HIERARCHICALLY STRUCTURED RECOMMENDER SYSTEM FOR IMPROVING NPS

by

Jieyan Kuang

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2016

Approved by:

Dr. Zbigniew W. Ras

Dr. Wlodek Zadrozny

Dr. Jing Yang

Dr. Angelina Tzacheva

Dr. Joseph Whitmeyer

ABSTRACT

JIEYAN KUANG. Hierarchically structured recommender system for improving NPS. (Under the direction of DR. ZBIGNIEW W. RAS)

Net Promoter System (NPS) is well known as an evaluation measure of the growth engine of big companies in the business area. The ultimate goal of my research is to build an action rules and meta-actions based recommender system for improving NPS scores of 34 companies (clients) dealing with similar businesses in the US and Canada. With the given original dataset, data preprocessing has been completed to result in better data representation and quality. The recommender system is built on top of a hierarchical clustering dendrogram which is generated by applying agglomerative clustering algorithm to a matrix of distance mixing semantic similarity and geographical distance by assigning weighted factors. To maximally expand the dataset of a single client by merging it with datasets of other clients under certain conditions, Hierarchically Agglomerative Method for Improving NPS (HAMIS) has been developed and applied to all clients respectively.

To extract meta-actions from customers' comments for triggering generated action rules and achieving desirable effect, a new text mining based strategy has been designed to accomplish tasks involving sentiment analysis, text summarization, feature identification and most importantly meta-action generation. Compared to other relevant works, our method has been proven to be more flexible and has achieved more satisfying results. Considering the fact that commonness, differential benefit and applicability exist when executing groups of meta-actions, a method for searching

best groups of meta-actions has been designed to boost the performance of applying meta-actions, which also improves the applicability and practicability of our system.

ACKNOWLEDGMENTS

There are a few people whom I would like to thank because I would not get this far without their help.

Firstly, I would like to thank my advisor, Prof. Zbigniew W. Ras, who has the greatest expertise and personality. I am not only grateful for his generous and patient guidance in my research area, but also admire his exceptional personal attitude to science, work and life. I learned a lot from him, his persistence and rigor as an advisor, as well as kindness and solicitude as an elder. I will keep learning from him, in professionality and personality, since he is the model that I should always look up to.

I would like to thank all the other members in my dissertation committee, they are Prof. Wlodek Zadrozny, Prof. Jing Yang, Prof. Angelina Tzacheva and Prof. Joseph Whitmeyer. Without their support and help, I couldn't defend my dissertation so soon.

Also, thanks to my great project team members Prof. Whitmeyer and Hualiu Yang, who provided helpful informations from the other angle of research during the whole time and useful advices for my own work.

I thank my working partners Albert Daniel and Doug Fowler who sponsored my research. I appreciate their trust in me and that they gave me this great opportunity to work on industry project during my Ph.D study. They provided great assistances to my work and also continuously inspired me from their angle of practical usefulness of my research. I couldn't get this work done without their help.

Last but not the least, I want to express my appreciation to my family and my friends. I thank my family and Ran for supporting me all the way, and my friends who help me so much that words would fail to convey my thanks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Introduction of the Project

Improving the performance of companies in the field of business is getting more and more important nowadays with no doubt, and the Net Promoter Score (NPS) is one of the most popular measures for such purpose. It is used to measure customers' satisfaction and loyalty to a product or service provider [47], and it is built on a scale 1 to 10 where 1 means very unlikely to recommend the provider and 10 means highly likely to recommend. Based on observations of customers' referral and repurchase behaviors along such scale, customers' are divided into three logic levels: promoter, passive and detractor, which present customers' satisfaction, loyalty and likelihood of recommending this provider in a descending order. Promoters are loyal enthusiasts who are buying from a company and urge their friends to do so. Passives are satisfied but unenthusiastic customers who can be easily taken by the competitors, while detractors are the least loyal customers who may urge their friends to avoid that company [45]. Customers are categorized into these three clusters based on their answers to the questions in questionnaires. Figure 1 explains how these three categories are computed.

Generally customers falling into interval 9-10 are seen as promoters, into 7-8 as passives, and into 0-6 as detractors. The partition into these three categories is widely

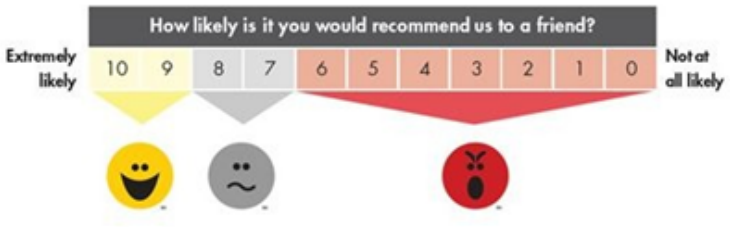


Figure 1: Categorization of Net Promoter Score (NPS)

accepted by business organizations but still other discretization of NPS can be taken into consideration especially when classifiers extracted from NPS datasets do not have acceptable precision/recall for one of these three categories. The classical way to evaluate the efficiency of a company's growth engine is to compute NPS efficiency rating which is defined as the percentage of customers who are promoters minus the percentage which are detractors. Companies with the most efficient growth engines such as Amazon, Costco, or Apple-iphone have NPS efficiency ratings between 50% to 80%. But even these top companies still have much room for improvement.

For the consulting company locating at Charlotte which we are currently working with, in order to collect information about customers' satisfaction in using services, customers are randomly selected to answer a phone questionnaire which is particularly designed to collect relevant information. Usually, each questionnaire consists of three categories of questions. The first and second groups of questions in the questionnaire collect the basic information about the customers and their service requests, and the third group, also the key part of the questionnaire, is about the customers' feelings on the service they got last time. Here are some examples of questions in these three groups:

- Information about the customer
  - name of the customer
  - contact phone number of the customer
- Information about the service
  - name of the client
  - invoice amount
  - type of equipment to be repaired
- Feeling about the service
  - how many days were needed to finish the job
  - was the job completed correctly
  - are you satisfied with the job
  - likelihood to refer to friends

All the responses from customers are stored in database, and each question is saved as one feature in the dataset, so more questions are answered by customers, larger dataset will be obtained for our analysis. But the reality is that we can't expect that customers will complete the entire questionnaire as we wish, so it is necessary to figure out the most important questions to ask. The entire dataset involves 34 sites, also called "clients", which locate in different areas crossing the United States as well as parts of Canada. Each client owns several shops which are geographically near each other. Customers are asked the same set of questions from the first and second

group, even though they are served by different clients. For questions in the third group, customers could be asked different sets of questions depending on the clients' preferences. In this group, customers are asked to give positive or negative feedbacks about certain aspects of provided service as mentioned in the sample questions. They assign scores ranging from 1 to 10 to a majority of questions in the third group, and the scores represent a customer's satisfaction level in an ascending order, higher the scores are, more pleased the customer is and more likely the customer would like to recommend this client to friends. Meanwhile, comments are also welcomed from customers when answering those questions. Based on the fact that clients could use different sets of questions in third group, there are over 70 questions included in the third group in the original dataset covering the questions used by all clients. Even the number of features relating to the third group is very large, the feature named $PromoterScore$ turns out to be the most distinguished one, because it is defined as the NPS. With regards to the value of $PromoterScore$ ranging from 1 to 10, customers can be divided into promoter, passive and detractor clusters as shown in Fig 1 and feature $PromoterStatus$ is created to store such results.

Overall, more than 25,000 customers who have been served by 34 clients were randomly chosen to answer the questionnaires in 2011 and 2012. There are 42,493 records in our dataset and this number is growing rapidly. Some of customers have been chosen multiple times during that period, while it doesn't necessarily mean that the customer who appeared only once in the dataset is no longer a customer. It is possible that customer may request that no call to him is wanted for a certain period of time in the future. Given the values stored in $PromoterStatus$, 99% of

records can be determined with the status while 1% missing values are due to the uncertainty in *PromoterScore*. As a result, the overall NPS efficiency rating for our entire dataset can be computed and it is equal to 0.76, as there are 34523 promoters and 2172 detractors, and 34523/42493 - 2172/42493 = 0.76. Moreover, the NPS efficiency rating can be computed for each client individually with the determined *PromoterStatus*, so each client can be targeted independently for achieving better performance.

To realize the ultimate goal of adopting proper actions to improve the performance of every single client, in another word, improve its NPS rating with the given dataset, Flexible Query Answering System (FQAS) is designed to accept queries from a client regarding how to serve customers better and giving actionable suggestions to resolve such queries. Hierarchically structured recommender systems are built with leaves of the tree representing personalized recommender systems. Each personalized recommender system is responsible for providing valuable action rules for its corresponding client. To make the quality of retrieved action rules as high as possible, Hierarchical Agglomerative Method for Improving NPS (HAMIS) is proposed to maximally extend the dataset representing each client by using data from some neighboring clients who have better NPS. It has been shown that the action rules generated from the extended dataset are more useful than from original dataset as they provide more options to clients, and have higher confidence.

In Figure 2, we show the main procedures of FQAS. As it shows, once a query from a client concerning the improvement of NPS ratings is submitted to FQAS, its corresponding recommender system will attempt to return action rules by following

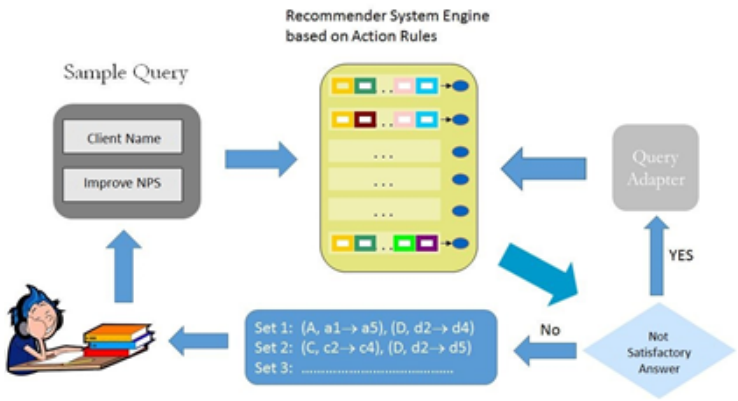


Figure 2: Flexible query answering system

HAMIS. If action rules are not returned to clients, Query Adapter in FQAS would take over and a relaxed query is submitted to the recommender system. At the end, action rules with their triggers will be returned to targeted clients and tested for its effectiveness.

1.2 Introduction of Recommender Systems

Recommender Systems have become a popular research area since the mid-1990s [22] [46] and they are used as the main engine for achieving our goal. Recommender systems were initially structured to give proper recommendations depending on customers' ratings in business area, based on the rapid development of e-commerce and on-line information, providing customers with advices about what to look, even purchase gains increasingly attention for bringing greater profits in reality. Typically speaking, recommender systems consist of three types: collaborative type, content-based and hybrid type. And they are formed along with development of machine learning techniques.

As the most well known type of recommender systems, collaborative or social fil-

tering recommender systems collect rating history from customers to form a general pattern on customers' habits and preferences, and recommend the items rated by other customers who have similar habits and preferences. The Ringo music recommender system [51] recommends music to users by get hints from other users having the similar preferences. However, the collaborative recommender systems rely on the amount of data collected from customers so bad, more ratings certain customers have made, better similarity estimation can be inspected between them. As a result, new customers won't get any good match until the system has learned their patterns quite well. Meanwhile, not only new customers, those customers who have different tastes could result in few recommendations due to matches can barely be found, so the social filtering system is extend to demographic filtering system which includes more demographic segment information, such as gender, location, education, etc. [38].

