motivation that are the same as we mentioned in the first project report were omitted, since it is a waste of time to repeat something we have already stated before. In this part, we focus on the changes of our ideas since the submission of the last project report, ranging from the source of data to the fine architecture of the principal processing engine.

First, the biggest issue we found that has made us have to change some of our initial thinking about the restaurant recommendation system is that it was discovered by us recently that the Yelp API can only pull out only one "snippet" review text. The screenshots in Figure 1 prove that at this moment, Yelp API has not yet opened the full access to all the restaurant reviews, which means for Yelp, there is no way to acquire all the reviewing information.

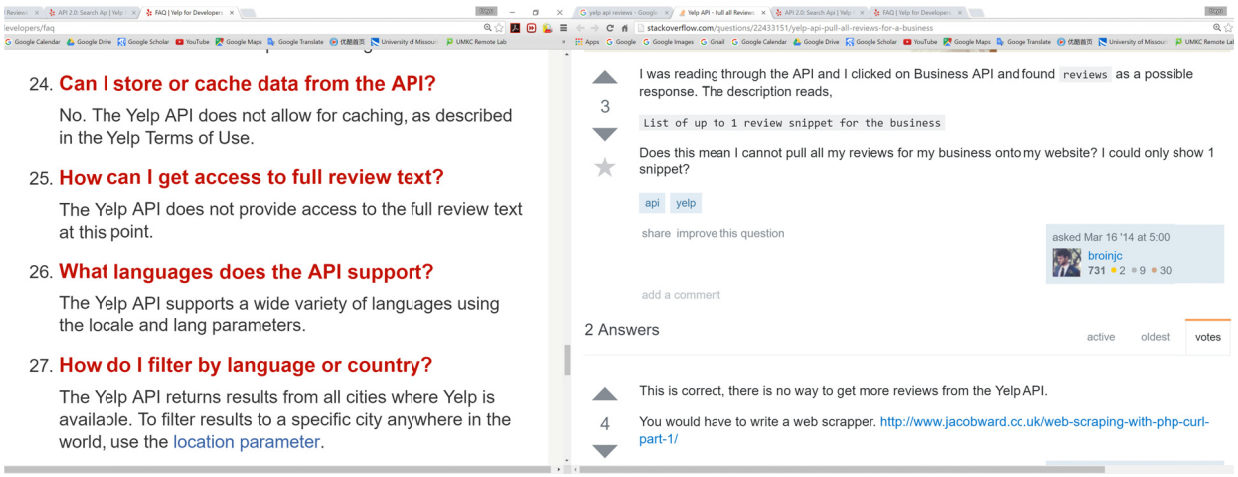

Figure 1. Screenshots that provides the impossibility to access the full restaurant review text via Yelp API. The left screenshot comes from the official Yelp documentation. The answer to question 25 says that Yelp does not provide full access to review text; the right screenshot comes from stack overflow, which again proves the same fact.

(Left) https://www.yelp.com/developers/faq
(Right) http://stackoverflow.com/questions/22433151/yelp-api-pull-all-reviews-for-a-business

Nevertheless, we have found a good substitute of restaurant API, the Foursquare API (see Figure 2), which is able to access the full review text. Therefore, we plan to change our data source from Yelp to Foursquare. At this time, the data we use to run NPL is a static set of data of restaurants generated by Foursquare. In the next increment, we plan to apply the API and use the real-time data for our recommendation system.

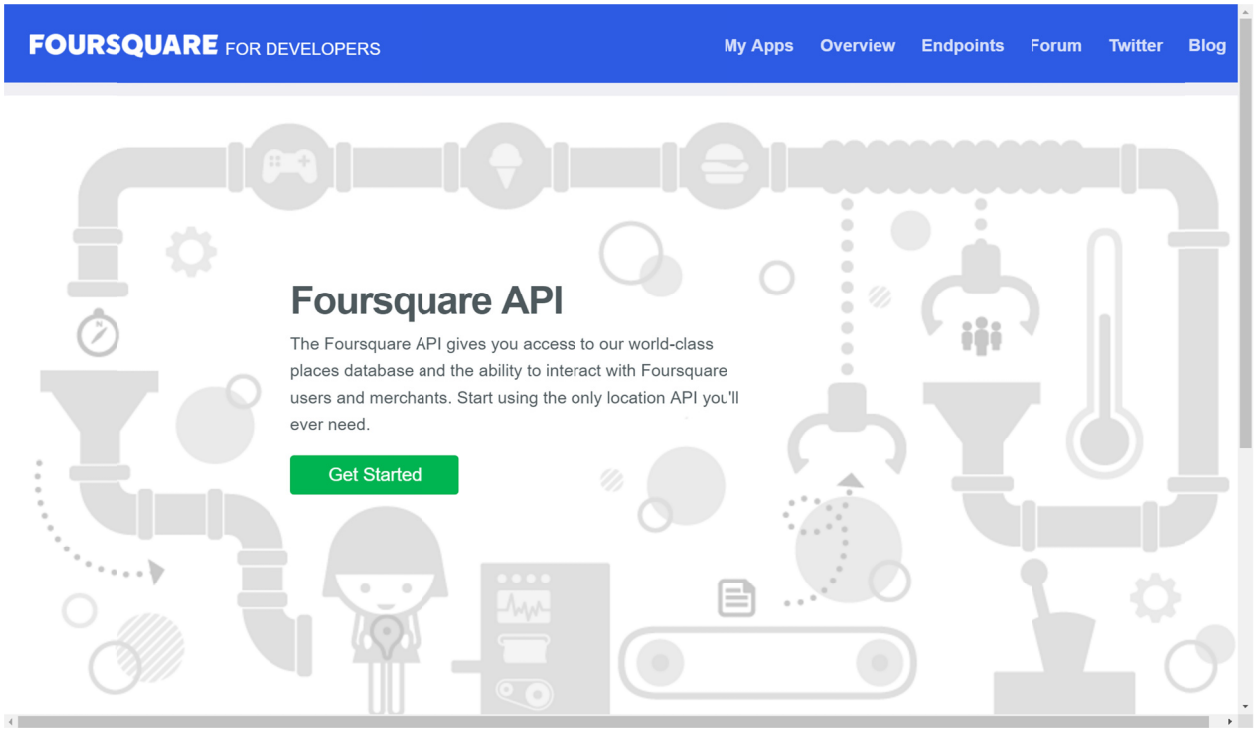

Figure 2. The Foursquare API that we plan to switch to as a restaurant data collection approach, instead of using Yelp API, which provides full access to real-time review text to restaurants/users.

(Foursquare API) https://developer.foursquare.com

Another exploration we performed is the method to treat the NLP trained data "in our way" to generate the recommendation list of restaurants. By filtering the original data collected from the API using NLP, TF-IDF, Word2Vector, and WordNet pipeline, topics, along with the sentiment analysis of it, were generated precisely. Until this time, all the data are trained by remarkable services well known by everyone, that is, not for specific application development. Our main engine, the *Pentagonal Analyzer* component in the architecture, is the core algorithm that distinguishes our system with others.

