A GENERAL TOOL FOR INTERACTIVE ANALYSIS OF OPINIONS IN REVIEWS

BY

JINYAO XU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Professor Chengxiang Zhai

ABSTRACT

Online review data contains useful information about consumers' opinions about the reviewed products. Understanding reviews is extremely important for a producer, such as a corporation designing and producing cell phone, to predict the market and make future strategies. However, due to the large volume of the review data, it is difficult to quickly digest the reviewers' the preferences, or to further recognize interesting patterns of reviewers' opinions. To address this need, in this thesis, we propose a novel analysis system based on Latent Aspect Rating Analysis Model. It not only understand the feedbacks or reviews from the costumers, but also analyze the market and trend behind the data.

As for the user's basic interface, we define some operators for the analysis system and the user can conduct our predefined queries with the operators over the data to analyze the products. Futhermore, we allow the users to do more complex jobs over the system by designing a pipeline syntax of the operators. Besides, the system can visualize the results, so that the users can better understand the reviews.

Overall, the proposed analysis system is general and can support analysis of any review data. It thus enables many interesting applications such as product analysis, market prediction and business intelligence.

To my parents, for their love and support.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Nowadays, review is considered as a very valuable resource for the the producers to understand their customers, especially when designing the next generation a product or modifying some existing functionalities of the current version. Also, it is extremely useful to take the customers' attitudes towards every aspect of a product into consideration, because everyone has his or her own reason to like or dislike a product. Besides, learning a product's pros and cons against its rivals could also help to draft further development strategies.

Traditionally, gathering feedbacks and reviews from the customers on the internet, reading them, and modifying the products according to the reviews are considered as a reasonable strategy. However, in the era of information explosion, the number of reviews and feedbacks could reach tens of thousand because people tend to express their opinions on the internet freely. As an example, Play Station 3, produced by Sony in 2006, has around 1200 reviews on Amazon. However, Play station 4, sold in 2013 received over 10000 reviews on Amazon[1]. On one hand, larger amount of reviews indicates a preciser prediction statistically. On the other hand, it is beyond human being's ability to digest everyone of them, not to mention deeply analyze or understand every sentence and every aspect of a review. Therefore, how to effectively analyze and digest the customer reviews in a most effective way become a very urgent problem for the producers. Some existing review websites such as *www.yelp.com* simply summarizes the reviewers' opinions by giving each product an average score based on each reviewer's overall score. Such simple numerical results can not help much. Moreover, some sentimental analysis tools can only show polarity of each sentence in a review, which cannot give the users deeper understanding of the texts.

To better solve this problem, we propose a novel analysis system built

[1]http://www.amazon.com

on top of the results of Latent Aspect Rating Analysis Model [1], which can help the producers to comprehensively understand the reviews from costumers. LARA can analyze each aspect of each review, revealing how much weights a customer put on each aspect when writing the review. Basically, it calculates a customer's ratings and weights for each aspect of the product based on the review written by him or her. However, the raw results alone is not straightforward enough for a company to analyze and understand the customers and the market. The new analysis system is invented to aggregate the LARA results, build an interface for the user to retrieve and display them in a more comprehensive way.

A major novelty of the proposed system, which differentiates it from all the existing systems, is that it provides a query based interface with a set of general analysis operators support. Similar to the SQL "SELECT FROM WHERE" pattern, users can manipulate and analyze data with our predefined query syntax. Furthermore, the operators in the query can be pipelined and combined. And we will introduce this advanced feature in Chapter 6.

The proposed system is a general review analysis platform that can work on reviews of all types of product. Thus, it enables more applications and review based analysis tools. Moreover, the system is higly extendable for developers.

CHAPTER 2

BACKGROUND

2.1 Overview

The analysis system is built based on Latent Aspect Rating Analysis model. Therefore, in this section we shall introduce the mechanism of LARA before we dive deeper into the design and implementation analysis system in the next chapter. Basically, LARA is a NLP model that takes a set of reviews of products and a user given overall score for each product input,

$$
\begin{aligned}
LARA\_Input = \{ & \\
& product1 : (\ rating1, [review1.1, review1.2, ...]\ ), \\
& product2 : (\ rating2, [review2.1, review2.2, ...]\ ), \\
& product3 : (\ rating3, [review3.1, review3.2, ...]\ ), \\
& \qquad ... \\
& productN : (\ ratingN, [reviewN.1, reviewN.2, ...]\ ) \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \} \quad (2.1)
\end{aligned}
$$

and outputs the weights of each aspect in each review as well as the word polarity of vocabulary, which can be used to further calculate the aspect ratings.

The advantage of LARA over some other models is that it assumes that each reviewer puts different weights on different aspects when writing his or her reviews, which further reveals the relative importance a reviewer attach on different aspects of a product. It assumes that the overall score given by the user is not determined by aspect rating alone, but the dot product of the aspect weight vector and aspect rating vector.

2.2 Aspect Segmentation

As for the implementation, LARA is a two-stage algorithm. In the aspect segmentation stage, it tokenizes sentences from reviews and tag each of them with a predefined aspect tag. Aspect segmentation is an iterative algorithm. In each iteration, it expands the keyword set of each aspect according to kai-square measurement ranking. Initially, it starts from the state *Initial*:

$$
\begin{aligned}
Initial = \{ & \\
& aspect1 : [keyword1.1, keyword1.2, ...], \\
& aspect2 : [keyword2.1, keyword2.2, ...], \\
& aspect3 : [keyword3.1, keyword3.2, ...], \\
& \quad ... \\
& aspectN : [keywordN.1, keywordN.2, ...] \\
& \} \quad (2.2)
\end{aligned}
$$

The aspect and its list of keyword in the *Initial* state is predefined by the user of LARA. The selection of keyword is important because it might greatly affect the final results.