Content-based recommender systems are built on top of the classifiers derived from machine learning techniques and they solely focus on giving recommendations that are similar to the items in customers history records. Considering the issues of data size in collaborative systems, applying appropriate classifiers, like k-nearest neighbor classifiers, can resolve this problem to some extent in content-based system. Although content-based recommender systems are not as sensitive to size of initial dataset as collaborative systems are, they are still limited by the scope of recommendations, because the suggestions are given within the knowledge hidden in customer's history ratings, new customers won't get much benefit as there is little information in their history. Even for those who are not new customers, they will not get new, or surprisingly good recommendations but similar options as the customers have seen.

Actually, customers should be kept from not only those things that are too far away from what they have seen before, but also those items that are particularly similar to something they have already seen, which could cause some negative emotions in customers about the recommender systems. Daily Learner [6] has overcome this difficulty.

Regarding the advantages and drawbacks of collaborative type and content-based type, hybrid recommender systems are designed to combine these two types for avoiding the limitations in previous two types. And a hybrid recommender system can be implemented by combining separated collaborative and content-based recommender systems, adding content-based characteristics to collaborative models or reversely, or constructing a single recommender model [1]. In terms of the first method, the final recommender system can either select one individual system which provides better recommendations under given circumstances, like Daily Learner returns results with high confidence from either type of systems, or combine the results from both types of systems by linear combination [11] or voting scheme [38]. At the same time, recommender system [2] and the "collaborative via content" in [38] are hybrid systems which are built though including characteristics of content-based systems into collaborative models. By this way, the profiles about customers no longer just concentrate on customers' ratings, but also the features of items that customers rated to, which are content-based profiles. While other tools are on the foundation of content-based recommender systems with collaborative characteristics built in, for example, [52] created a collaborative view of customers where customers are described using semantic term vectors and it achieved good performance. But in recent years, more and more

efforts are put on building one single recommender by merging the characteristics from both types together, and many different approaches have been used, such as mixed rule-based classifiers in [3], a unified vector of user characteristics in [48], and knowledge based techniques in [9].

The improvement of recommender systems for providing better recommendations in more fields is urgent indeed, as much more business companies encourage such recommendations for attracting their customers. Researchers in this area still declare that the current methods used in recommender systems are in need of extension in a few aspects which includes comprehensive understanding of customers and items, extension of techniques and more [1] [10]. This project can be seen as another promising attempt of extending recommender systems with hierarchically structured model driven by action rules and meta-actions.

CHAPTER 2: DATA PREPROCESSING

Given the initial dataset, additional preprocessing procedures need to be performed on it to get a high quality dataset for mining. Values in dataset are collected from customers and manually entered into the database, it is imaginable that some inconsistency in format or simple errors could happen when typing these values. For example, there are instances written in Spanish instead of English, although they mean the same thing, still they are recognized as different values by machines. Such errors could lead to unreliable results. Besides these human made mistakes or inconsistency, the main attention has been put on data transformation, feature construction and feature selection in the following parts, which concern more about the essence of data [17]. Data transformation focuses on transforming the data from a complex format into a more convenient representation, either in symbolic, categorical or numerical, which lessens great burden for future work. Next, new features are created with extended information from original features, and this process can be referred as feature construction. Last but not least, feature selection is responsible for elimination of noise, redundant and irrelevant features and preservation of those highly correlated and essential instances, which is realized by computing correlation between features. Meanwhile, the process of feature selection can be used to evaluate the results from previous two steps. The product of data preprocessing will be a dataset that is clean, well-organized, and ready for further work [26].

2.1 Data Transformation

There are various types of data stored in given dataset, from categorical to numerical, date time to serial, etc. In the area of data mining, it is inconvenient for algorithms to deal with features in such a complex format, and it could be extremely time consuming and ineffective under such circumstances. At the same time, the information delivered by these features can't be ignored before further digging, so transforming them into a usual format that is easy for machines to manage is one of the most common ways to address this issue. Instead of applying some other complicated algorithms to process transformation, methods that are most fitting in current circumstances are more preferred. Examples of results from transformation are illustrated below.

The first example involves features stored in form of date time, and it includes features like $DateInterviewed$, $InvoiceDate$ and $WorkOrderCloseDate$. Since such features are in form of 'mm-dd-yy', sometimes 'yy-mm-dd'. However, regarding the nature of date time, days are countable and the number of days from the starting date to another certain date is another way to represent it by giving the length of time in a form of number of days. Thus, all the three features are transformed into numerical type which can be easily processed and still keeps the essence of date. In details, It is notified that the time frame in given dataset is from 2011 to 2012, so the starting date is set to be '12-31-2010' and the original data are replaced by numeric values calculated by counting the number of days from initial date. For instance, if a stored value is '01-28-2011', then the transformed value will become 28 as '01-28-2011'

minus '12-31-2010' equals to 28, meaning there are 28 days from the initial date to '01-28-2011'. By calculating the day differences as described, all the features stored in form of date time have been converted to numerical type which is more convenient for other measures, like standardization and discretization, etc.

Another representative example of transformation is phone numbers stored in feature $ContactPhone$. As we all know the phone number should be ten digits long, but the representation of phone numbers varies from case to case. In a great majority of instances, it is shown as 'xxx-xxx-xxxx', however, there are quite a lot of other instances recording phone number as numeric values directly without the hyphen, or including brackets like '(xxx)-xxx-xxxx', which cause a headache in transformation. Obviously, it is necessary to define a unified format to solve it like how date time is resolved. However, unifying all the phone numbers to numeric values doesn't make meaningful sense given the fact that numeric value fails in delivering the hidden knowledge in original format. It is a common sense that the first three digits of phone number are referred as area codes which indicate the customers locations approximately. Imagining that applying discretization in phone numbers which is in form of numeric values, it ends with meaningless results like interval [123456789, 234567890] which doesn't make any sense. While regarding the area codes from phone numbers, customers from the same area could be identified. Accordingly, numeric type is not suitable for features involving phone numbers, but with respects to the location information hidden in values of phone numbers, the first three digits of phone number are quite useful and they should be stored as categorical values by adding letters CP (Contact Phone) in front of them. As a result, data associated with phone numbers in such a

messed format are converted to categorical type with area codes labeled by symbolic letters. What's more, the location information inferred from the newly transformed feature is inspiring for constructing more features that might be potentially useful.

2.2 Feature Construction

Besides transforming features to a better format, constructing new features that present the extended information derived from the original features also addresses the problem of feature interaction and could provide better discriminative ability for building classifiers. Unlike data transformation, feature construction plays more effects on extracting the actual usage of features.

The first construction is inspired by the example of transformed $ContactPhone$. As described in the last section, the location information hidden in the area codes is very useful to identify customers' state location, as well as the the feature Zip which stores customers' zip codes in the form of numeric values. Using numerical values to represent zip codes causes the same problem as using numerical values for phone numbers. The first two or three zip codes can be used to estimate customers' state location, which is even better than using phone numbers for that purpose. Additionally, the majority of values in the features that are supposed to indicate customers' state location are missing, which urges the necessity of retrieving customers' location information from other forms like zip codes and phone numbers. Thus, a new feature is built by filling missing values for state information from zip codes first and phone number second.

Another example involves the feature named $UpdateContactName$ which contains

customers' newly updated contact names. In contrast to the massive names stored in it, the fact that whether a customer changed his/her contact name or not matters more to us, and the pattern of customers' behavior is more crucial for discriminating instances. Via consulting with our data consultants, it is agreed that the empty cells in $UpdateContactName$ are seen as non-updated cases, as well as when same values happen to $ContactName$ and $UpdateContactName$. Otherwise, the instances are seen as updated cases. In the new feature concerning if contact name is updated or not, "Yes" is assigned to the instances being updated cases while "No" is assigned to the ones belonging to non-updated.

2.3 Correlation-Based Feature Selection

In our dataset, each question in the questionnaire is treated as a feature. As mentioned in the introduction, there are over 70 features in the third group. In addition to the features in the first two groups, there will be over one hundred features in total in our dataset. At the same time, some of the features in the third group contain no value or several meaningless notes. Apparently, such features are useless for further data mining. Besides that, the existences of some other features could make no or little contribution to data mining tasks. Keeping those features in dataset not only makes the future work more difficult and inefficient, but also can result in poor results or even wrong conclusions. Therefore, constructing a representative set of features from original dataset by getting rid of noisy, redundant and irrelevant attributes without losing any valuable information is a first priority. Generally, feature selection consists of estimation of attribute utility and other evaluation with specific learning

methods, such as computation of correlation [20] [25] [59].

In terms of estimation of attribute utility, basically two aspects are included. Firstly, there are some features in the third group that are completely empty or contain no valuable information, those features should be removed from dataset directly because they provide nothing and the knowledge in dataset is not affected by their removal. Secondly, consultation with our data consultants provides the detailed interpretation of each feature, and the general process of collecting answers from customers. It turns out that some features in the third group are about other field of business that clients works on which are handled in a different way, and those features are not compatible to others and should be analyzed independently. Thus they are removed out from current dataset. Still, understanding all features literally is necessary but insufficient, as the knowledge hidden in the dataset is more important. Before continuing the following strategies of feature selection, those numerical features in the third group are temporarily kept out as advised, which is not only due to its large dimension, but also led by the consideration in their potential utility. As we believe that those features are supposed to be very useful in generating action sets and they present customers' answers in a more elaborate way by assigning ten ratings, it is necessary to analyze them solely with another specially designed strategy. So the following processes are focusing on the features from the first and second group, as well as four nominal features from the third group.

In statistics, correlation refers the relationship between any two random variables. By computing the correlation between features, knowledge on the dependence of pairs of features that are unknown outwardly are unveiled. Correlation measure is defined

as:

$$correlation(X,Y) = \frac{\sum_{i=1}^{n}\{(X(i) - \mathrm{X}') \times (Y(i) - \mathrm{Y}')\}}{(n-1) \times S(x) \times S(Y)} \quad (1)$$

Suppose the correlation between attributes X and Y is what about to be computed and there are n instances in the dataset. X' and Y' are the means of X and Y respectively, and $S(X)$ and $S(Y)$ is standard deviation of X and Y respectively.The correlation is computed as summation from 1 to n of the product $(X(i)-X')\times(Y(i)-Y')$ and then dividing this summation by the product $(n-1)\times S(X)\times S(Y)$ where n is the total number of examples and i is the increment variable of summation. But there still could be other definitions as well. Given this equation definition of correlation in Equation(1), the dependence between any pairs of features could be calculated [49]. For every pair of features, the generated correlation is a number between -1 and +1 that measures the degree of association between those two features. A positive correlation implies a positive relationship between two features, so features with large values tend to be closely associated with features with large values. Reversely, a negative correlation implies a negative relationship between features, which means features with large values tend to be associated with features with small values. Correlation based Feature Selection (CFS) is an algorithm that couples this evaluation formula with an appropriate correlation measure and a heuristic search strategy [19]. It is able to quickly identify irrelevant, redundant and noisy features and keep relevant features as long as they are not strongly correlated with other features.

It's already known the NPS status (promoter, passive and detractor) stored in *PromoterStatus* in dataset is the key criterion to evaluate customers' satisfaction.