There are several questions need to be solved in this part, which is based on the following postulate.

Postulate. There is not a perfect method to recommend restaurants that satisfies everyone. Any recommendation system has its own way of evaluating how good or how bad a restaurant is. These evaluation method may be not as objective as the user may think it should be.

Therefore, the foundation of our implementation of recommendation is essentially a set of standards set up by us that evaluate a general performance of a restaurant. These "subjective" standards act as a component in the software architecture. But, the good thing is that these "subjective" standards may change itself during the training of data and it may become quite "objective" when tremendous amount of data was passed by. The self-changing standards form a core of machine learning part of our restaurant recommendation system.

Objective

In this time period, we are implementing our system. Therefore, besides the objectives mentioned in the previous project report, most of us are focusing on the coding part. Accordingly, in this time period, our objectives are divided into two parts, the correctness of implementation, and the conformance of architecture-implementation at different levels.

In software engineering, there is a long standing issue that is difficult to completely solved is the information loss acrossing different levels. For example, software architecture is a high level documentation in software design. But when we are programming the software to achieve our initial design at executable level, there must be some information loss, since some information (design smells, architectural patterns/styles, background thinking, etc.) could not be represented by code itself. This increased a danger that the code does not completely match the initial architecture. If the architecture is well designated, the program will not work correctly if the code does not match with it. Or, many bugs will appear at the executable level. Figure 3 depicts the costs to fix a bug at different stages/phase in software development.

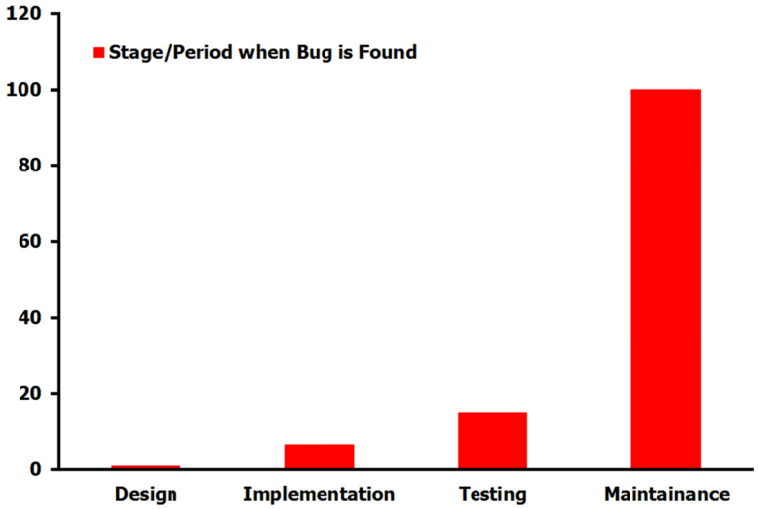

Figure 3. The costs of fixing bugs when bug is discovered in different stages/phases in software development. Suppose the cost in *Design* period is normalized to 1.

From Figure 3 we can see that, even though sometimes seemingly impossible, the earlier we found the defects of our restaurant recommendation system, the less we will spend in fixing the issue. Therefore, it is better to "check while implementing" the architecture, which means we need to update all documents across every level during the implementation. However, due to some inevitable loss of information, the "check while implementing" cannot guarantee the conformance of architecture-code. Therefore, *reverse engineering* will be done after the entire restaurant recommendation system has been completely implemented. Reverse engineering is the approach to recover/retrieve the architecture of the software from the executable code. Admittedly, it is a very costing and time-consuming process, yet we can reduce our cost in doing this since we already have an architecture with us and we assume that the executable code is not too far away from the architecture we initially

designed. Reverse engineering in our project is a way to "double check" if we correctly implemented our system, which will be completed before the final submission of the project.

Expected Outcome

At this moment, the implementation we have finished is the NPL processing part, which is the server part (refer to the architecture proposed in the first project report). We have completed the core NLP, TF-IDF, N-Gram, Word2Vec, and WordNet. Therefore, the trained data is already there. The next step, we are going to generate the largest component, the *Pentagonal Analyzer* component, to pick up top 10 restaurants from the sea of restaurants.

The corners of the pentagon is fixed by us, which makes our project to be easier to implement but remain the machine learning properties of the component. The five "keys" are: food, popularity, environment, service, and cost.

First of all, by mining the NLP trained data of restaurants, we extract the similar reviews that talk about the above mentioned keywords, followed by extracting the sentiment analysis (attitude) to the topic (positive/negative/neutral). Therefore, we first create non-correlated data that look like as shown in Figure 4.

```
Object Restaurant {
    val name:String
    val address:String
    val phone:String
    val numOfReviews:Int
    val url:String
    val &Reviews:Database
    val (food,pos,neg)=(food,pos_reviews,neg_reviews)
    val (envi,pos,neg)=(envi,pos_reviews,neg_reviews)
    val (serv,pos,neg)=(serv,pos_reviews,neg_reviews)
    val (cost,pos,neg)=(cost,pos_reviews,neg_reviews)
}
```

Figure 4. The object model for a single restaurant that does not contain any trained correlated data fields. All restaurant will be save in a list that is considered as the leaf level of *Tournament Tree*.

Then, the *Pentagonal Correlative Analysis* will participate into the data processing, which is the core factor of our machine learning process. In this part, we have to import our own interpretation of standards first, then the algorithm will work based on the imported standards. The correlative standards is based on the following important assumption.

Assumption. Amongst all the customers' reviews, the reviews talking about the food quality of the restaurant are considered to be the most important and most objective reviews compared with other reviews. The reviews about food quality are set to be the standard to consider the correlation with other corners in the pentagon.

Accordingly, the standards we input will be look like the following (Figure 5). For each correlation, we have a lower limit and an upper limit. For example, the food-environment correlation is set to be (0.9, 1.3), which means if the food score is 100, then the environment score should be between 90 to 130. If

a restaurant has a very good food quality yet flies and cockroaches everywhere (less than 0.9), then we don't recommend this restaurant but change the correlation a little bit by our algorithm. On the other hand, as the same philosophy, if a restaurant has a very good environment but ugly food, then we do not think that the environment is that good as the reviewers' remarks.

```
Object Standards {
    var food-envi(lower,upper)
    var food-serv(lower,upper)
    var food-cost(lower,upper)
}
```

Figure 5. A sample imported standards prior to the algorithm running. Note every data field is *mutable ("var" instead of "val")*, which allows further change by the training algorithm.

After the training algorithm, the data of restaurants should look like as demonstrated in Figure 6. Here, we add a boolean variable which determines if it is a good recommendation or a bad recommendation. A good recommendation restaurant should satisfies the following two requirements.

Requirement 1. The overall score of all the facets of the restaurant is equal to or higher than 80 (percentage scoring standard).

Requirement 2. If any one facet is not within the correlative range mentioned in the standards, the algorithm adds or reduces (depending on too low or too high) 10% of the current score. If after this variation, the score is still out of the range, then the restaurant will be marked "bad recommendation".

```
Object Restaurant_Trained {
    val food:Int //Normalized
    val envi:Int
    val serv:Int
    val cost:Int
    val good:Boolean
}
```

Figure 6. The *pentagonal-trained* model of restaurants. The "food" score is normalized where other data are comparative results with it.