The workflow of Aspect segmentation can be described as followed: [1] In each iteration, given the input text and the current list of seed word for each aspect, assign each sentence the aspect with the majority vote of all the word aspect. Next, it uses the kai-square measurement to calculate the dependencies between aspect and words. Then it expands the key word list for each aspect by including the high dependencies. The iteration ends when the number of iteration reaches threshold or the aspect keyword set does not change any more. Finally, it outputs a $k \times n$ matrix W for each review in the review set D, where k is aspect set size, and n is the vocabulary set size. The matrix W describes the number of occurence of each words in vocabulary in each review.

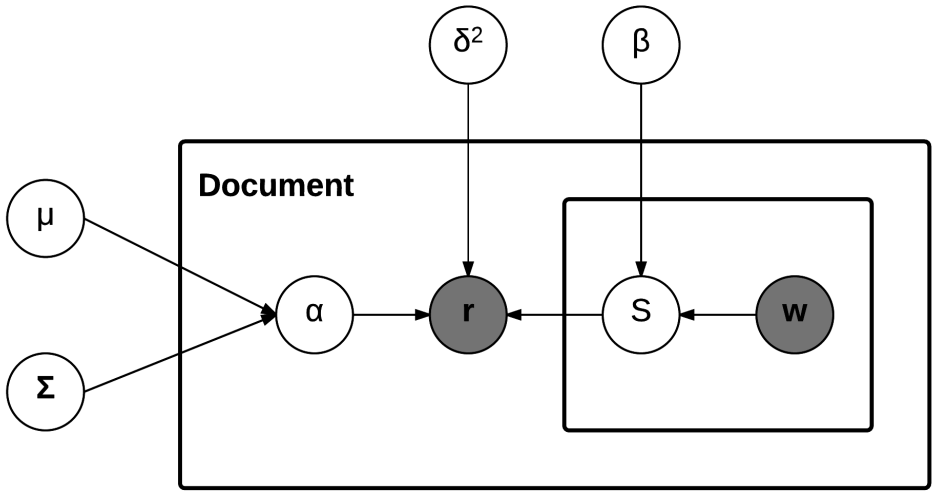

Figure 2.1: Graphical Representation of LARA

2.3 Latent Rating Regression

In the second stage, named the Latent Rating Regression, it runs an Expectation Maximum algorithm over the output data from the first stage W for each review. Rather than diving into the complicated math part of the EM algorithm, we will declare some key assuptions in statistic that support LARA [1].

- We assume that each reviewer put different weights on different aspects of a product in the same review. And the aspect weight vector of each reviewer α is retrieved from the following normal distribution $N \sim (\mu, \Sigma)$, which is independent from each review. The aspect weight is regarded as a latent variable during the learning process.

- each word $V[i]$ in the vocabulary set V has a sentimental score β_i indicating the polarity, the score is independent of each review

- the score of a review given by the user is retrieved from the distribution: $N \sim (\sum_{i=1}^{k} \alpha_{di} \sum_{j=1}^{n} \beta_{ij} W_{dij}, \delta^2)$

Then we write the Maximum Likelihood Estimate for each review as:

$$P(r_d|d) = \int p(\alpha_d|\mu, \Sigma) p(r_d| \sum_{i=1}^{k} \alpha_{di} \sum_{j=1}^{n} \beta_{dij} W_{dij}, \delta^2) d\alpha_d$$

and the aspect weight α is the latent variable from $N(\mu, \Sigma)$. Term sentiment matrix β is iteratively calculated in the EM algorithm with the aspect weight vector α as the latent variable. To further clarify the relationship between the above random variables, we use plate notation in Figure 2.1 to explain it.

2.4 Post Process

As the Figure 2.1 described, LARA outputs the latent variable α for each review as the aspect weight vector , and a tuple of four $(\beta, \delta^2, \Sigma, \mu)$ of the corpus level variable. However, what we need from result of LARA for our analysis system is a set of aspect weight vectors α and the term sentiment set β .

To make the whole process clear, we show one of review [1] from the input set of LARA in Figure 2.2. This is the most common form of review on the internet, a text of around 200 word with an overall rating given by the reviewer as a numerical summarization for the product. After the two-stage LARA algorithm decribed in the previous sections, it will generate a result of:

$$\begin{aligned} LARAResults = \{ & \\ & \beta : [nice : 2.3,\ \ fair : 0.4,\ \ bad : -2.0\ \ ...\ ] \\ & \alpha : [\ function : 0.5,\ \ price : 0.4,\ \ look : 0.1\ ] \\ & \} \end{aligned} \quad (2.3)$$