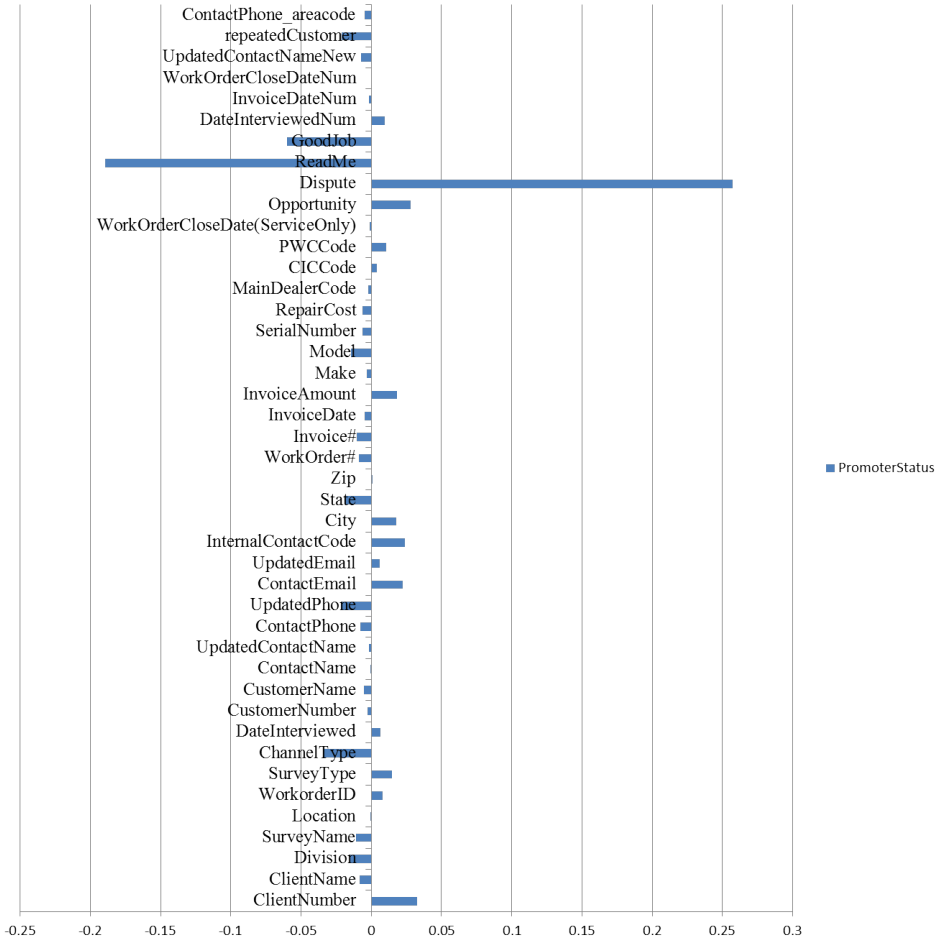


Figure 3: Feature correlations between *PromoterStatus* and other features calculated from Equation 1.

As the most distinguishing feature in dataset, its correlation with other features in the dataset matters more to us. If some features are more highly correlated with *PromoterStatus*, then these features are more preferred and the correlation between *PromoterStatus* and all the other features are shown in Figure 3. From the figure, some features are found significantly highly correlated with *PromoterStatus*, like *GoodJob*, *ReadMe* and *Dispute*. However, the features in dataset should be less correlated with each other but with high correlation with *PromoterStatus*, so some features are excluded which is due to redundancy. At the end, weighting by cor-

relation, the final determined set of features in our dataset includes: $ClientName$, $SurveyType$, $Division$, $ChannelType$, $InvoiceAmount$, $Make$, $CIC\ code$, $PWC\ code$, $Opportunity$, $Dispute$, $ReadMe$, $GoodJob$, $DateInterviewedNum$, $InvoiceDateNum$, $WorkOrderCloseDateNum$, $UpdatedContactNameOrNot$ and $PromoterStatus$.

CHAPTER 3: INITIAL CLASSIFICATION

After the entire dataset is preprocessed, the dataset is clean, well formated and ready for further analysis. As a primary factor for building the system, constructing the best classifiers will guarantee mining high quality action rules. Additionally, the results of performing classification on the dataset provide a glance at the consistency of knowledge hidden in dataset. Better classification results are achieved, more consistent knowledge is stored in our dataset. However, it turns out the classifiers built from current dataset are still too poor to meet our satisfaction, which could be led by lack of sufficient information associated with the decision attribute. Thus, the process of expending the dataset by adding more helpful features is used to resolve the issue of poor classification result. At the end, the best extension of current dataset will be formed and the best classifier will be built.

3.1 Selection of Best Classification Algorithm

To build the classifiers in a convenient way, WEKA is selected to be the main tool, because of its effective, comprehensiveness and ease in using. WEKA is one of the most popular tools in the field of data mining and machine learning and it is implemented by the machine learning group from University of Waikato. Its widely acceptance and usage are basically due to its collection of various well-known machine learning algorithms which can be directly applied with its GUI or called from Java

code. Besides containing main algorithms, WEKA is a powerful software that has multiple modules covering data preprocessing, classification, clustering, visualization and so on. For the classification algorithms, there are a lot of well known algorithms available in it [35] [55] [4], such as $J48$ ($DecisionTree$), $Random\ Forest$, $Nearest\ Neighbor$ (KNN) and $Naive\ Bayes$, etc. However, one limitation of using WEKA here is the size of dataset that it can successfully process. Especially when building classifiers using tree structured algorithms, the system couldn't end properly due to the burdensome computation with oversize datasets. Actually this shortage in WEKA has become a common issue nowadays and its implementers even include another module to create samplings for handling the cases of large datasets. Thus, it is believed that random samplings are enough for our purpose and 10 samplings are generated from the entire dataset for the following tasks. Each sampling contains 1,200 instances and all of them will be processed using available classification algorithms in WEKA by Java codes. To evaluate the performance of each algorithm, confusion matrix is used and it is computed by averaging the 10 confusion matrices from 10 random samplings covering 34 clients. Confusion matrix summarizes the results of classifying instances into decision classes promoter, passive and detractor visually, more instances are assigned to correct classes, more accurately this classification is processed by certain algorithm, better this algorithm performs.

Before discussing the tests on various algorithms, a representative example of comparing classification results by $J48$ and $Random\ Forest$ is demonstrated in Figure 4. They are comparable because if $J48$ is seen as on decision tree model, then $RandomForest$ is an ensemble classifier using many decision tree models. Decision

J48	Promoter	Passive	Detractor	Recall	Precision
Promoter	151.8	343.4	0	0.307	0.449
Passive	138.4	361.6	0	0.723	0.422
Detractor	47.8	152.2	0	0	0
				Accuracy = 0.43	
				Total time: 4 secs	

RandomForest	Promoter	Passive	Detractor	Recall	Precision
Promoter	253.6	215	26.6	0.512	0.451
Passive	222.2	245.6	32.2	0.491	0.439
Detractor	85.6	98.8	15.6	0.078	0.209
				Accuracy = 0.4307	
				Total time: 67 secs	

Figure 4: Comparison of classification performances by *J*48 and *Random Forest*

Tree is a widely known classification algorithm because it is easy to interpret the rules once the tree is created and the process is quite fast [33]. By consisting of multiple decision tree models, *Random Forest* is less prone to overfitting which solves the drawback of decision tree, and its proximity measure is used to fill in missing data and calculate outliers. But it is expected that it will take more time to finish the work [8] [54]. The top table is the average confusion matrix and relevant figures including *Recall*, *Precision*, *Accuracy* and *Total time* from *J*48, while the bottom one is from *RandomForest*. *Recall* can be understood as the accuracy of instances actually classified correctly in each class in decisive attribute and *Precision* is the accuracy of instances hypothetically classified correctly in each class in decisive attribute. *Accuracy* is the fraction of instances that are classified correctly regardless of classes in decision attribute, and *Total time* is the length of time this algorithm takes to finish the classification with 10 random samplings.

In the confusion matrix which is labeled by blue, the number in each row indicates the number of instances that are actually promoters, passives or detractors, and the number in each column shows the number of instances that are assigned or predicted as promoters, passives and detractors respectively by algorithms. It is known that all the correctly predicted instances are located in the diagonal of confusion matrix, so greater numbers in its diagonal, greater number of instances are classified accurately, higher the accuracy will be. As the table shows, there are approximately 500 promoters and passives and 200 detractors in each random sampling, and the classification on *Detractor* is disappointing with both algorithms, while the accuracy on *Passive* is much better, 72.3% with $J48$ and nearly 50% with *Random Forest*. The poor prediction on *Detractor* implies that the descriptions of detractors are blurred to passives, even promoters by these algorithms, but they are labeled as detractors by following customers' responses, so it encourages more curiosity on why it happened and how to clearly distinguish the detractors from passives and promoters in our dataset, which is useful to identify movements for improving detractors. To compare the performance of these two algorithms shown in Figure 4, the main factors being considered are the accuracy of classification and length of time it takes, which assure the classification results is both effective and efficient without wasting resources. For example, the *Recall* of passives by $J48$ is 72.3%, the highest figure in both tables, while that of promoters by *Random Forest* is higher, which is 51.2% compared to 30.7% in table of $J48$. Thus, both of them are good at something while bad at other areas, *Accuracy* accesses the overall performance on correctness, so it is selected as one of the main criterion. Although *Random Forest* wins in *Accuracy* by 0.07%, it

is totally held back by its time consuming issue as shown in the figure, 67 seconds comparing with 4 seconds by $J48$. As costing too much time is one of the last things we want, it is not desirable or affordable in case with larger dataset, and the difference between the accuracy performed by them is too minor to take count of, $J48$ wins the competition against *Random Forest* in this comparison.

After all the most representative algorithms have been applied to the same 10 random samplings, all the relevant figures are analyzed and compared in the same way as introduced in the example of $J48$ and *Random Forest* above. Generally, the classification algorithms can be arranged into different categories, such as rule-based, tree-based and probabilistic-based. And the most representative one in each category is chosen to compare their performance and they are listed below [35].

PART Use rules to classify the data in test dataset.

RBF (Radial Basis Function) It utilizes normalized Gaussian radial basis function network and the advantage is that it finds the input to output map using local approximators.

BayesNet It can be seen as a Naive Bayes classifier where class has parents and each attribute has no class as its sole parent.

NaiveBayes Statistical classifiers based on Bayes' theorem, they predict the probability that a record belongs to a particular class.

KNN (K-Nearest Neighbor) It is based on learning by analogy. The training samples are described by n dimensional numeric attributes and each sample represents

a point in an n-dimensional space.

RandomForest It is a extension version of decision tree model, and it solves the overfitting problem in decision tree model with much more time taken.

J48 It is a decision tree model and it is built by recursively splitting the training dataset based on a optimal criteria until all records belonging to each of the partitions bear the same class label. These trees are built relatively fast, obtaining similar or often better accuracy.

To identify the best classifiers for the samples representing the entire dataset, the key factors, *Accuracy* and $TimeTaken$, are collected by performing each one of the representative algorithms and are shown in Figure 5. In the chart, blue bars represent the length of time each algorithm has taken to finish the work, and orange bars show the accuracy of classifiers performed on samplings. In terms of accuracy, it is quite clear that *Random Forest* gives the best prediction, but the length of time it has taken is the longest, which is 67 seconds and remarkably longer than all the other algorithms. While $NaiveBayes$ has been the most efficient one as it has taken only 3 seconds to finish, but the low accuracy fails making it become our best option. $J48$ seems to balance both time and accuracy in a good way.