Here, it is worth notice that the popularity (number of reviews) is an independent corner in the pentagon that does not correlate with any other corner. Because sometimes a restaurant's popularity is not because of its food or environment, but some other social events and advertisement. For example, the Amy's Baking Company in Arizona is very popular all over the states, but because in the FOX TV Show "Kitchen Nightmares", the owners of the restaurant did crazy things to Chef Gordon Ramsay to eventually make him give up.

The actual algorithm about how to determine which restaurant is the best choice will be discussed in the architecture part of this report. Our expected results are the core engine can be implemented correctly and the algorithm can work accurately at this stage. After this step, the only leftover will be the presentation part. By applying the polar graph in Highcharts API, the pentagonal representation can be demonstrated easily to the user.

Project Domain

Actually, we have many different domains involved in this project. Yet at this moment, the core issue we are facing to is the algorithm issue. To be specific, how to generate the recommendation list of indicators of restaurants efficiently?

The actual algorithm we will use is discussed in the architecture part. But research tells us it is essentially a "time *versus* space" issue. If we consider a fast algorithm, we can use a *Tournament Tree* to store our restaurants at the leaf level. Then by compare each two adjacent restaurants, we push the "winner" up to the next level, until the entire tree is filled up. The algorithm is discussed later in details, which has a $O(n + k\log n)$ time complexity. However, this algorithm requirement an additional $O(n)$ of space, though its time complexity is proved to be the best sequential algorithm.

Another way of thinking is based on the frugalness of space. We can build up a heap of size k, the number of restaurants recommended, using the first k restaurants. Then, we begin to compare the rest restaurants one-by-one, with the root of the heap. If the root is the better restaurant, then it means that all the restaurant in the heap is better than current restaurant. We can just throw away the current restaurant and go to the next one. If the current restaurant is better, then we replace the root with the current restaurant, followed by a re-heap operation to build the heap again. After we finish looking at all the stored restaurants, the restaurants in the heap will be the recommended restaurants. This algorithm does not require extra space, but it is slower, by giving the worst case time complexity of $O(n\log k)$. Figure 7 is a graphical representation of the algorithm.

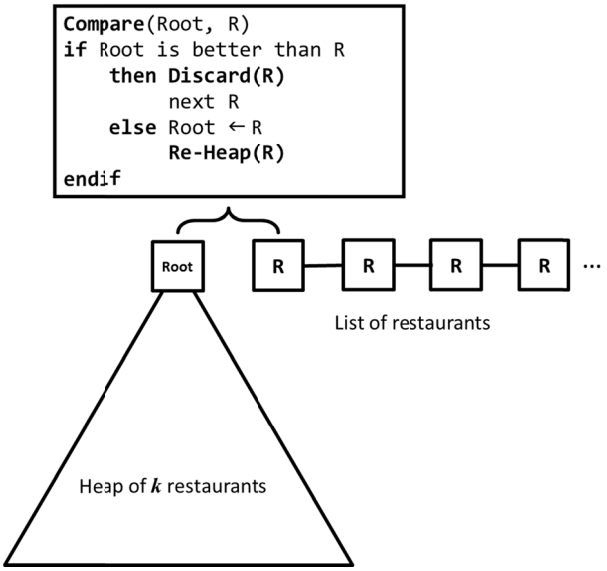

Figure 7. A heaping algorithm to select the top k restaurants amongst the entire list of trained restaruant data. The algorithm is good in space ($O(1)$ space complexity) but not good in time ($O(n\log k)$ time complexity).

Other domains of the project were proposed in the first project report, thus no repeatedness is necessary here. After the algorithm is implemented, our next domain will be the graphical representation using Highcharts.

Data Collection

Dataset Used:

To begin with, as a thumb rule Yelp Academic Reviews Dataset has been broken down into 80% training data and 20% test data. We have discovered an API CityGrid Reviews API keeps the recent reviews available for developers once we have listing_id of the restaurant from its Places API. This API aggregates its reviews from *Demandforce, RatePoint, Service Magic, Judy's Book, Insider Pages and Citysearch*. We plan to keep updating our model polling the API at a fixed interval of time. Two possible issues are associated with this are first, CityGrid is still working on to grow its review dataset, mostly has reviews from big cities and second, under contract sometimes the API just exposes partial review text (see Figure 8).

CITYGRID
Developer Center

<review>
<review_id>cg_477970532</review_id>
<review_title>Great Fish and Movie Stars</review_title>
<review_text>
Great mellow, expensive sushi bar off of Mulholland for those that don't want to drive out of Bel Air down into the city or valley.
</review_text>
<pros>Good Food, Easy Parking</pros>
<cons>Expensive</cons>
<review_rating>8</review_rating>
<review_date>2002-05-30T00:00:00.000Z</review_date>
<review_author>samowamo</review_author>
<helpful_count>0</helpful_count>
<unhelpful_count>0</unhelpful_count>
<type>user_review</type>
<source>CITYSEARCH</source>
<reference_id>0</reference_id>
<source_id>131</source_id>
<attribution_logo>
http://images.citysearch.net/assets/imgcb2/cs_logo_88x31.png
</attribution_logo>
<attribution_text>Citysearch</attribution_text>
<attribution_url>
http://www.citysearch.com/profile/106103/los_angeles_ca/sushi_ko.html#%7B%22history%22:%7B%22reviewId%22:%22477970532%22%7D%7D
</attribution_url>
<listing_id>106103</listing_id>
<business_name>Sushi KO</business_name>
<impression_id>000b000003b4ca359791e9464e98a457a5e2ed6726</impression_id>
<review_author_url>
http://my.citysearch.com/members/public/profile/samowamo?i=000b000003b4ca359791e9464e98a457a5e2ed6726
</review_author_url>
<review_url>
http://www.citysearch.com/review/106103?reviewId=477970532
</review_url>
<public_id>sushi-ko-los-angeles</public_id>
</review>

Figure 8. The CityGrid API source code.

Collection Process:

Yelp Academic dataset contains reviews of several categories. We just needed reviews for restaurants, therefore we extracted those reviews which belonged to business category ‘restaurants’. Processing the data further, we found there is an attribute ‘useful’ is the reviews which increments every time a user find a review useful and hits the useful button on yelp (see Figure 9).

Figure 9. Yelp business category—"useful" attribute.

Tasks and Features

During this iteration, we continue the ML pipeline aiming to acquire the Feature Vector by the end of all the preprocessing of the text reviews from Yelp. To be specific, we implemented Word2Vector with the help of N-Gram, TF-IDF and NLP. In addition, domain-specific (topic-specific) *Pentagonal Analysis* with topic discovery and feature vector for machine learning are also done at this moment.

In the last project report, our work mainly focused on the "words/terms" to be processed and understood via some existing services. In this period, in addition, we began to transfer our focus to the "topics", since our pentagonal analysis is based on some fixed topics related to restaurants. Specific data processing methods and results are listed below.