We can calculate the overall rating of the kth aspect for the product in an review with the formula:

$$A_k = \sum_{i=1}^{n} \beta_{ki} W_{dki}$$

Furthermore, we get the aggretate aspect rating of the product by averaging

[1] www.amazon.com

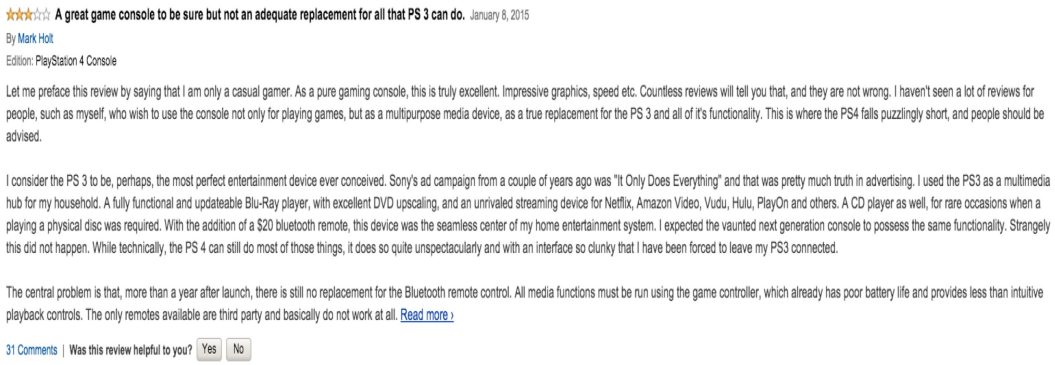

Figure 2.2: Play Station 4 Review From Amazon

the kth aspect score of each reviews commenting on the product.

$$AProductk = \frac{1}{m}\sum_{i=1}^{m} A_{ki}$$

In this case, we collect both score of each review and the score for each product to build our analysis system. As for the set of α for each review, we should also group them by product, take the average and get the aggregate aspect weight for each product

$$ProductAspectWeight = \frac{1}{m}\sum_{1}^{m} \alpha_m$$

CHAPTER 3

RELATED WORK

Many existing review analysis models focus on a review's overall sentimental polarity. They either classify the reviews into {Positive, Negative} [2, 3, 4], or try assign a numeric score for each sentence with some NLP knowledge [5, 6, 7]. Some of them did a great job in classifying the reviews and giving the user a brief summary of reviewers' opinions over a product. However, there are more information hiding in reviews besides polarities. For example, a reviewer commenting a cell phone can praise the powerful functionality for 80% of his review text, while complaining its ugly look for the rest of 20%. If a model only analyzes the sentimental polarity regardless of what aspect each sentence is talking about, it is likely to just tag the review as a positive or neutral one, and forget about the negative sentences in the review.

Latent Aspect Rating Analysis Model, however, is able to dive into the aspect level of a review, and get users' opinions for each aspect of a product. It can reveal the users' relative preferences to every dimension of an item. It needs a set of review text and their corresponding overall ratings as the input, and can outputs the aspect score in each review. In this case, we can understand and digest a review more efficiently. We propose a new analysis system based on the LARA model. The greatest difference between our analysis system between the existing text or review analysis system is the generality and extendability.

Some existing text analysis frameworks [8, 9] have done a good job in helping the users to understand some long or complex reivews. However, our new proposed framework is superior to them in many ways. First of all, our analysis system is a operator-based and query-based system. Users can use the query interface to conduct a set of predefined operators over the data from the output of LARA model. And the operator set can be extended by developers according to the needs of the system users. Furthermore, as a general analysis platform, the system can basically analyze reviews of any

product. Also, as a rule of the operator set, developers should keep the operators as general as possible, so that every operator can be used in any type of reviews.

CHAPTER 4

SYSTEM DESIGN

The analysis system is built on a reliable infrastructure. The implementation of the system is also a very challenging job. Considering an application built on LARA, we have to persist the LARA results and build a data service as well as an analyzer over it. Moreover, a query based system needs an efficent and generalized parser to interpret the input. The system has three parts, as the Figure 4.1 describes. Frontend receive inputs from users and visualized the results. The middle layer parses the query and converts it to a request to backend. The backend provides data services, handling the requests, retrieving relavant data from database, analyzing it, and return a json object.

The frontend is built upon some basic javascript codes and a visualization library for result display. The middle layer is the core of the system, and this is also the reason why we choose not to integrate it to the backend. Middle layer is responsible for parsing the queries from the frontend. The parser extracts the operator, determing what operation the user need, and then reform the data into a json. Lastly, it builds a request according to its protocol with the backend. There are two advantages to set up a middle layer for this functionality. Firstly, in terms of engineering, a team can maintain or extend the isolated part since there is no coupling with the middle layer and backend. Secondly, if switch the backend to other review mining algorithm, the operator set can still be used as a perfect proxy between backend data service and frontend data visualization.

The backend data service is built with Python CGI with a SQL database . We store the results of LARA in the database and the CGI script handles the queries from the frontend, retrieves relavant data items, analyzes it and sends back a json object. It can scale if the data size grows.