Table 1: Comparison of scores for different classification algorithms

	PART	**RBF**	**BayesNet**	**Naive Bayes**	**KNN**	**Random Forest**	**J48**
Accuracy	2	3	4	1	5	7	6
Time Taken	2	4	6	7	3	1	5
Total Score	4	7	10	8	8	8	11

Regarding the fact that none of them wins in both criterion, we have to select

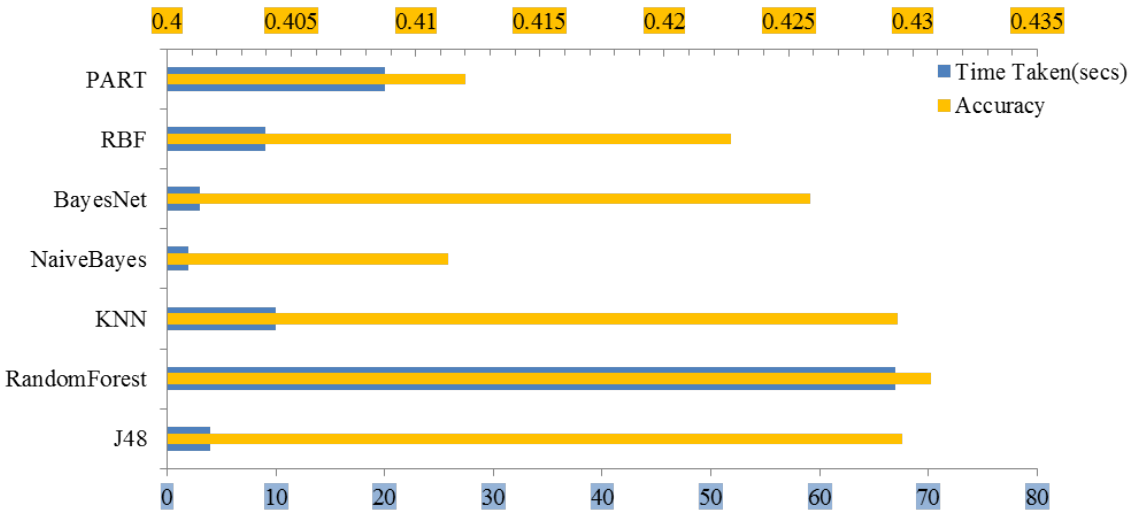


Figure 5: Comparison of performances by different classification algorithms

the one that performs well in both aspects without significant disadvantages to build the best classifiers. The Table 1 is created to offer a clear view of selecting the best algorithm by assigning scores to each one of them. The scores are given with respects to the rank of each algorithm in aspect of *Accuracy* and *TimeTaken*, higher certain algorithm ranks in one criteria, higher score will be assigned to it [33]. So the highest score in each criteria is 7 and it should be given to the one with highest accuracy or shortest time. Accordingly, the lowest score 1 is assigned to the one with lowest accuracy or longest time. Based on Figure 5, it is obvious that *Random Forest* gets 7 in *Accuracy* and so should *Naive Bayes* for the criteria of time consumed. And it is quite ironic that both of them are assigned with the lowest score in the other criteria, so the total score for *Random Forest* and *Naive Bayes* are both 8. It turns out *J*48 achieves the highest total score in contrast to all the other algorithms, which is within our expectation because of its balanced performance in both aspects. Therefore, *J*48 is determined to be the best classification algorithm and it will be

used for the classification work in this thesis.

3.2 Improvement in Classification

From the initial classification result produced by *J*48, even it produces the best performance among all the other algorithms, the confusion matrix and accuracy are still too lousy to get well prepared for the following analysis work. So improving the classification result becomes a vital role in the whole process and several methods that are best fitting in our situation are deployed as illustrated in details below.

3.2.1 Construction of Hierarchical Features

In previous chapter, we already know that the features involving date time have been transformed into numeric features which indicates the number of days from the starting date, and those features can be referred as *day* features. By looking into the nature of date time values, it is easy to realize the hierarchy hidden in the values of *day* features which can be defined as *day*, *month* and *season* in an ascending order of hierarchical level, the higher its hierarchical level is, more general the description of certain pattern hidden in the dataset will be, consequently better classification result can be expected. Therefore, new hierarchical features involving date time are constructed as *month* features and *season* features in the same 10 samplings and both of them will be tested with *J*48 in WEKA respectively.

Instead of including factor *TimeTaken* which is unnecessary in current circumstance, F-score is used as the main factor for evaluating the performance with different forms of features. F-score is another measure of a classification's accuracy and here the F-score is a weighted average of *Precision* and *Recall*, so higher F-score

DayFeatures	*Promoter*	*Passive*	*Detractor*	*Recall*	*Precision*
Promoter	151.8	343.4	0	0.307	0.449
Passive	138.4	361.6	0	0.723	0.422
Detractor	47.8	152.2	0	0	0
				Accuracy = 0.43	
				F-score = 0.374	

MonthFeatures	*Promoter*	*Passive*	*Detractor*	*Recall*	*Precision*
Promoter	107.4	387.8	0	0.216882	0.45202
Passive	97.2	402.8	0	0.8056	0.420635
Detractor	33	167	0	0	0
				Accuracy = 0.4268	
				F-score = 0.353	

SeasonFeatures	*Promoter*	*Passive*	*Detractor*	*Recall*	*Precision*
Promoter	210.8	284.4	0	0.425687	0.448702
Passive	191.8	308.2	0	0.6164	0.424869
Detractor	67.2	132.8	0	0	0
				Accuracy = **0.4342**	
				F-score = **0.391**	

Figure 6: Comparison of classification with *day*, *month* or *season* features

is more desirable in our experiments. Similar as what has been done with tests on different algorithms, classifications on samplings with *day*, *month* or *season* features have been completed solely, and average confusion matrices and relevant figures from each test are presented in Figure 6 in a top down order. The first table is generated from initial dataset with only *day* features which is same as the one shown in previous section. The second table is generated from dataset with *month* features, and the accuracy is 0.4268, while the F-score is 0.353, which is the lowest among them and

didn't match our expectation. The last table is from the samplings with season features alone. As it shows, despite the prediction in detractor it is not improved quite much, the accuracy is slightly raised with 0.0042 comparing to the first table. And the F-score, 0.391, is much higher than any of the other two forms. Theoretically, the better form of features results in better classification results, in another word, in better F-score. Hence, it is quite obvious that *season* is the best form for date time, as it produces the highest value in both accuracy and F-score.

3.2.2 Normalization and Discretization

In the area of training dataset, normalization and discretization are two popular data processing approaches and they will be applied to the samplings for checking if they help improve classification result or not. In terms of normalization, it is a process of transforming numerical values which are currently in different scales into a common scale [50]. And it is carried out like this:

1. Mean absolute deviation S_f

   $S_f = \frac{1}{n} * (|X_{1f} - M_f| + |X_{2f} - M_f| + ... + |X_{nf} - M_f|)$

2. Normalization measurement

   $Z_{if} = \frac{(X_{if} - m_f)}{S_f}$

In the formula above, let's assume X_{if} is a i_{th} value of a feature f, and M_f is the mean value of feature f. So the first formula tells us how to compute the standard deviation S_f for certain feature f and use S_f to compute every normalized value Z_{if}. Given this process of normalizing numeric attributes, classifications on samplings

before and after normalization have been accomplished and the average confusion matrices and other figures are shown in Figure 7. In the figure, the table on the top is from samplings without normalization and the bottom one is from samplings with normalization. It is obvious that the accuracy and F-score in the bottom table is slightly higher than the top one, which is 0.4352 and 0.3928 respectively. Thus, we can conclude that normalization is helpful for classification and it should be applied whenever it is needed.

Without stdz	*Promoter*	*Passive*	*Detractor*	*Recall*	*Precision*
Promoter	210.8	284.4	0	0.425687	0.448702
Passive	191.8	308.2	0	0.6164	0.424869
Detractor	67.2	132.8	0	0	0
				Accuracy = 0.4342	
				F-score = 0.391	

With stdz	*Promoter*	*Passive*	*Detractor*	*Recall*	*Precision*
Promoter	214.6	280.6	0	0.43336	0.45217
Passive	194.4	305.6	0	0.6112	0.424091
Detractor	65.6	134.4	0	0	0
				Accuracy = 0.4352	
				F-score = 0.3928	

Figure 7: Comparison of classification without/with normalization

The second attempt in this part is performing discretization. Theoretically speaking, the process of discretization is to cut off the values of features (usually continuous features) into several discrete intervals which can form more distinctive descriptions in information system [14]. In our circumstances, Rough Set Exploration System

(RSES) is more preferred to perform discretization than WEKA. The supervised discretization algorithm implemented in WEKA is Multiple Interval Discretization Algorithm which cuts the values by following the Minimum Description Length Principle (MDLP) [34]. But it only accepts fixed number of intervals for all numerical features, which leads to the inflexibility of deciding the best cut for specific attributes with regards to its own distribution of values. Additionally, it only handles the numerical attributes. In contrast to WEKA, discretization using RSES is more dynamic and comprehensive. By more dynamic, it means RSES permits discoveries of cuts for each attribute independently; by more comprehensive, it means RSES covers all the attributes, including both numerical and symbolic. The process of discretization converts the original table into a simplified one with less complexity but same richness of semantic information. The known strategy of decision algorithm generation is also based on MDLP, but the methods used in RSES are broader and the existing methods can be distinguished via criteria of local and global [5]. Simply speaking, local method produces partitions that are applied to localized regions of object space, while global method deals with all attributes and each attribute value set is partitioned into intervals independent of the other attributes [14]. Apparently, RSES is more proper under this circumstance, and same 10 random samplings are applied using RSES to check the performance of discretization on classifying the dataset.

In Figure 8, average confusion matrix and relevant figures collected from classifications on samplings with discretization performed are shown in the bottom table. The new factor *Coverage* is listed in it which indicates the percentage of instances that have been inspected by the classifiers in each decision class. For example, in the

Without discretization	*Promoter*	*Passive*	*Detractor*	*Recall*	*Coverage*
Promoter	8.7	8.4	2.28	0.448916	0.392
Passive	7.88	8.16	2.54	0.439182	0.3732
Detractor	2.36	2.64	1.2	0.193548	0.3136
				Accuracy = 0.409	
				F-score = 0.381	

With discretization	*Promoter*	*Passive*	*Detractor*	*Recall*	*Coverage*
Promoter	45.06	0.28	0.3	0.987292	0.9248
Passive	0.3	45.48	0.14	0.990418	0.923
Detractor	0.28	0.18	16.54	0.972941	0.8556
				Accuracy = 0.986	
				F-score = 0.834	

Figure 8: Comparison of classification without/with discretization

confusion matrix generated from dataset without discretization, if looking at the row for promoter, there are 8.7+8.4+2.28 = 19.38 instances that are recognized by the system, which occupy only 39.2% of the total number of instances and is a very low coverage. It is easy to observe that the accuracy, F-score and all the other figures in the bottom table are extremely high and over twice than without discretization, which is far beyond our expectation but still makes sense. So discretization plays such exceptionally positive roles in classification.

3.2.3 Addition of New Features

Besides transforming the values in dataset as the normalization and discretization have covered, extending current dataset dimensionally is another potential option for improving classification result. The idea of well-known strategy MDLP is to keep

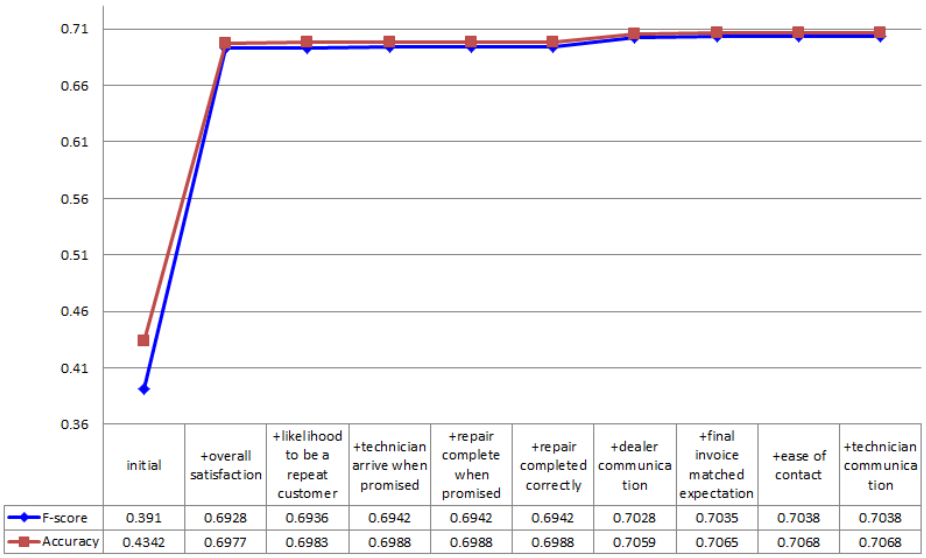
the minimum descriptions of objects but preserve the same information as contained in a more complex description. On the contrary, here what has been suggested is increasing the dimension of dataset by reasonably adding some more features in a way that they gets the classification maximally improved. This idea is built on the believe that adding more features potentially relating to decision attribute could fill more valuable information into dataset, consequently the description of each decision class could be formed less ambiguously and then classification could be improved [13]. But the problem of which attribute to add is our first concern.