N-Gram and Word2Vector:

For our project, working with the restaurant and food domain, there are many common bi-grams used in the field (e.g. whole milk, Chinese food). Those bi-grams could be used by customers in their reviews and thus we have to consider them when creating our final feature vector.

Moreover, when handling text reviews, users tend to use different synonyms referring to the same word as the backgrounds of the users differ (e.g. users might use these words to demonstrate how 'nice' the restaurant was (these are actual results): cool, pleasant, neat, lovely…etc.). Thus, we have to be able to distinguish those synonyms especially when implementing the sentimental analysis of the reviews provided with our recommendations (review expected outcomes). On the other hand, processing the reviews as-is is not actually a good idea, as different forms of words could be interpreted differently, requiring pre-processing (usually using NLP).

In order to be able to understand the user reviews more, we first prepare the dataset by Lemmatizing and removing unnecessary parts (e.g. stop words, special characters). Next, using N-Gram technique, we recognize the different N-Grams with scoring. Also, using TF-IDF, we identify the most important phrases used by users in the acquired dataset. Finally, the dataset should be ready to perform Word2Vector, which is the technique that will enable us to identify the synonyms that are mostly used by users in their reviews depending on the top phrases from TF-IDF.

Unfortunately, due to lack of processing power, we were only able to process text of reviews of size 300KB. Although we provided ~8GB memory allocation, results are still of low quality and needs

improvements. Table 10 (10-1, 10-2) shows the list of top TF-IDF phrases, after N-Gram, and the top three synonym phrases using W2V.

Table 10-1. Top TF-IDF phrases via the application of N-Gram.

Top TF-IDF	Synonyms
One long	[initially quote,0.99] [winter also,0.99] [offering seasoning,0.99]
Turkey fan	[call wigle,0.99] [twice recently,0.99] [district -lrb-,0.99]
Rough-looking surroundings	[:-rrb- work,0.99] [literally everything,0.99] [leave target,0.99]
Pricy side	[suck back,0.99] [problematic pricing,0.99] [remind locker,0.99]
Guy catch	[day since,0.99] [pork worth,0.99] [mayonnaise top,0.99]
Flavor origin	[base bring,0.99] [hear good,0.99] [comparison pittsburgh,0.99]
Average taste	[, customer,0.99] [must partially,0.99] [exactly !!,0.99]

Using N-Gram is noticeable as most of the results are bi-grams. Moreover, most of these results are restaurant/food related, which indicates that the domain was picked up by TF-IDF and W2V.

On the other hand, performing W2V without the top TF-IDF values enables a more specific search for synonyms. For example, we can target the word (nice) looking for its synonyms directly from the reviews. Whereas using top TF-IDF words caused the slow of processing, excluding it enables the processing to text reviews of files as big as 300MB. The results here are more useful and of high quality. Examples are in the next table.

Table 10-2. Words used with W2V.

Nice	Horrible	Friend
cool 0.74	terrible 0.95	coworker 0.92
pleasant 0.73	poor 0.81	buddy 0.90
neat 0.71	awful 0.78	co-worker 0.88
lovely 0.69	Horrible 0.77	girlfriend 0.88
nice, 0.69	lousy 0.76	boyfriend 0.85
cheery 0.64	horrendous 0.74	finance 0.83
cute 0.63	Terrible 0.73	husband 0.83
welcoming 0.63	Poor 0.73	colleague 0.83
friendly 0.63	horrid 0.72	wife 0.83
uncluttered, 0.63	crappy 0.70	cousin 0.82
great 0.63	bad 0.70	dad 0.82
fun 0.62	shitty 0.69	bf 0.81
cheerful 0.62	Bad 0.68	brother 0.81
friendly 0.62	Awful 0.65	sister 0.80
helpful 0.62	horrific 0.63	fiancé 0.79
cozy 0.61	sub-par 0.63	hubby 0.79
delightful 0.61	TERRIBLE 0.63	roommate 0.79
professional 0.60	atrocious 0.62	daughter 0.78
helpful 0.60	horrible, 0.61	mom 0.78
Nice 0.60	lackluster 0.61	mother 0.77

Name Entity Extraction/Relation Extraction:

ConceptNet is a semantic network which contains lots of concepts that a computer should know. This is really useful when computer has to understand a human written text, which in our case is restaurant reviews. This API links two terms with a labelled relationship. We have used it to find words relevant to food used frequently in the restaurant reviews. The moment computer comes across the term 'soup' in the review text, it figures out that the sentence talks about 'food' (see Table 11).

Table 11. Related terms discovered by ConceptNet.

Restaurant	Food	Music	Ambience
At [[the restaurant]], you would [[call the waiter]]. At [[the restaurant]], [[choose a menu]]. [[a pizza parlor]] is a type of [[restaurant]]. You are likely to find [[a salad bowl]] in [[a restaurant]] You are likely to find [[Diet coke]] in [[Restaurants]]. You are likely to find [[dust]] in [[restaurant]].	[[Steak]] is a sort of [[food]] Kinds of [[food]] : [[hamburger]] A [[bread]] is a [[food]]. [[Bagels]] are a type of [[food]] [[cake]] is related to [[food]] **[[chickens]] are [[food]]** [[pizza]] is a kind of [[food]]. [[Turkey]] is [[a food]] *Something you find in [[a jar]] is [[food]] Kinds of [[food]] : [[bacon]]	**[Music]] can be [[soothing]]** [[music]] is related to [[sound]] Sometimes [[playing the violin]] causes [[music]] [[metal]] is a kind of [[music]]. [[Music]] can be [[relaxing]]	[[weather]] is related to [[ambience]] **[[mood]] is related to [[ambience]]** [[atmosphere]] is related to [[ambience]]

We have also implemented Named Entity Recognition to detect 'things' used in writing reviews of restaurant.

WordNet:

WordNet is very useful when considering the exploration of term relations (Meronymy, Hyponymy...etc.). However, since it only include the most common words in the English language, it is hard to use it with domain-specific terms. From experiments, it appears that training W2V over a large set of text reviews results in more useful synonyms than using WordNet where synonyms are limited and there is a lack of domain specific terms.

There is a huge difference between the mechanism of how WordNet and W2V works. For WordNet, there is a large corpus in the back-end consisting of a collection of words and their relationships. On the other hand, W2V uses a much smaller corpus considering the processing power of the machine. When using the reviews from Yelp as the corpus for W2V, there are not much reference when looking for synonyms. Moreover, W2V provides the similarity score because it is based more on learning from the existing dataset than just navigating through a word net, as in WordNet.

In order to exhibit the difference in results, we provide a simple example of synonyms provided by W2V and WordNet when inquiring about the same word. Looking at the results (Table 12), we can safely say that W2V results are much comprehensive than WordNet. It does not only provide us with much larger set of synonyms, it also provides the similarity score. On the other hand, WordNet only provide the words that are directly connected in the net. The number of synonyms provided by WordNet is small compared to W2V. Also, the quality of results in W2V is very strong and has broader acceptance to words as synonyms than WordNet.

Table 12. Comparison between synonyms in W2V and WordNet.