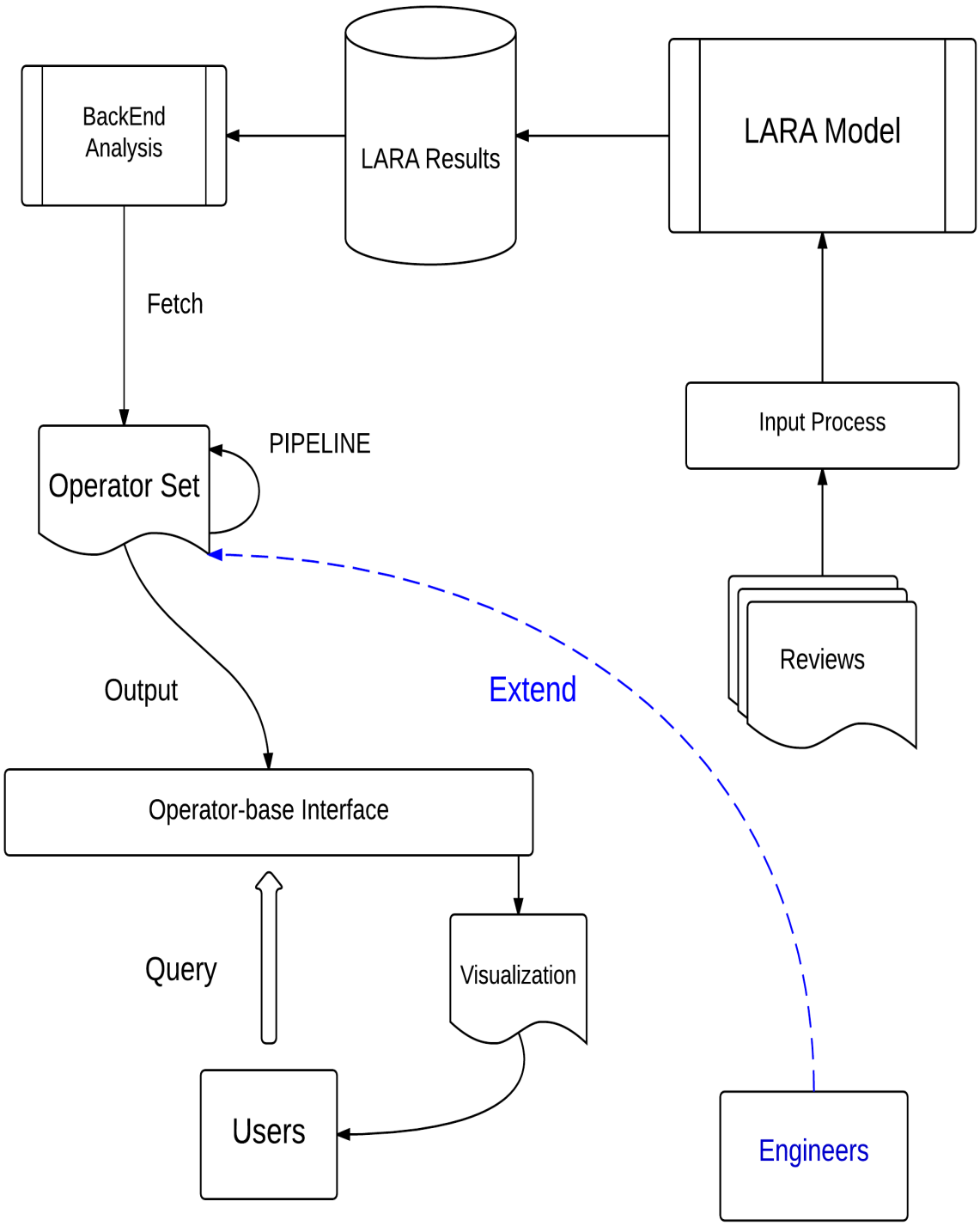

Figure 4.1: System Design

CHAPTER 5

OPERATORS

5.1 Syntax Design

The core of the system is the operators and the corresponding queries over data because the user will directly interact with this part. In this chapter, we shall firstly formally define operator and the syntax of the query in our analysis system. And then we will analyze the use and the advantages of each operator we currently have in our system.

We define the operator as a special convertor that takes in a set of LARA output(aspect weight vectors, aspect rating vectors, product information, and related meta data), re-transforming or re-combine the raw data, and output either a json object to describe the user desired info, or a rearrangement of a list of products according to the query.

As for the definition of query in our analysis system, a query is a combination of an operator, a product or a list of product as object or target, and a set of constraints over the operators and targets. And the combination shoule obey our predefined syntax for our query rule.

Overall, operators and query syntax design is one of the most challenging task in building the analysis system. On one hand, when designing an operator, we need to consdier how useful it is for the analysis system user. On the other hand, we need to design a reasonable syntax for the query so that each operator can be used effectively. The syntax cannot be over complicated for the users, nor can it be ambiguious. Also a well-defined syntax should allow developers of the system to further expand the operator set or even the syntax itself easily. Furthermore, we need to consider the generalization when pipelining two or more operators, which will be covered in the next chapter.

The syntax of the query is:

$$\begin{aligned} \textbf{OP } \{product\} \mid \{list\_of\_product\} \\ \textbf{PREP1 } \{constraints\} \\ \textbf{PREP2 } \{constraints\} \\ ... \\ \textbf{PREPn } \{constraints\} \end{aligned}$$

Furthermore, the output results of a query is one of the three items:

- an item (in our system, an item means a product)
- a list of item
- a json object describing the property of an item/items

Maybe in the implementation, we will visualize the result. However, the data that supports the visualization is one of the three types.

An **OP** corresponds to one specific type of query action over the data. And currently, the demo analysis system has OperatorSet = { COMPARE, RANK, WEIGHT}, which will be further expanded in the future work. And we will dive deep into each of our predefined operators in the later sections.

The **PREP** stands for preperation, and is usually followed by a constraint. Currently, we have PREP set preperationSet = {WITH, IN}, which is also expandable along with the growth of the operators set.

The relationship between the operators set and the preperation set is very important for the design. On one hand, for each operator **OP**, it is followed by many, one or no preperation in a valid query. And a same preperation might be used to constrain different operators. Therefore, we can conlude it is a many-to-many relationship.