As described in the first chapter, features in the first two groups store the basic information about clients and customers, there is not much space to enlarge in these groups because such information is not changeable, it is useless to put much attention on them. However, the space for adding features from the third group is still adequate, as only four nominal features from third category are included in current dataset, and it is time to consider adding some new from the rest of numerical features which are potentially valuable. These numerical features demonstrate customers' response in a rating scale from 1 to 10, 10 means the customers is extremely happy about this service and 1 means he/she is not pleased, even angry with what has been done, higher the score is, more satisfied the customer is. As mentioned, there are over 70 numerical features in the third category covering questions asked by all the clients, but some of them are applicable to several clients while some are used by all clients. At the same time, some features are relevant with other businesses that clients are dealing with in a different way, which is not proper for our purpose. Thus, the numeric features that are used by all clients and are related to the business area we focus at are defined

as *common features*, and 9 common features are selected to be the candidates for being added into dataset after consulting with data experts.

The purpose of adding new features into current dataset is to improve its classification result, so a feature should be kept as long as the main factor, F-score, is improved, at least not damaged after classifying the dataset with such feature. The strategy of testing the qualification of all the 9 features is to apply $J48$ in WEKA to samplings from the entire dataset with one new feature added a time and check the generated F-score, if F-score is improved, at least not damaged, then this added feature should stay in the dataset and the newly expended dataset will be used for the following rounds; otherwise, this feature shouldn't be added and the dataset without this feature will be used for the next round. This procedure will repeatedly executed until all the candidates are checked and the resulting dataset will be extended in dimensions maximally with the best classifiers. In this case, sequence of adding features doesn't affect the final result since every one of them probably will be tested along with other features after all.

To present the procedure of applying the extending strategy in details, a line chart is created and shown in Figure 9. In the figure, the lines show the changes of F-score and accuracy from classifications on samplings in blue and red respectively whenever a new feature is added into the dataset by following the order from left to right at the horizontal axis. On the bottom of the chart, a table contains exact values of F-score and accuracy from the classification on samplings with corresponding feature in each column. By monitoring the trend of blue or red line, it is easy to perceive that their values are quite close to each other all the way up with red line representing accuracy been a



	initial	+overall satisfaction	+likelihood to be a repeat customer	+technician arrive when promised	+repair complete when promised	+repair completed correctly	+dealer communication	+final invoice matched expectation	+ease of contact	+technician communication
F-score	0.391	0.6928	0.6936	0.6942	0.6942	0.6942	0.7028	0.7035	0.7038	0.7038
Accuracy	0.4342	0.6977	0.6983	0.6988	0.6988	0.6988	0.7059	0.7065	0.7068	0.7068

Figure 9: Process of adding new features into dataset

little bit higher than blue line, and generally they are going up while staying steadily for a few moments during the whole process. The sharpest raise for both lines happens to the extension with $OverallSatisfaction$ which leads to F-score dramatically growing by nearly 78%, from 0.391 to 0.6936. Obviously, $OverallSatisfaction$ should be kept in dataset. After that, the lines go flatly for a while, which indicates F-score and accuracy increase gradually, even stay at the same when dataset is extended with features including $Likelihoodtobearepeatcustomer$, $TechnicianArriveWhenPromised$, $RepairCompletedWhenPromised$ and $RepairCompletedCorrectly$. Still they stay in dataset because none of them makes F-score or accuracy drop. Then another representative raise in chart appears with addition of $DealerCommunication$, accuracy and F-score are improved gently by 1.2% which is relatively remarkable compared with the increment by other features. Later adding $FinalInvoiceMatchedExpectation$ also increases the F-score slightly, while the extension with left features added makes the classification results stay the same. In the end, it is not surprising that all the 9

features are successfully added into the dataset, even though some of them don't improve the classification result at all, they don't damage the classification. Generating same classification results could result from the possibility that similar patterns of semantic knowledge hidden in data that have been already discovered by classifiers, including those features also provides a great variety of information for evaluating customers' feeling.

Without benchmarks	*Promoter*	*Passive*	*Detractor*	*Recall*	*Precision*
Promoter	210.8	284.4	0	0.425687	0.448702
Passive	191.8	308.2	0	0.6164	0.424869
Detractor	67.2	132.8	0	0	0
				Accuracy = 0.4342	
				F-score = 0.391	

With benchmarks	*Promoter*	*Passive*	*Detractor*	*Recall*	*Precision*
Promoter	394.7	92.9	7.6	0.797052	0.76433
Passive	102.9	346.3	50.8	0.6926	0.669696
Detractor	18.8	77.9	103.3	0.5165	0.638837
				Accuracy = 0.7068	
				F-score = 0.7038	

Figure 10: Comparison of classification results from samplings without/with new features

Comparing the classification results on samplings before and after these features are added into our dataset as shown in Figure 10 is sufficient to evaluate the quality of enlarged dataset. In the figure, the table on the top is from the dataset without any new features and the table on the bottom is from the finalized dataset with all new features added. The first thing that attracts us is the extraordinary progress in

classifying detractors, instead of none of detractors are classified correctly in the initial case, over half of them are seen properly with more valuable information provided. The accuracy of classification on the finalized dataset is 0.7068, which is 62.68% higher than the data before extension. Meanwhile, F-score from finalized dataset is 80% greater than the initial one. Apparently, performing the extension with new features in previous dataset is fruitful regarding the significant improvement in classification.

CHAPTER 4: INITIAL ACTION RULE MINING

4.1 Introduction of Action Rules

Action rules mining is a critical method in the area of data mining and it was firstly proposed by Ras and Wieczorkowska in [42] and investigated further in [21], [36], [44], [39]. Action rule suggests what is the smallest set of necessary actions needed for switching from current state to another within the states of decision attribute. Decision attribute is a distinguished attribute [42] while rest of the attributes are partitioned into stable and flexible categories. So in our domain, decision attribute is the attribute which *Promoter*, *Passive* and *Detractor* are referring to. As the name implies, values of flexible attributes can be changed, then attributes which tell the answers to questions in third categories are seen as flexible attributes. In early papers, action rules have been constructed from two classification rules $[(\omega \wedge \alpha) \rightarrow \phi]$ and $[(\omega \wedge \beta) \rightarrow \psi]$, where ω is a stable part for both rules [41] [60]. Action rule was defined as the term $[(\omega) \wedge (\alpha \rightarrow \beta)] \Rightarrow (\phi \rightarrow \psi)$, where ω is the description of clients for whom the rule can be applied, $(\alpha \rightarrow \beta)$ shows what changes in values of flexible attributes are required, and $(\phi \rightarrow \psi)$ gives the expected effect of the action. Let us assume that ϕ means *detractors* and ψ means *promoters*. Then, the discovered knowledge shows how values of flexible attributes need to be changed under the situation required by stable attributes so the customers classified as detractors will become promoters. In

this section we introduce the classical strategy of constructing action rules from action sets.

By an information system [37] we mean a triple $S = (X, A, V)$, where:

1. X is a nonempty, finite set of objects

2. A is a nonempty, finite set of attributes, i.e.

   $a : U \longrightarrow V_a$ is a function (can be partial function) for any $a \in A$, where V_a is called the domain of a

3. $V = \bigcup\{V_a : a \in A\}$.

For example, Table 1 shows an information system S with a set of objects $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$, set of attributes $A = \{a, b, c, d\}$, and the set of their values $V = \{a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2\}$.

Table 1 : Information System S

	a	b	c	d
x_1	a_1	b_1	c_1	d_1
x_2	a_2	b_1	c_1	d_1
x_3	a_2	b_2	c_1	d_2
x_4	a_2	b_2	c_2	d_2
x_5	a_2	b_1	c_1	d_1
x_6	a_2	b_2	c_1	d_2
x_7	a_2	b_1	c_2	d_2
x_8	a_1	b_2	c_2	d_1

Additionally, we assume that $A = A_{St} \cup A_{Fl}$, where attributes in A_{St} are called *stable* and attributes in A_{Fl} are called *flexible*. "Customer name" is an example of a stable attribute. "Interest rate" for each customer account is an example of a flexible attribute.

Let $S = (X, A, V)$ is an information system, where $V = \bigcup\{V_a : a \in A\}$.

By an *atomic action set* we mean a singleton set containing an expression $(a, a_1 \rightarrow a_2)$ called atomic action, where a is an attribute and $a_1, a_2 \in V_a$. If $a_1 = a_2$, then a is called stable on a_1. Instead of $(a, a_1 \rightarrow a_1)$, we usually write (a, a_1) for any $a_1 \in V_a$.

By *Action Sets* we mean a smallest collection of sets such that:

1. If t is an atomic action set, then t is an action set.

2. If t_1, t_2 are action sets, then $t_1 \cup t_2$ is a candidate action set.

3. If t is a candidate action set and for any two atomic actions $(a, a_1 \rightarrow a_2)$, $(b, b_1 \rightarrow b_2)$ contained in t we have $a \neq b$, then t is an action set.

By the domain of an action set t, denoted by $Dom(t)$, we mean the set of all attribute names listed in t.

By an *action rule* we mean any expression $r = [t_1 \Rightarrow t_2]$, where t_1 and t_2 are action sets. Additionally, we assume that $Dom(t_2) \cup Dom(t_1) \subseteq A$ and $Dom(t_2) \cap Dom(t_1) = \emptyset$. The domain of action rule r is defined as $Dom(t_1) \cup Dom(t_2)$.

Now, we give an example of an action rule assuming that our information system S is represented by Table 1, a, c are stable and b, d are flexible attributes. Expressions

(a, a_2), $(b, b_1 \rightarrow b_2)$, (c, c_2), $(d, d_1 \rightarrow d_2)$ are examples of atomic actions. Expression $(b, b_1 \rightarrow b_2)$ means that the value of attribute b is changed from b_1 to b_2. Expression (c, c_2) means that the value c_2 of attribute c remains unchanged. Expression $r = [\{(a, a_2), (b, b_1 \rightarrow b_2)\} \Rightarrow \{(d, d_1 \rightarrow d_2)\}]$ is an example of an action rule. The rule says that if value a_2 remains unchanged and value b will change from b_1 to b_2, then it is expected that the value d will change its value from d_1 to d_2.

4.2 Analysis of Initial Action Rule Mining

To achieve the ultimate goal of building efficient recommender system which can provide actionable suggestions for improving a client's performance, in another word, improving its NPS efficiency rating, extracting action rules is one of the most operative methods and it has been applied to various areas like medical area and sound processing area. The first step of extracting action rules from the dataset is to complete the initialization of mining program by setting up all the variables. The process of initialization consists of selection of stable attributes, flexible attributes and decision attribute, determination of favorable state and unfavorable state in decision attribute, and definition of confidence and support for resulting rules [58] [43].