WordNet (Friend)	W2V (Friend)	
Adventist African Amerindian Apostolic Father Apostle Aquarius Archer Aries Balance Black	coworker 0.92 buddy 0.90 co-worker 0.88 girlfriend 0.88 boyfriend 0.85 finance 0.83 husband 0.83 colleague 0.83 wife 0.83 cousin 0.82	dad 0.82 bf 0.81 brother 0.81 sister 0.80 fiancé 0.79 hubby 0.79 roommate 0.79 daughter 0.78 mom 0.78 mother 0.77

In addition, WordNet is very limited when it comes to domain-specific terms. When looking for synonyms for the word (wheat), a related term to restaurant/food domain, the results are not that useful when it comes to user reviews (see Table 13).

Table 13. WordNet results for domain-specific terms "Wheat". The result reveals the limitation of WordNet that is not very useful in domain-specific terms.

Wheat
Indian corn
Indian rice
Pennisetum Americanum
Pennisetum glaucum
Secale cereale
Triticum aestivum
Triticum aestivum spelta
Triticum dicoccum
Triticum dicoccum dicoccoides
Triticum durum

In contrast, WordNet shows the most potential when used for other types of relations. This would be helpful to identify that a specific group of terms are all foods, compliments or locations. For our project, we will not be using WordNet as its benefits are replaceable by other technologies.

Topic Discovery:

Topic Model is a statistical model used by machines to discover latent semantic topic structure in a collection of document while reading a text. Given a topic, a machine can intuitively relate the document to certain terms related to the topic. We have done topic discovery of useful reviews to categorize the point of emphasis of yelpers. This would help us in building recommendation considering the important categories. For restaurant reviews we expected to find the reviews to be related to 'service', 'food quality', 'ambience', 'music' and 'location' of the restaurant. We first preprocessed the useful reviews, performed tf-idf in pipeline on the document to sustain only the useful terms of the review. We performed topic discovery using Latent Dirichlet Allocation algorithm. The output below shows the review text categorized into 4 topics. The first topic can be vaguely categorized as 'drinks', the second topic can be labelled as 'personal liking', the fourth topic can similarly be termed as 'service'.

The Latent Dirichlet allocation (LDA) can help us to do the topic discovery. We can process the comments with the LDA, then we can get the significance of different topic words. But we will do the NPL and TFIDF before, then the documents will be filtrate, the important content will be leave. We can obtain a concise and efficacious topic words for the restaurant recommendation system.

We performed a Spark LDA to assigned topics to the review text. Table 14 listed some of the results from LDA. From the table we can see which is the important topic word in the comments, which means the customer always concerned. Then we can give the customer a good recommendation based on this, and the system also can use it to do machine learning.

Table 14. Result of Spark LDA. Here listed just a sample result, where the full results could be find in our git repository.

Topic 0	Topic 1
awful,0.006135691261203186 luck,0.0059903342022183965 unique,0.005747277229653947 denny,0.0057231699594680946 longer,0.005707604355334688 lady,0.005594023842594309 welcome,0.005545655628432461 seat,0.005542027652765835	just,0.007931126190233723 favorite,0.007849852825712074 white,0.007558053323772629 dry,0.0071751501505762285 louie,0.0069937462757673294 service,0.006939546723439335 fancy,0.0068227904141238424 ni,0.00679460653328151
Topic 2	**Topic 3**
huge,0.0067525194334546 nso,0.0065101161577663024 short,0.006193676778729693 double,0.005941631365049656 selection,0.00591354224072007 style,0.005902208380232106 steak,0.0058403067434912494 high,0.005838168495882904	healthy,0.03226732599574432 mark,0.01458918955769614 round,0.014033674661588546 pour,0.008855863148914013 cool,0.007855771341182982 artichoke,0.007704403715787098 relax,0.007497437794463714 seat,0.007166783646278402 hostess,0.006633418592975372

Feature Vector for Machine Learning:

Feature vector is words which are important and specific that can help the system to do machine learning. For instance, in the restaurant recommendation system, how does the food taste and if the service supplied by the restaurant is good or not are very important, but if we lose some important words, we will make some fateful mistakes. So that how to choose the feature vector is very important.

There are a lot of design the feature vector, and the popular and useful method is use the TF-IDF weighting. Because the TF-IDF can reflect the importance and popularity of words effectively. We can calculate the TF-IDF for each term in the documents, then compare them between two documents, the distance between two vectors can reflect the similarity of those two words. So we can use the cosine value to show the correlation about their vectors. The smaller cosine value, the more similar those two words are.

Table 15 demonstrated the result of Spark NER training results (just a sample) for those topics, by using the customized data of synonym dictionaries for “food”, “environment”, “service”, and “cost”.

Table 15. The Spark NER training results for the restaurant review text (not all, just a sample) based on the four topics of “food”, “environment”, “service”, and “cost”.

Food	Environment	Service	Cost
feed	climate	duty	amount
cuisine	habitat	use	charge
snack	setting	office	damage
meat	situation	business	expenditure
drink	status	supply	figure
foodstuff	surroundings	assistance	outlay
fare	ambiance	work	payment
cooking	aura	account	price
meal	backdrop	benefit	price
bread	background	maintenance	tag
grub	circumstances	employment	rate
sustenance	conditions	utility	tariff
groceries	context	appropriateness	value
table	domain	overhaul	worth
slop	element	advantage	bite
pabulum	encompassment	dispensation	disbursement
menu	entourage	employ	dues
bite	hood	value	line
aliment	jungle	labor	nick
refreshment	locale	avail	nut
board	medium	favor	score
comestible	milieu	courtesy	setback
store	neighborhood	applicability	squeeze

Implementation Specification

Software Architecture:

Based on the architecture mentioned in project report 1, it is obvious that the principal engine of our restaurant recommendation system is the Machine Learning (Training Model) component. Since the training method is completely freshly designed by us, no existing services could be appropriate for this part, which means we will implement this by ourselves from scratch. We decided to use the *Implicit Invocation* architecture style to present this component, where the architecture is demonstrated in Figure 16.

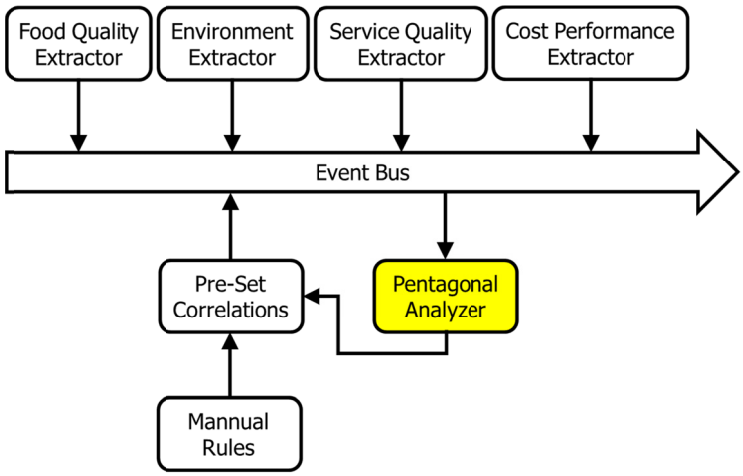

Figure 16. The architecture of implicit invocation style of the machine learning (training) model. The *Event Bus* component takes all kinds of inputs from various components. Different components may write an event to the bus at any time when the bus is on. The *Pentagonal Analyzer* component takes all the training algorithms in order to generate the pentagonal results for those restaurants.