In most cases, the constraint following a preperation keyword filters or limits the data fetched by the operator. Different types of preperations describes different types of constraint. The "IN" keyword, for instance, limits a range for the object selected by the operators. And omission of a preperation indicates no limits or constraints. In the following three sections, we will briefly introduce every operation in the operator set along with its corresponding preperations in our current demo analysis system. A valid query,

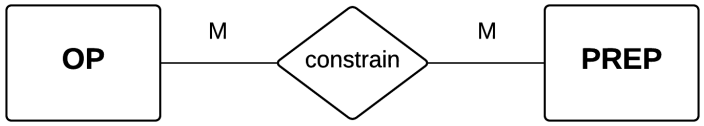

Figure 5.1: ER Diagram of operator and preperation

for example, can be

$$RANK\ \ PhoneX,\ \ PhoneY,\ \ PhoneZ\ \ WITH\ \ [\ 0.3,\ \ 0.4,\ \ 0.4\ ]$$

In the example, we use the operator "RANK" to rank the following 3 phones. However, it is meaningless to conduct the "RANK" operator over a list of target objects without giving any ranking rules. In this case, we need to add a "WITH" preperation. According to the definition of "PREP", it is responsible to constrain the behavious of the operator before or give the operator some kind of rules. We here use the "WITH" to provide the user's aspect weight vector. The length of the weight vector given should be the same as the size of the aspect set, otherwise, the parser will complain the error.

5.2 Compare

Considering the need of the producers to compare their product against their competitor, a compare operator is necessary. Basically, we compare two or more product by each of their aspect ratings and weights. We define our operator syntax:

COMPARE $\{product1\_name\},\ \{product2\_name\},\ ...\ \{productN\_name\}$

In this case, we can compare any set of N products with the "COMPARE" operator. And it returns a json object to describe the properties of each of the input products. It is very usefuly because it returns a list of aspect ratings and aspect weights ratings for each of the selected products. The analysis system plots the results into a bar chart, and the producer can clearly understand the pros and cons of their product against other competitive

productors. Although, the "COMPARE" operator is simple, it is powerful and useful in many scenarios. Imagine a company trying to attract more cutomers and beat its rivals, it will urgently need to know the pros and cons of each aspects comparing to its competitors' products. And later in Chapter 7, we will show a real application of "COMPARE" operator.

5.3 Weight Analysis

Another point the producer is likely to be interested in is the weight their customers put over their product when writing comments or reviews. By analyzing how much a reviewer weighs each aspect of a product, the producer can easily find out a best direction for feature development. Moreover, a simple aspect weight analysis can be achieved by running LARA and it cannot be very usefuly for a producer. In the new analysis system, we classify the reviews according to their upload timestamp by year. In this case, we can get a list of set of reviews:

$$
\begin{aligned}
Reviews = \{ & \\
& 2008 : [review1.1, review1.2, ...], \\
& 2009 : [review2.1, review2.2, ...], \\
& 2010 : [review3.1, review3.2, ...], \\
& \quad ... \\
& 2015 : [reviewN.1, reviewN.2, ...] \\
& \} \qquad (5.1)
\end{aligned}
$$

Then we run LARA on each set in the Reviews list, and get a list of aspect weight vectors for each product of different years. In order to retrieve information from the result we declare a "WEIGHT" operator to display the aspect weight vector. Since we integrate the year information in the LARA results, we need a constraint helper keyword "IN" to tell the system which period we need to retrieve. The syntax of the operator is:

WEIGHT {*product_name*} **IN** [{*begin_year*}, {*end_year*}]

In this case, it get the customers' focus by year and producers can easily predict the next year's trend. The output is a list of weight vector:

$$
\begin{aligned}
Reviews = \{ & \\
& 2008 : [aspect1Weight, aspect2Weight, ..., aspectKWeight], \\
& 2009 : [aspect1Weight, aspect1Weight, ..., aspectKWeight], \\
& 2010 : [aspect1Weight, aspect1Weight, ..., aspectKWeight], \\
& \qquad\qquad ... \\
& 2015 : [aspect1Weight, aspect1Weight, ..., aspectKWeight] \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \} \quad (5.2)
\end{aligned}
$$

Furthermore, the analysis system plot the ouput data on a 2-d coordinate system. "WEIGHT" is extremely useful in the real world for any company. All the companies are trying to improve their products by modifying or deleting some of the functionalities. In this case, the producers need to know in each year or half-year, what is the users' focus among all the aspect of a product. For instance, a phone producer find that more and more users tend to attach more importance to the screen size, it should design its next generation a big screen version. Further application of "WEIGHT" operator will be shown in chapter 7.

5.4 Rank

Lastly, we introduct a "RANK" operator, which is very powerful for the user of the analysis system to get a deeper insight over the advantages and disadvantages of their product against their rival. Actually, Ranking is an improvement version of the "COMPARE" operator. We could use it with the following syntax:

$$
\begin{aligned}
\textbf{RANK}\ & \{product1\_name\},\ \{product2\_name\},\ ...\ \{productN\_name\} \\
& \textbf{WITH}\ [\ \{Weight1\},\ \{Weight2\}, ...\ \{WeightK\}\ ]
\end{aligned}
$$

It takes in a list of product names and a normalized weight vector given by the user, the size of the weight vector equals to the size of asepct set. The

retrieval function calculate the ranking score based on the vector similarity of the product aspect rating vector and the weight vector given by the analysis system user. Usually, a dot product is enough

$$score = \sum_{i=1}^{k} aspectRating[i] weightVector[i]$$

CHAPTER 6

PIPELINING

6.1 Operator Pipeline

To build a mature and flexible analysis system, integrating operators and query operations alone is far from enough . Therefore, we decide to allow the users of the system to pipeline a list of operators. Pipelining not only makes the analysis system more flexible and extendable, but also provide users a more powerful tool to get deeper insight of the complex data with more complicated operations. Actually, pipelining of operators is not a new idea, and it has been used in some other fields [10]

For instance, if we are trying to rank a list of product, we can call the rank operator and get a list of results. But after that, if we are further interested in the customer reviews aspect weight analysis of the results of size k, we have to retype the "WEIGHT" query for k times, which is very annoying. Furthermore, if k is a very big number, it is unrealistic for a human being to query all of the items one by one. One of the solution for the stated problem is to create a new operator "RANK_WEIGHT", and it is a combinination of "RANK" and "WEIGHT". This is an expansion of the operator set and it seems like a reasonable solution. However, combination of operators are becoming very frequent when we are conducting some complicated analysis. Creating a new operator in this case can just increase the coupling of the system as a whole. Moreover, it decrease the flexibility of the tool.