There is no argument of choosing *PromoterStatus* as the decision attribute, because it directly connects to NPS rating and it dominates the status of customers' response in the entire dataset. Among the three levels in *PromoterStatus*, *promoter* represents the highest level of customers' satisfaction, which is the most ideal state for targeting; *detractor* is the worse response from customers and it is the least likable scenario; while *passive* is a state representing customers' moderate attitude. In order

to match our ultimate goal and obtain the rules for giving the most distinct difference between the most desirable and undesirable state, the favorable and unfavorable state are set to be *promoter* and *detractor*, so the program will focus on extracting action sets that aim to improve $PromoterStatus$ from *detractor* to *promoter*. Given the fact that classification result retrieved previously from samplings is not good enough for generating action rules with high quality, confidence and support should not be set too high to meet. Based on our general experience, confidence is set to 85%. In terms of choosing stable attributes, all the features associating the general information about clients and customers can be considered, which are already listed in the first and second categories of questionnaires. However, selecting too many features is quite unnecessary and time-consuming. Since the final recommender system should be client-oriented, all the generated action rules should target specific clients, resultantly $ClientName$ is preferred in set of stable attributes. Similarly, among so many available features, the ones giving information that most interests clients and clients would like to tell apart should be selected in the set of stable features. Via consulting with our data experts, $Division$ and $SurveyType$ win this competition, as $Division$ tells the specific department involved and $SurveyType$ tells the type of service: field trips or in-shop, which matters quite a lot because the handling ways for them differ in a great number of places. By indicating the survey type for each service, clients can tell the actual practicability of certain action rules, instead of adopting them with no clue about if they are applicable.

Unlike the selection of stable attributes, choosing flexible attribute is more complex. Basically, the candidates for being flexible attributes can be divided into two groups

with regards to their types, as well as their distribution. One group consists of nominal attributes including *Opportunity*, *GoodJob*, *Dispute* and *ReadMe*, whose values contains only "*Yes*" and "*No*". The other group includes all the 9 numerical features which are just added. Usually, damaging the extraction of action rules could occur if improper features are used in flexible category, and having a glance of the distribution of these features associating with decision attribute in both groups could reveal some clues concerning which one is a better choice.

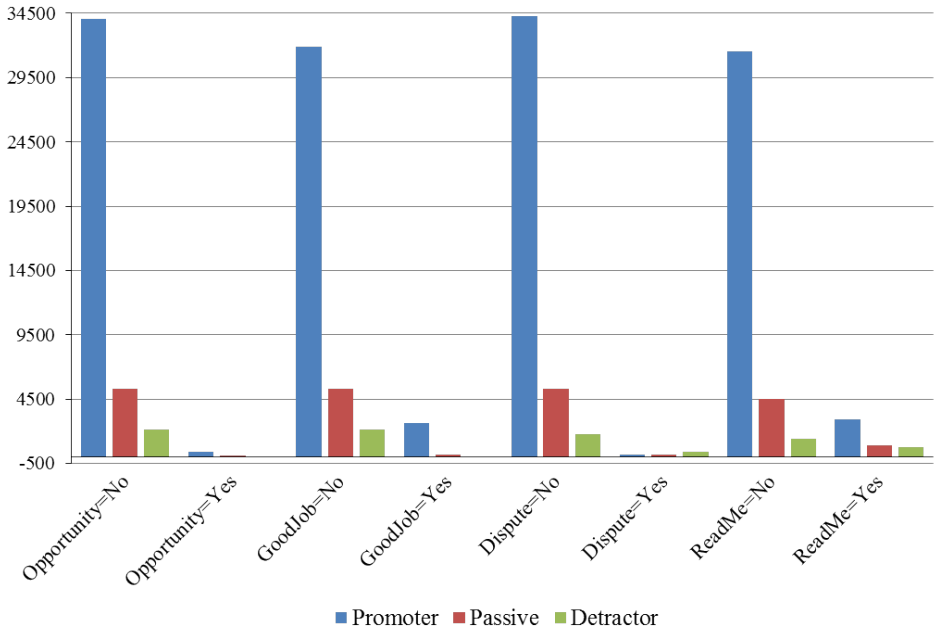


Figure 11: Distribution of decision values in nominal candidate features

The distribution of decision attribute values in nominal candidate attributes is presented in Figure 11. In the bar chart, blue, red and green bars represent *promoter*, *passive* and *detractor* in decision attribute respectively, and the length of the bars stands for the numbers of instances which are promoters, passives or detractors containing specific description ("*Yes*" or "*No*") in corresponding features. This chart is

created to show how the values of decision attribute are distributed in those nominal features which only have two values, and it is easily inspected that blue bars representing promoters stand extraordinarily high in all the features with "*No*" marked. If looking into the details for *Opportunity* solely, it was found that a great majority of instances which occupies nearly 99% in total are labeled with "*No*" while "*Yes*" barely can be seen. Among those instances with "*No*" in *Opportunity*, there are 82% of them (34094 out of 41578) that surprisingly pointed to *promoter* in decision attribute, which is telling a theory that most of the customers offering highly positive chances of promoting business actually don't think there will be a optimistic future between them and the clients. And the same scenario happens to *GoodJob* as well. However, this theory disobeys the original purpose of defining these features based on our common knowledge, *Opportunity* indeed shows customers willingness of coming back to the same client in the future if they do have needs, and "*No*" is supposed to result in the negative prospective in *PromoterStatus* since it rejects the chance of promoting clients, so is "*No*" in *GoodJob*.

For the numerical features, each of them contains numerical scores ranging from 1 to 10, and it is needless to analyze the distribution in every single score for each feature, because the general distribution of decision attribute in positive and negative responses will meet the intention. Inspired by the categorization of NPS score, the numerical values are divided into two groups: scores being less than 7 belong to negative side, and scores from 7 to 10 belong to positive side. By doing this, numerical features can be handled in a similar way as nominal features are handled, and a new chart in Figure 12 shows the distribution of decision attribute in the categorized

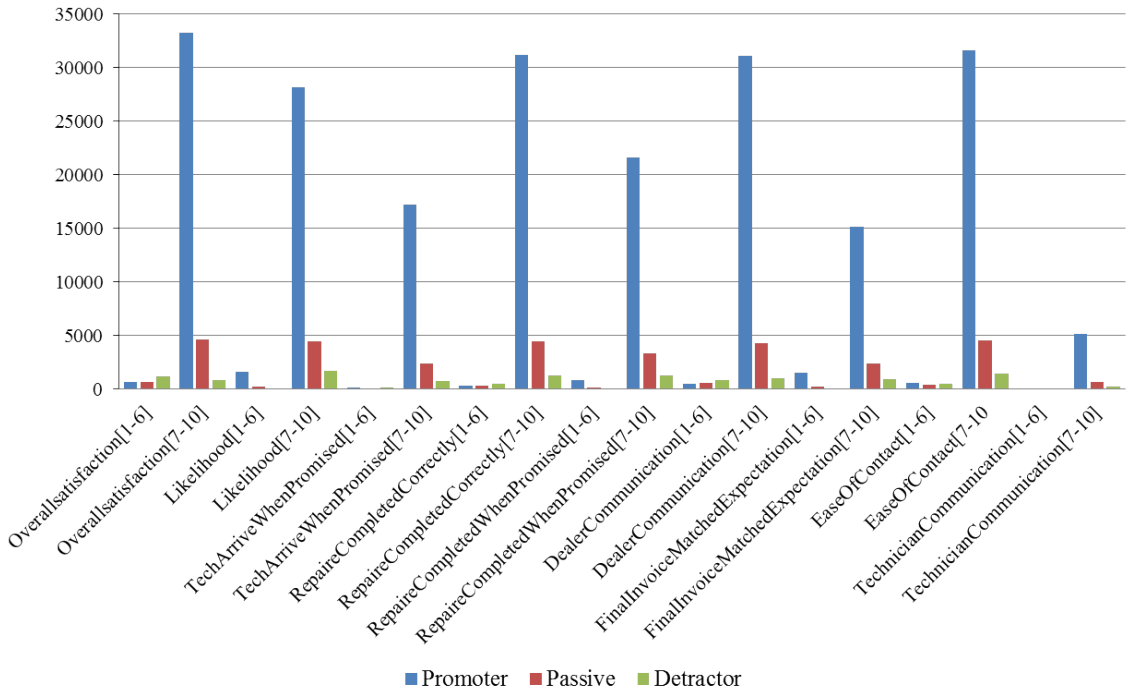
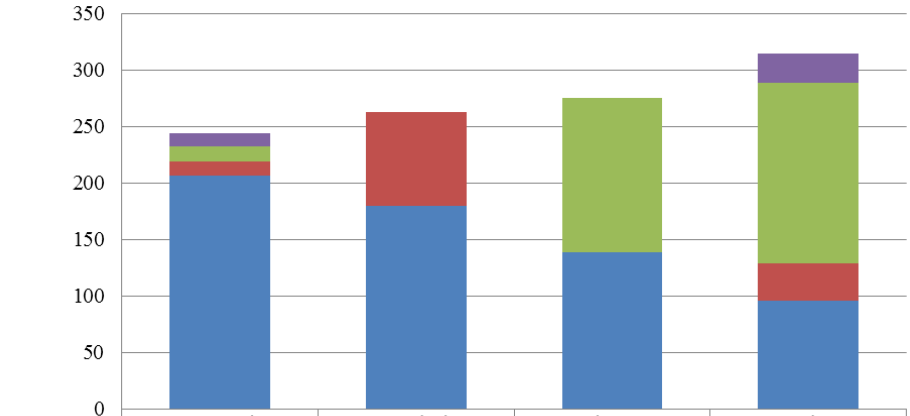


Figure 12: Distribution of decision values in numerical candidate features

numerical candidate features. It is interesting that the distribution shown in Figure 12 is totally different from that in Figure 11. In this figure, blue bars represent promoter, and red and green are passive and detractor. And the blue bars still stand extraordinarily high but this time they sit in the positive side of each feature. Take $OverallSatisfaction$ for example, statistically 93.7% of instances are counted in positive group in $OverallSatisfaction$, and 85.7% of these positives (33286 out of 33815) are matched with promoters, which is within our expectation due to the consistency in semantic meanings between decision attribute and those numerical features. Not surprisingly, the situation for all the other numerical features is precisely the same with $OverallSatisfaction$, and it completely differs with the situation of nominal features.

Regarding the opposite outcome from analyzing the distributions of the decision

attribute in two groups of features separately, intensive hypothesis concerning the contradictory in resulting action rules with selecting two groups of candidate features independently is raised and needed to be cleared. To clarify the speculation, experiments are designed and processed as choosing one group of candidate features a time as flexible attributes and comparing the resulting sets of action rules extracted with those two different settings of flexible attribute.