In our restaurant recommendation system, after the analysis of feature vector, it is not difficult to grasp four different topics from the processed data, that is, food, environment, service, and cost. Also, the number of reviews can act as the popularity of the restaurant. Then, an event bus is generated in order for statistical data collection. The four components, *Food Quality Extractor*, *Environment Extractor*, *Service Quality Extractor*, and *Cost Performance Extractor*, begin to catch the information similar to itself and put the sentimental results (positive/negative) of each catch into the *Event Bus* in parallel right after the data trained by NLP approaches is generated. The *Pre-Set Correlations* is the component that changes itself with the mounting incoming data. The *Pentagonal Analyzer* takes all the regulations set by it, and afterwards, it changes itself based on the fact *pentagonal Analyzer* gives. Therefore, though the initial correlations are given by human beings, the component keeps studying the facts of restaurants by itself, and changes itself to match the realistic data collected. Besides, it is the *Pentagonal Analyzer* component who generates the data that is ready for the Highcharts component in the down streaming.

Algorithm of Pentagonal Analysis

In this section, we are talking about the algorithm inside the *Pentagonal Analyzer* component, which is implemented from scratch since it is designed for our restaurant recommendation system only. The discussion of different algorithms is based on the assumption that we have a very big data of restaurants' information to deal with, because if we only have 50 restaurants to choose, then any kind of algorithm will be meaningless.

Before we talk about the data training algorithm, it is necessary to introduce a little bit of a simple data structure of *Tournament Tree*, which we would like to use as the storage method of restaurants. Tournament tree is a complete binary heap in which all the data are stored in the leaves, which means if we have n restaurants, we need $(2n-1)$ units of space to store all the restaurants' information (n leaf nodes for restaurants and $(n-1)$ internal nodes). At the leaf level, all restaurants' information are stored and being updated. When the update finishes, it is the time to generate indicators that represent the top 10 restaurants based on the study of the algorithm. By comparing restaurant_1 and restaurant_2 (they are both leaves at this time), the winner (the one that is better restaurant) goes up to the next level. We keep doing this until the entire tree is filled, then the root will be the best restaurant selected by the algorithm. A picture representation of the Tournament Tree selection algorithm is shown in Figure 17.

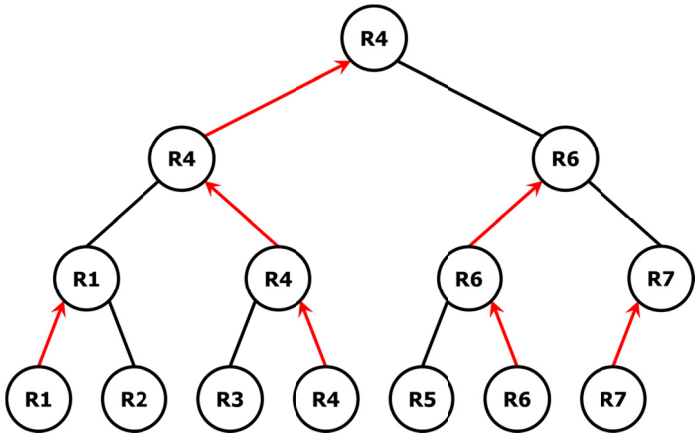

Figure 17. A *Tournament Tree* Selection Algorithm for restaurant recommendation system. R1 - R7 are restaurants stored on the leaf level. By comparing two restaurants next to each other, the "winner" (better restaurant) is pushed up to the next level (red arrows in the picture). After going through every node in the tree, the "final winner" (R4 in the picture above) will be placed at the root of the tree.

One might remain suspicious about this algorithm since if we want to choose the best restaurant, we have to go through all the nodes in the tree, which gives an $O(n)$ time complexity, the same as a simple linear search of the storage container. However, if we want to choose multiple restaurants, tournament tree makes different. Suppose we would like to pick the top two restaurants and show both to the user. We run the tournament tree to obtain the best restaurant, then it's time to pick up the second best restaurant. First, we clear the best restaurant that has been taken already. Then, seemingly we need to run the entire tree again to get the new root, but actually it is not necessary. Let's take Figure 18 as an example. R4 was removed from the tree since it is the best restaurant that has already been taken in the previous step. We can observe that essentially most of the tree does not change. There is only one

path from leaf to node that changes (red path), which is the path that contains the previously selected node. Therefore, the second running the tree does not need to update the entire tree. Instead, it only updates one path of the tree, then we can get the second best restaurant.

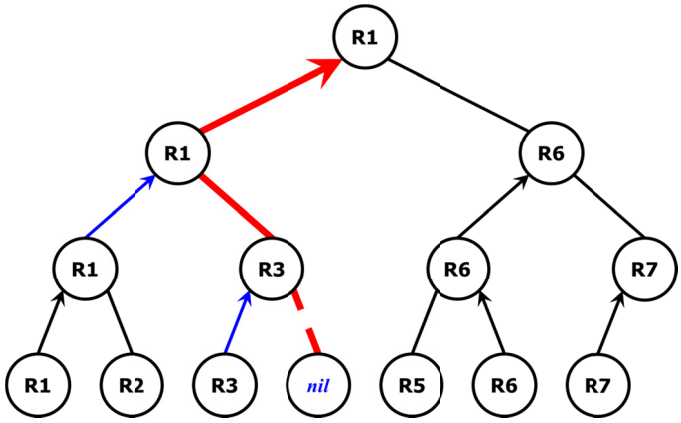

Figure 18. The tournament tree selection algorithm to choose the second best restaurant after the best restaurant was taken. We only need to update one path (red path) instead of the entire tree to obtain the second best option.

Accordingly, the time complexity to select the top one and two restaurants is $O(n + logn)$. Moreover, the time complexity to select the top k restaurants is $O(n + klogn)$, which is proved to be the fastest sequential algorithm. However, the demerit of the algorithm is that beside the storage of restaurants' data, we need an extra $O(n)$ of space to build the tournament tree. So if the data is pretty large, the algorithm requires extra space (about double space) to work.

The next question in *Pentagonal Analyzer* is: how to compare two restaurants and determine the better one? To solve this issue, we need some postulate (listed in Table 19) to fulfill the machine learning requirements.

Table 19. Postulates used to compare two restaurants (finding the better one).

Postulate 1	Every restaurant has five NLP trained data fields to describe it—food, environment, service, cost and popularity.
Postulate 2	The five data fields have different degree of significance. To be specific, food > popularity > environment > service > cost. In the machine learning part, if any conflict appears, then the algorithm uses the data field with highest degree of significance as the standard.
Postulate 3	In implementation, the overloaded operator ">" means "better than" and "<" means "worse than". For example, if R1 > R2, then R1 is better than R2 (R1 is the better restaurant).
Postulate 4	When two restaurants are competing, our algorithm attempts to avoid any "ties". The following standards is applied to compare two restaurants. 4.1. Restaurants with overall rate lower than average will never be recommended. 4.2. Restaurants with lower standard deviation amongst the five data fields are better. 4.3. Restaurants with higher food quality score are better. 4.4. Restaurants with higher popularity score are better. 4.5. Randomly return a restaurant.