Piplining design is even harder compared to the normal operator design in terms of ambiguity prevention and expressibility. Before putting our hands to design, we need to address some key problems in pipeline deisgn:

- For any operator OP_x in the pipeline, what is its input? And where should we get the input? Also, how can we use its output?

- For any two operators *OP_x* and *OP_y* in the pipeline, where *OP_x* is followed by *OP_y*, how should they interact with each other? And which interactions could make most sense for the analysis system.

- How can we design the syntax of the pipeline?

For the first two problems, the solution becomes clear if we try to consider them as a whole. We can take the output of the former operator as the input of the the latter operator for two adjacent operators in any pipeline query. However, under certain circumstances, this solution will break, because the output and the input of two different operators can be type incompatible. For example, type incompatible situation happens when we try to pipeline "RANK" and "WEIGHT" operator. the "RANK" operator returns a list of products , while the "WEIGHT" operator is designed to take only one product and output a json object describing the custors' aspects weights by year. Therefore, there will be a list vs object type conflict when pipelining the two operator. To solve the type conflict, we design the following two rules to constrain our pipeline actions:

1. if the output of an operator is a json /meta data describing the property a product, it can only be placed in the end of a pipeline.

2. if the output and input of two pipelined operator *OP1*, *OP2* come to a object(product) vs list(list of product) conflict, the input of *OP2* should compromise with the output of *OP1*.

The first rule is easy to understand. "WEIGHT" in our analysis system is an operator that outputs a json to describe the customers' aspect weight trend. And it is unreasonable to put it in the middle or the beginning of the pipeline. The second rule is described in a vague way in terms of generality. It means that if the output of the *OP1* is a list of k, and the input of *OP2* is an item, *OP2* should run k times on every items in the list of the output of *OP1*. Coming back to the previous example, if we have a "WEIGHT" followed by a "RANK", the "WEIGHT" should run over every product in the list of the output of the "RANK" operator.

Now we need to prove the completeness of the two rules in terms of combining operators. According to the description of the previous chapter, the input of a operator can only be:

- an item (item means product in our system)
- a list of item

The output of an operator is either one of the about two type or a json that describe a property of a product. Therefore, the above 2 rules are enough to cover all the situation of our current syntax of query.

We could further generalize the pipeline syntax as a list of operators. And we could write it in Context Free Grammar:

$$G = \{V, \Sigma, R, S\}$$

V is the non-terminal symbol, so in our analysis system V stands for the set of operators that output a list of items or an item, like "RANK". Σ is the terminal symbol, and in our analysis system Σ is the set of operators that output a json that describes the property of an item, like "WEIGHT". For the relation R, it is defined as:

$$\begin{aligned} S &= P \\ P &= \sigma \mid vP \end{aligned}$$

where $v \in V$ and $\sigma \in \Sigma$

6.2 Pipelined Query Syntax Design

In this chapter, we will analyze how we can design the syntax of the query with pipeline integrated. Similar to the previous section in this chapter, we need to build a robustic and complete rule to handle all the complex combination. However, it is very easy to design a terrible structure for the syntax without deep consideration.

For instance, a naive pipeline operator syntax design is:

$$\begin{aligned}\textbf{PIPELINE } \{\textbf{OP1}\}, \{\textbf{OP2}\}, ..., \{\textbf{OPn}\} \ \{product\} \\ \{\textbf{PREP1}\}\{constrain1\}, \\ \{\textbf{PREP2}\}\{constrain2\}, \\ ... \\ \{\textbf{PREPn}\}\{constainN\}\end{aligned}$$

This solution looks reasonable, however, it might is ambugious. In the previous chapter, we conclude that the relationship between **OP** and **PREP** is: *many-to-many*. Therefore, there is no way for us to figure out the mapping between operators and the constraints. Actually, it is a bad design if we write the list of operator together in one expression. Firstly, it disobey the "flexibility" principle we proposed in the previous section. The aim of operator combination is to make the system flexible and writing everything in one mess cannot achieve the goal. Secondly, even if we solve the ambiguity, it is highly unreadable for the analysis system user. Moreover, it it very likely for a human being to write a completely correct query once a pipelined query get long and complex.

Therefore, we need to do the pipeline query seperately. Since we decide not to write a list of query in one big query, we don't have a unified syntax form for the pipelined query as the single query. In our implementation, pipeline queries is conducted in seperate stages. A user can firstly conduct a "RANK" based query, get the result displayed on the front end. And then the user can choose whether he or she need to further pipeline another query over the existing results.

To distinguish a pipeline query and a normal query for the analysis system, we need to further extends our reserved keyword. We define the keyword "PIPELINE" as a special operator.