	Opportunity	GoodJob	Dispute	ReadMe
Yes-->Yes	12	0	0	26
Yes-->No	13	0	137	160
No-->Yes	13	83	0	33
No-->No	207	180	139	96

Figure 13: Distribution of changes of state for nominal flexible attributes in the retrieved set of action rules

Firstly, nominal features, which include *Opportunity*, *GoodJob*, *Dispute* and *ReadMe*, are set to be flexible attributes along with other determined settings mentioned previously, and then program of mining action rules is used to process the dataset. When doing the analysis on resulting set of action rules, more attention is put on flexible attributes where the purpose of experiments is derived from, so we mainly focus on changing status of flexible attributes which includes $(No \rightarrow No)$, $(No \rightarrow Yes)$, $Yes \rightarrow No)$ and $(Yes \rightarrow Yes)$. In the resulting set, 448 distinct action rules are

generated, and all the flexible attributes are associated with over half of the rules. To determine which changes in flexible attributes dominate the whole situation, we look into the appearances of all the possible diversifications for each feature. In Figure 13, the distribution of changing states for each flexible attribute is presented, the vertical axis shows the number of rules that are associated with the corresponding state transformation of certain attributes, and the bars with different color indicate different state transformation listed in the table on the bottom which also contains exact number of rules involving each transformation in each feature. As the figure shows, over 300 action rules are found associated with $ReadMe$, and by looking into the distribution of state transformations in *Opportunity* and $GoodJob$, we can discover that the blue bars play a dominating role since 207 out of 245 (84.5%) are blue in *Opportunity* while 180 out of 263 (68.4%) are blue in $GoodJob$. Generally speaking, blue bar stands for the transformation of state ($No \rightarrow No$), which implies the state of these two features should stay at negative level if we want to improve the client's performance. Thinking more profoundly, this implication from our analysis is not consistent with what is commonly known, or is not helpful for providing actionable suggestions at the end, the only explanation left is the influence of these features seems to be too trivial to affect a client's performance since the performance can get improved without changing the states. Therefore, we can conclude that such attributes are not suitable to be considered as flexible attribute. However, the case involving *Opportunity* is more complicated, as "Yes" in *Opportunity* indicates the chance that the customer come back for service is high due to unsatisfactory service last time and future work in need, while "No" doesn't completely show negativeness.

So it should be handled accordingly.

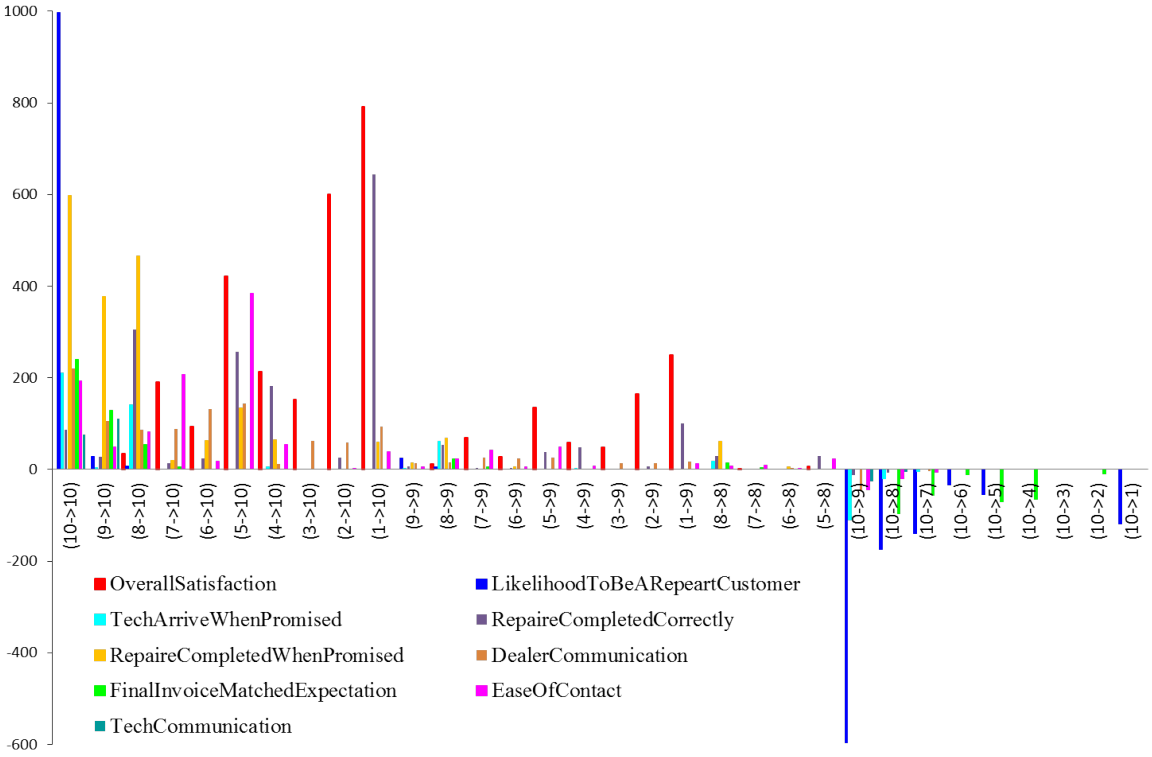


Figure 14: Distribution of changes of state for numerical flexible attributes in the retrieved set of action rules

Meanwhile, other experiments on mining action rules with numerical features selected as flexible attributes are also completed. But this time, numerical values are processed as how they originally are without transforming into different clusters, the reason is the changes in state of numerical features could be very slight, the distinct transformation could barely be inspected with scores clusters into more general groups. Compared to the limited number of possible state transformation for nominal features, there are much more possible state transformation in the case of numerical features. In our case, all the state transformation of flexible attributes appearing in the retrieved set of action rules are collected and shown in Figure 14. The general

idea of this figure is similar to Figure 13, all the appeared transformation for numeric features are listed in horizontal axis and the bars in different colors represent features as introduced in the left bottom legend. The distinguishing characteristic in Figure 14 is those negative number of rules containing certain action sets are listed on the right side of the chart. Observing these state transformation on the right side of horizontal axis starting with $(10 \rightarrow 9)$, the final state of those changes is smaller than the initial state, which makes the result be negative if using the final state minus initial state, so such changes can be referred as "negative" transformation. As a result, the number of rules containing negative transformation for each flexible feature is marked as negative value, which causes that the bars are listed on the bottom of horizontal axis. In the figure, more bars in different colors standing in one position implies the corresponding transformation in that position is related to more features than others, so clearly transformations $(10 \rightarrow 10)$, $(9 \rightarrow 10)$ and $(8 \rightarrow 10)$ are the most popular ones among all the features. At the same time, the blue bar at the position of $(10 \rightarrow 10)$ is the highest in the chart, which indicates this transaction is the most significant for $LikelihoodTobeARepeatCustomer$, as well as the red bar in $(1 \rightarrow 10)$ which shows the remarkable importance of this transformation for $OverallSatisfaction$. However, if carefully comparing the positions of blue and red bars which stand for $LikelihoodTobeARepeatCustomer$ and $OverallSatisfaction$ respectively, we can see that their situation completely differ from each other, all the red bars are located at the positive area while majority of blue bars are located at the opposite side. Understandably that transformation standing at positive area is positively correlated with our goal of improving detractors to promoters, so they are truly useful for building

recommender system. However, the interpretation of negative changes is similar to the case occurring to *Opportunity* and *GoodJob*, they are implying damaging the performance in certain aspects somehow could achieve the improvement in final result, which defies our common understanding. So we can comprehend this fact as those features associating to majority of negative changes, like *LikelihoodTobeARepeatCustomer* and *FinalInvoiceMatchedExpectation* in our analysis, rarely influence the goal of achieving a more desirable state and they are not appropriate for being flexible attributes in action rule mining.

Until now, we have a clear idea about how to prepare the settings for processing action rule extraction, moreover, some features that seem to have potential but actually fail in providing desirably helpful information are excluded out of our list. Overall, the basic preparation for further work has been done and the key parts - construction of hierarchical dendrogram and design an agglomerative algorithm are going to be illustrated thoroughly in the following chapters.

CHAPTER 5: CLUSTERING CLIENTS SEMANTICALLY

5.1 Analysis of Individual Clients

The hierarchically structured recommender system will be client oriented in the end, and it should be familiar with detailed information of target clients, such as how this client performs in serving customers, how consistent its customers' responses are, its neighboring competitors and even their performance on services. Considering all the valuable information, digging deeper in individual clients is inevitable and worth doing. It has been introduced that NPS rating is used as the key measure for evaluating a client's performance and it is calculated as:

$$NPS[i] = \frac{Num[i, Promoter]}{Num[i, *]} - \frac{Num[i, Detractor]}{Num[i, *]} \tag{2}$$

In our following analysis, clients' name will be replaced by numbers based on their alphabetical order, rather than using exact actual names due to the confidentiality. So i in Equation (2) is referring to a certain client who is labeled with this number. By $Num[i, Promoter]$, we mean the number of $Promoter$ instances in the dataset of Client i. Similarly, by $Num[i, Detractor]$, we mean the number of $Detractor$ instances in the dataset of Client i. And $Num[i, *]$ indicates the total number of instances in dataset of Client i regardless of the class categories. Then the overall NPS rating for our entire dataset should be calculated as: $NPS[*] = \frac{Num[*, Promoter]}{Num[*, *]} - \frac{Num[*, Detractor]}{Num[*, *]}$, and the result is 0.7613, which is from percentage of promoters 81.24% (34523 out of

42493) minus percentage of detractors 5.11% (2172 out of 42493).

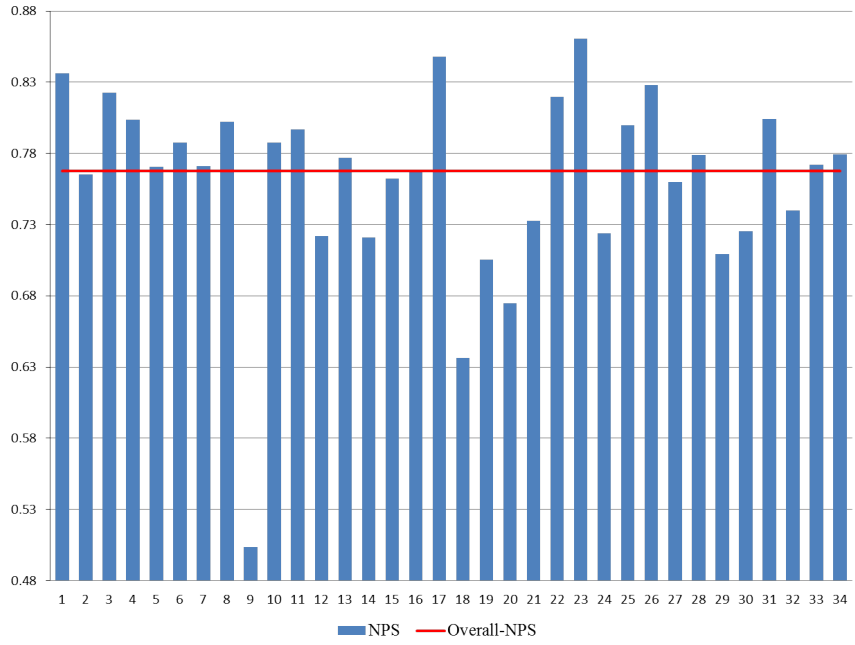


Figure 15: NPS rating of individual clients comparing with overall NPS rating

Apparently, overall NPS rating cannot represent the status of NPS rating for each client, there could be clients having better or worse NPS ratings. The Figure 15 shows the NPS ratings for each client which is computed based on Equation (2), as well as the comparison between overall NPS rating and individual NPS rating. In the figure, blue bars are the NPS ratings corresponding to each client labeled by numbers at horizontal axis, higher the blue bar is, greater NPS rating the matched client has; and red horizontal line presents the overall NPS rating. We can observe that there are 20 clients whose NPS rating is greater than the overall rating and three of them are even over 0.8, while there is one client, Client 9, whose NPS rating is exceptionally low, barely passes 0.5. Even if NPS rating is fairly good, there is still some space for improvement.