UML Models:

Based on the new architecture of the *Pentagonal Analyzer* component, the UML model at this stage of the restaurant recommendation system is demonstrated in Figure 20.

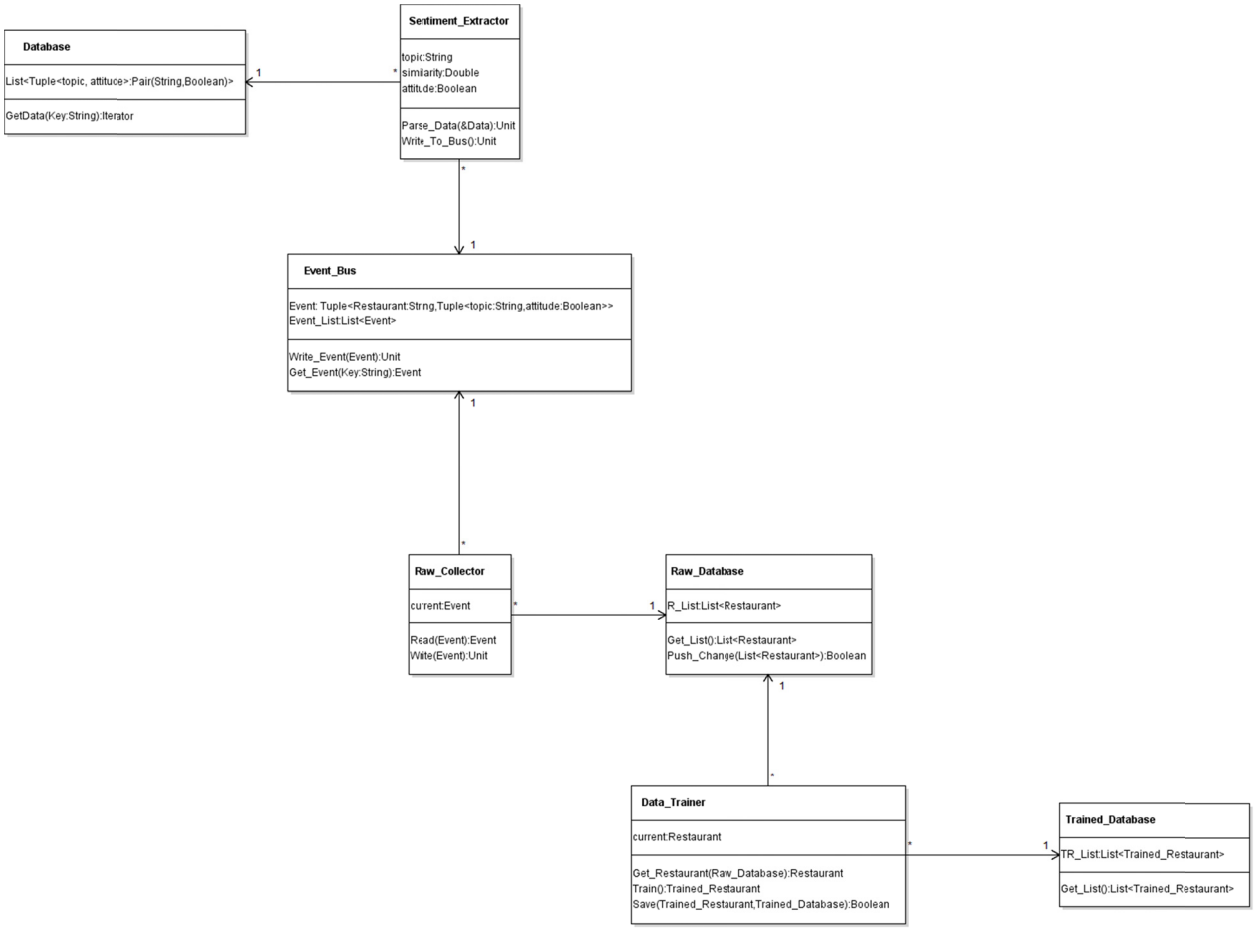

Figure 20. The UML diagram of the core engine of restaurant recommendation system. The final trained data is stored in `Trained_Database`.

Workflow Diagram (Figure 21):

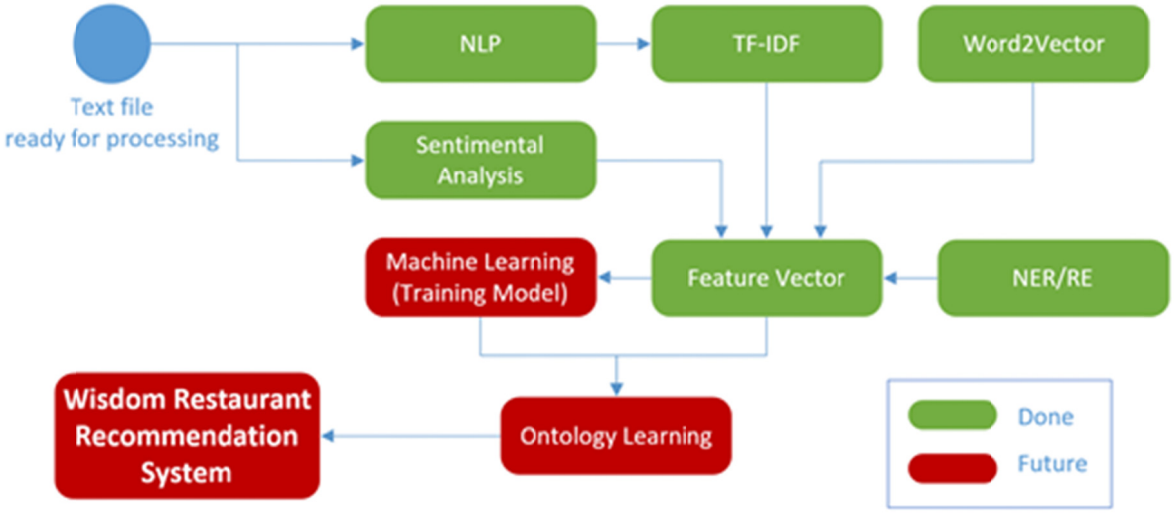

Figure 21. An updated workflow diagram of the recommendation system, with green components indicating the completed part and red components indicating the parts being implemented.

Existing Services:

Apache Mahout, Elasticsearch and Google HighCharts have been referred in the project report 1. There are some other services we've used or will use as below:

Foursquare:

Foursquare is a app for searching and discovering places and things. We've tried to implement the restaurant system with Yelp, but it just can pick up parts of the comments about the restaurant, so the Foursquare is a good choice.