The key word should be add as the first term of as the middle of a pipelined query. For instance, after we run the "RANK" operator and have the result displayed in the frontend. We can type "PIPELINE WEIGHT" to get the aspect weight analysis of the results from the ranking. One optimization of the pipeline query is that we can filter some results from the output of the previous query as the input of its next query. For instance, if we just need

the top k items from the "RANK" query results, we can filter it. This idea will be further explored in the future work section.

CHAPTER 7

SAMPLE RESULT

For the evaluation of the analysis system, we crawl yelp academic data set as the input of LARA. It contains reviews for over 7500 businesses around nearly 30 colleges in United States written by students or residents nearby. To build a demo system, we select only restaurants out of the 7500 buisness. After data preprocess, filtering those restaurants with reviews less than our pre-set threshold, we finally get 3724 valid restaurants and 263196 reviews. We regard each restaurant as a product, and the owner as the producer. We predefine our aspect label set as:

$$Aspect = \{environment, taste, price\}$$

After running LARA over the data, we build backend database to store the results. For the front end of the system, we build a lexer to tokenize the query sentence, as well as a syntax parser to interpret them before sending the request to backend.

Figure 7.1 is the visualization result of rank operator, while Figure 7.2 and 7.3 is the results for compare and weight analysis.

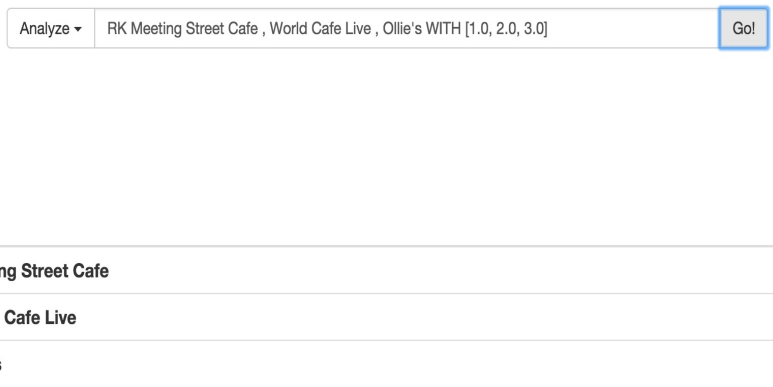

Figure 7.1: Result of Rank Operator

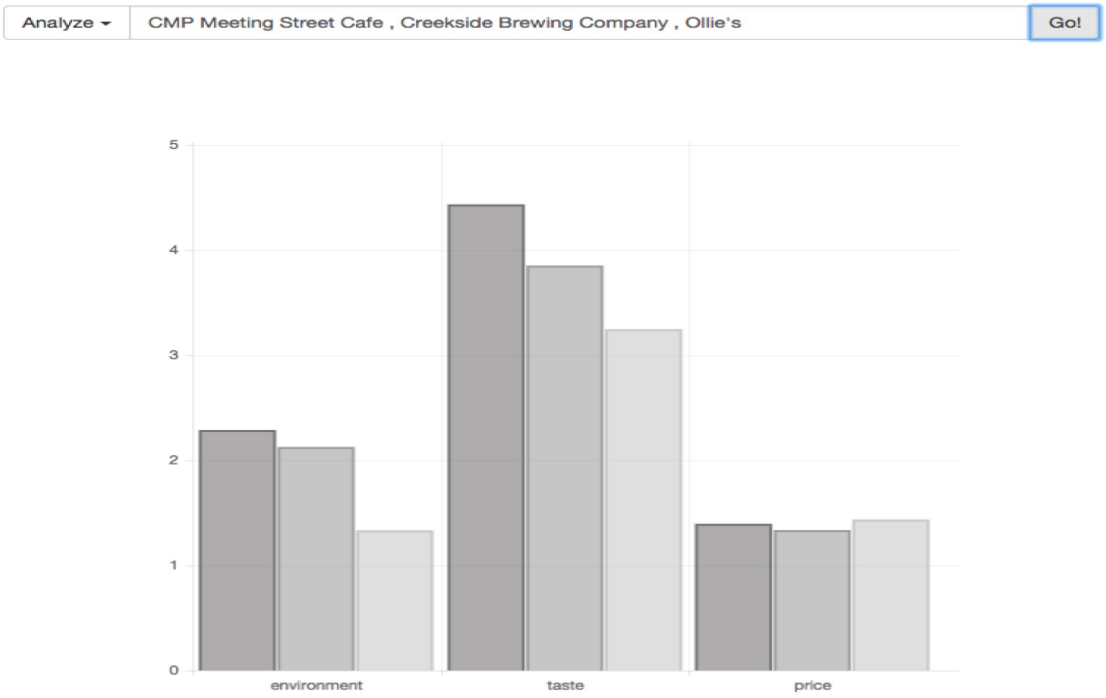

Figure 7.2: Result of Compare Operator

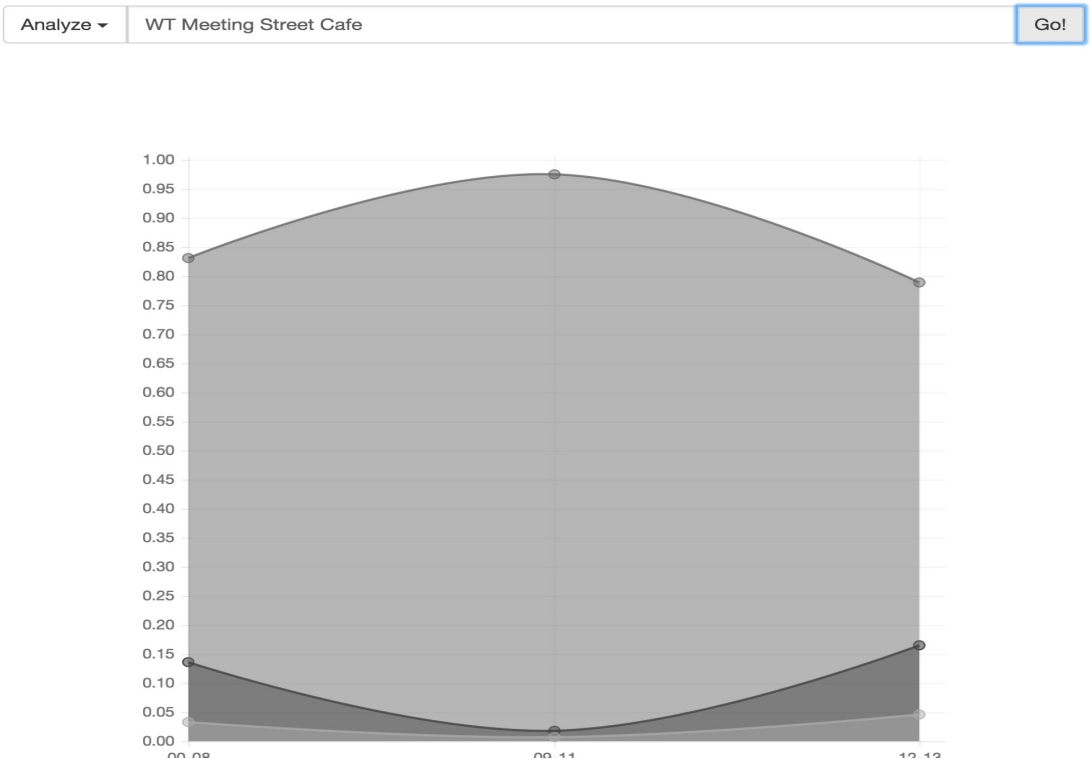

Figure 7.3: Result of Weight Operator

Imagine, after running the "RANK" operators over the data, which is shown in Figure 7.1, the user is interested in the comparision among the results of the "RANK" output. Then, he can type the query "PIPELINE COMPARE" to send the results to the input of "COMPARE" process. And the "COMPARE" results of the output of "RANK" will shown in Figure 7.2.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

In this thesis, we propose a novel reivew data analysis system for products based on the Latent Aspect Rating Analysis Model. LARA takes a set of reviews and an overall rating of each product and outputs the aspect weight vector, rating vector for each products. The analysis system is supported by the LARA output results. To make the system easy to use, we define a set of useful operators for the users to query the data. Also, the syntax for the operator is very expressive and simple and it is the challenging part of the thesis.

Overall, we define three types of operators: "RANK", "COMPARE" and "WEIGHT" for the operator set. And developers can further expand the operator set according to our predefined rules. Operators analyze the a product in multi-dimension. "COMPARE", as a very basic operator in the analysis system can help the producers the compare their product against any competitive products in each aspect. "RANK", as a advanced operator, can rank the list of products given by the user according to the user weight vector. "WEIGHT" operator can plot the tendency of user's focuses of aspects for a product within a certain period.