Foursquare can provide a lot of advision such as food, coffee, nightlife spots, fun, shopping and so on. And for those venues, the Foursquare can give a wealth of information about them. And it also have a reliably grade and a lot of comments of restaurants which posted by users. These details about the restaurant such as name, address, food type can be used for the basic information of the restaurant for our restaurant recommendation system. And its comments can be picked up, and we can get useful data through processing them with the NLP, TF-IDF, Word2Vector, Feature Vector. And those data can be used for the machine learning and preference system. Figure 22 is a simple depiction of how Foursquare API is connected with our restaurant recommendation system.

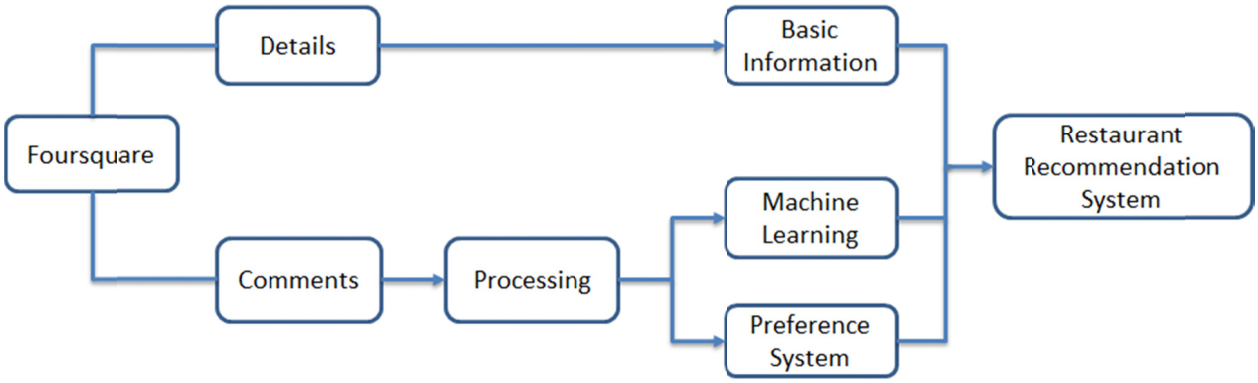

Figure 22. The use of the Foursquare in the Restaurant Recommendation system. Basic information and comments of the restaurant can be used in different way.

ArchStudio:

ArchStudio is a open source software that can analysis the architecture of a system. It's an environment of tools for modeling, analyzing, and implementing system architectures. It is based on the Eclipse open development platform.

As we know, maybe implement a system from a framework is very hard, but it's much harder to find a architecture in the ready-made system. The ArchStudio can help us to find and analyzing the architecture in the system, then we can easily to know if the system achieved the goal we expected. It's a good way to revise the architecture we've used.

Others:

The N-Gram, Word2Vector, TF-IDF, WordNet, Name Entity Extraction, Topic Discovery, Feature Vector are also be used in the Restaurant recommendation system, the introduction of those services have be referred before. You can see those details in their parts.

New Services/Features to be Implemented:

In the last project report, the preference system has been referring, and there is some new improvement will be used. As we know, without just use the average scores method for evaluation, we recommend the preference with five aspects for the user: Cost performance, Food quality, Dining environment, Service, Popularity. For the recommendation, we will also use the average score to remove 90% restaurant with low average score, but for the remained restaurant, we will evaluate them with preference.

Moreover, in fact, the 5 aspects in the restaurant is not independent with each other. For example, in the comments, a lot of people referred that the food quality is very good, then we think the environment of the restaurant won't very bad in general, and maybe the food quality will be influenced by the price, we will have different expectations with different prices. So the food quality will reflect the Cost performance a certain extent. For instance, Figure 23 showed us that the grade of a restaurant in comments and in actual. Because if a restaurant has an excellent food quality and a good dining

environment, most of people will give a good comment about the food quality in the restaurant but ignore the dining environment. So we need a data to measure the correlation in these aspects.

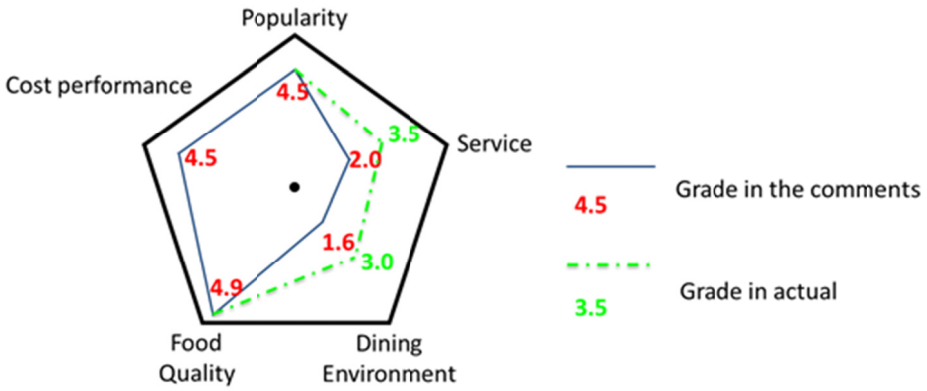

Figure 23. The grade of restaurant in comments and in actual. The blue solid lines and red figures is for the grade of comments, the green dashed lines and figures is for the grade in actual.

For the correlation in these aspects, we will give them an initial value, and revise them continually based on the reflect of the users. Through the machine learning in a long-time, the correlation in those aspects will be more and more perfection.

Project Management

Contribution of Each Member:

Member	Contribution
Samaa Gazzaz	**Tasks and Features:** • **N-Gram** • **Word2Vector** • **WordNet** **Implementation Specification:** • **Workflow Diagram**
Pooja Shekhar	**Tasks and Features:** • **Relation Extraction using ConceptNet** • **Topic Discovery using LDA** • **Named Entity Extraction** **Dataset Collection** • **Preparation of Training & Test Set** **Discovered CityGrid API for dynamic updating of model**
Chen Wang	• **Topic Discovery** • **Feature Vector for Machine Learning** • **Existing Services** • **New Services/Features to be implemented**
Dayu Wang	• **Motivation - primary contributor (there isn't one word the same as project report 1)** • **Objectives - primary contributor (there isn't one word the same as project report 1)** • **Expected Outcomes - primary contributor (completely new proposals)** • **Project Domain - primary contributor (targeted to the algorithm part)** • **Architecture (graph) of the Core Pentagonal Analyzer - primary contributor** • **Discussion of Algorithms - primary contributor** • **UML Diagram - primary contributor** • **Concerns/Issues and Future Work - secondary contributor**

Version Control/Screenshots:

For this project, we used GitHub as the main version control tool. The whole project, in addition to documentation, is up on: https://github.com/SamaaG/WisdomRecSyshttps://github.com/SamaaG/WisdomRecSys

Concerns/Issues:

1. The way to initialize a query of restaurant still needs to discuss.
2. At this moment, each component in ML pipeline takes some time to obtain the results. I am concerning the performance of the entire system once we combine all the components together.

Future Work:

1. Implementation the rest components (red components in Figure 21, the workflow diagram).

2. Consider the designation of query input and output.