To make the system more flexible, we design a pipeline system for operators. A user can combine mutiple operators by conducting query in seperate stages. This allows the users to do very complex job in our analysis system, so that they can dig more information over the reviews.

The analysis system enables market prediction, business intelligence and user behaviour analysis. It provides the users with a operator set and query-based interface to retrieve and further analyze the results of LARA in a more efficient way. Moreover, the concept of the "operator set" make the

system more extendable, so that it can be further developed by adding new operators to meet the need of different analysis requirements. Furthermore, the application of operator pipelining can let the user learn most knowledge of the product review data with least effort. Also, it makes the analysis system more flexible and extendable.

8.2 Future Work

For future work, the operator interface of the analysis system could be further improved. We can integrate more operators except for "RANK", "COMPARE" and "WEIGHT" that can give us deeper insight of the data. Moreover, as for the visualization part, we could actually visualize more information other than query results. Since LARA could calculate the polarity value of the corpus vocabulary β. We are able to visualize the review text results from the searching engine. As for the pipeline part of the system, we could add filter for each "PIPELINE" operator . One of the idea is to introduce a "FILTER" keyword that select certain items in the output list. However, it would be nicer if we add graphic interface for the user, such as click box to filter the results.

REFERENCES

[1] H. Wang, Y. Lu, and C. Zhai, "Latent aspect rating analysis on review text data: A rating regression approach," *The 16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2010.

[2] M. Hu and B. Liu, "Mining and summarizing customer reviews," *The 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[3] N. Mishra and C.K.Jha, "Classification of opinion mining techniques," *International Journal of Computer Applications(IJCA 2012)*, October 2012.

[4] K. Dave, S. Lawrence, and D. M. Pennock, "Mining the peanut gallery: opinion extraction and semantic classification of product reviews," *WWW 03*, 2003.

[5] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," Stroudsburg, PA, USA, October 2013.

[6] B. Pang and L. Lee, "Opinion mining and sentiment analysis," 2012.

[7] B. Liu, *Sentiment Analysis and Opinion Mining*. Morgan and Claypool, May 2012.

[8] W. C. Mann and S. A. Thompson, *Rhetorical Structure Theory: A Framework for the Analysis of Texts*. PN, 1987.

[9] M. C. Lacity and M. A. Janson, "Understand qualitative data: A framework of text analysis methods," 1994.

[10] E. H. hsin Chi and J. T. Riedl, "An operator interaction framework for visualization systems," 1998.