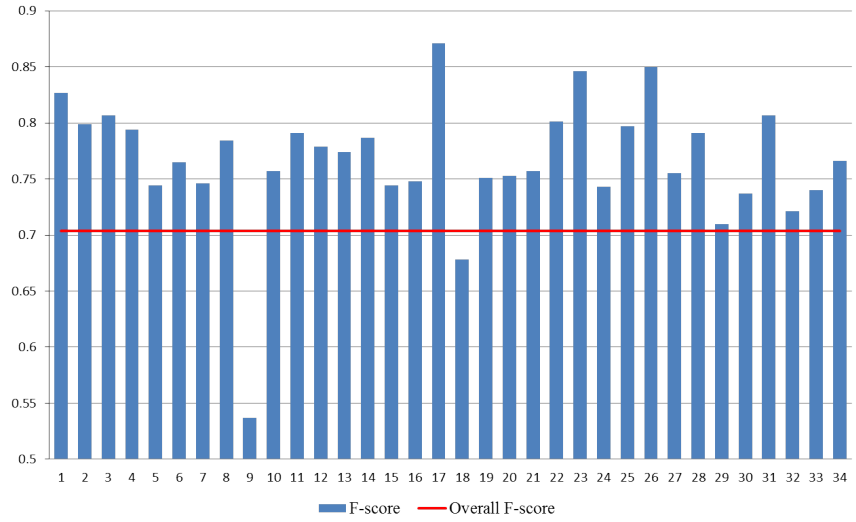


Figure 16: F-score of individual clients comparing with overall F-score

Many efforts have been taken in the third chapter for improving the overall classification result, namely overall F-score and the overall F-score achieved in the end of third section is 0.7038 which represents the general situation in our dataset. At the same time, the curiosity on comparing F-score of individual clients with overall F-score drives us to perform classification tests with $J48$ in WEKA to datasets of every single client. Then Figure 16 is created to present our comparison result. Similar as Figure 15, blue bars stand for the F-score of each corresponding client listed at the bottom and red horizontal line shows the overall F-score. F-score can tell us the general quality of a dataset in a way that examines the consistency of knowledge hidden in the dataset. In Figure 16, it is found that most of the clients have better F-score than overall F-score, except Client 18 and Client 9 again, which suggest that the knowledge in most of individual datasets is more crucial and worthy, and their quality could benefit more than combining all of them together. So it helps us confirm that the idea of building personalized recommender system for every single client is

indeed better than building a comprehensive one.

5.2 Introduction of Semantic Similarity

It is believed that clients can collaborate with each other by exchanging knowledge hidden in datasets and they can benefit from others whose hidden knowledge is similar, so it is necessary to define a distance measure that estimates the difference of knowledge concerning $Promoter$, $Passive$ and $Detractor$ between clients. The following definition of semantic similarity is proposed to fulfil this purpose.

Semantic similarity is firstly mentioned in [28]. In this section, we introduce the notion of semantic similarity between clients. Assume now that $RC[1]$, $RC[2]$ are the sets of classification rules extracted from the datasets collected for clients $C1$, $C2$. Also, we assume that

$RC[1] = RC[1, Promoter] \cup RC[1, Passive] \cup RC[1, Detractor]$,

where $RC[1, Promoter] = \{r[1, Promoter, i] : i \in I_{Pr}\}$, $RC[1, Passive] = \{r[1, Passive, i] : i \in I_{Ps}\}$, $RC[1, Detractor] = \{r[1, Detractor, i] : i \in I_{Dr}\}$, where $\{r[1, Promoter, i] : i \in I_{Pr}\}$ is a collection of classification rules defining "Promoter", $\{r[1, Passive, i] : i \in I_{Ps}\}$ is a collection of classification rules defining "Passive", and $\{r[1, Detractor, i] : i \in I_{Dr}\}$ is a collection of classification rules defining "Detractor".

In a similar way, we define

$RC[2] = RC[2, Promoter] \cup RC[2, Passive] \cup RC[2, Detractor]$,

where $RC[2, Promoter] = \{r[2, Promoter, i] : i \in J_{Pr}\}$, $RC[2, Passive] = \{r[2, Passive, i] : i \in J_{Ps}\}$, $RC[2, Detractor] = \{r[2, Detractor, i] : i \in J_{Dr}\}$.

By $C1[1, Promoter, i]$, $C1[1, Passive, i]$, $C1[1, Detractor, i]$ we mean confidence of

$r[1, Promoter, i]$, $r[1, Passive, i]$, and $r[1, Detractor, i]$ in a dataset for client $C1$, respectively.

By $C2[1, Promoter, i]$, $C2[1, Passive, i]$, $C2[1, Detractor, i]$ we mean confidence of $r[1, Promoter, i]$, $r[1, Passive, i]$, and $r[1, Detractor, i]$ in a dataset for client $C2$, respectively.

By $C2[2, Promoter, i]$, $C2[2, Passive, i]$, $C2[2, Detractor, i]$ we mean confidence of $r[2, Promoter, i]$, $r[2, Passive, i]$, and $r[2, Detractor, i]$ in a dataset for client $C2$, respectively.

By $C1[2, Promoter, i]$, $C1[2, Passive, i]$, $C1[2, Detractor, i]$ we mean confidence of $r[2, Promoter, i]$, $r[2, Passive, i]$, and $r[2, Detractor, i]$ in a dataset for client $C1$, respectively.

Then, we can define the concept of semantic similarity between clients $C1$, $C2$ denoted by $SemSim(C1, C2)$ in Equation(3).

$$\begin{aligned} SemSim(C1, C2) = & \frac{\Sigma\{|C1[1, Promoter, k] - C2[1, Promoter, k]| : k \in I_{Pr}\}}{card(I_{Pr})} \\ & + \frac{\Sigma\{|C1[1, Passive, k] - C2[1, Passive, k]| : k \in I_{Ps}\}}{card(I_{Ps})} \\ & + \frac{\Sigma\{|C1[1, Detractor, k] - C2[1, Detractor, k]| : k \in I_{Dr}\}}{card(I_{Dr})} \\ & + \frac{\Sigma\{|C2[2, Promoter, k] - C1[2, Promoter, k]| : k \in J_{Pr}\}}{card(J_{Pr})} \\ & + \frac{\Sigma\{|C2[2, Passive, k] - C1[2, Passive, k]| : k \in J_{Ps}\}}{card(J_{Ps})} \\ & + \frac{\Sigma\{|C2[2, Detractor, k] - C1[2, Detractor, k]| : k \in J_{Dr}\}}{card(J_{Dr})}. \end{aligned} \tag{3}$$

5.3 Construction of Semantic Similarity-Based Hierarchical Dendrogram

Given the definition of semantic similarity, the distance between any pairs of clients are quantified in a semantic way and smaller the distance is, more similar the clients are. Then a semantic similarity-based distance matrix is built above the definition. With the distance matrix, a hierarchical clustering structure(dendrogram) is generated by applying an agglomerative clustering algorithm and the dendrogram is shown in Figure 17.

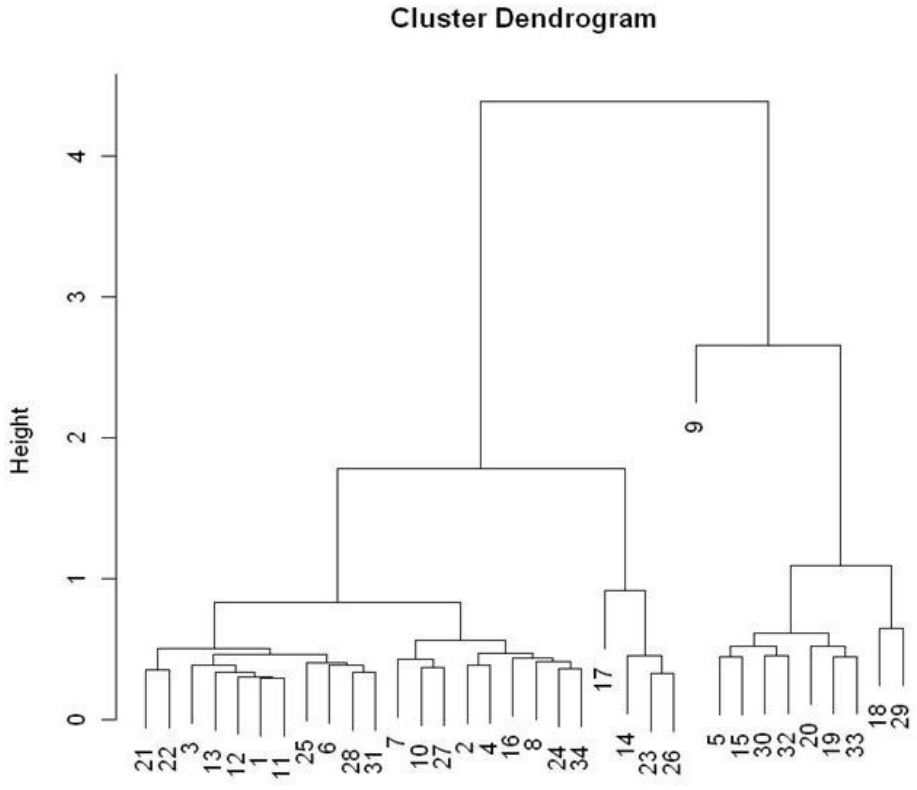


Figure 17: Hierarchical clustering of 34 clients based on semantic similarity

Figure 17 shows the hierarchical clustering of 34 clients with respect to their semantic similarity calculated by the given Equation(3). With the dendrogram, we can easily find out the groups of clients which are relatively closer to each other in seman-

tic similarity. If describing it using tree structure based terminology, then every leaf node in the dendrogram represents the corresponding client as the number shown, and the depth of one node is the length of the path from it to the root, so less difference of the depth between two leaf nodes, more semantically similar they are to each other. From this hierarchically structured dendrogram, it is easily discovered that Client 9 is not near any other clients in terms of semantic similarity since Node 9 is hung in there in a very high level and has no siblings who are leaf nodes, which makes the minimum difference of the depth between Node 9 and its most similar node at least 2. While all the other nodes have options with whom the depth difference is 0 or 1, so this makes Client 9 special again regarding its poor NPS rating and F-score. Meanwhile, it is interesting to compare the relationship between clients from semantic similarity angle or from physical distance angle and it turns out the conclusion is just under our prediction. The comparison results differ from clients to clients depending on specific circumstances. On one hand, the correlation between clients based on semantic similarity could be consistent with the differentiation in their geographical locations. For example, Client 28, Client 31 and Client 6 are semantically close to each other in the dendrogram, at the same time, they are actual neighbors in the geographical map. On the other hand, it is found in several groups of clients that the semantic distance between them is not necessarily positive correlation with the geographical distance between them. For example, Client 24 and Client 34 are children nodes of same parent in the dendrogram shown in Figure 17, it means that the knowledge hidden in the datasets of Client 24 and Client 34 are similar to some extent and they are semantically close to each other. However, Client 24 is known to be locating in

the west of US while Client 34 is the one at Georgia, from physical point of view, they are extremely far away from each other and this is the most representative example in our dataset. The inconsistency between semantic and geographical distance inspires us to propose the following methodologies for building our recommender system in a more appropriate way.

CHAPTER 6: HIERARCHICAL AGGLOMERATIVE METHOD FOR IMPROVING NPS

6.1 Background

Now we firstly explain the background information concerning Hierarchical Agglomerative Method for Improving NPS (HAMIS). Broadly speaking, HAMIS is able to maximally enlarge the dataset of a specified client by following a bottom-up path in an existing hierarchically structured dendrogram with respect to semantic similarity. In the dendrogram, every leaf node represents a dataset of a corresponding client and every parent node represents the merged dataset of its mergeable children. Therefore, higher the bottom-up path ends in the dendrogram, larger the resulting merged dataset is potentially, namely, more generalized dataset is returned by HAMIS. The bottom-up path formed during the process links all the successfully merged nodes. Mergeable node means the node that can be used for merging and it is identified by the following criteria:

- It is the most semantically similar node in current situation.
- Its NPS rating is not less than the targeted client.

Given the definition of semantic similarity in the second section of this thesis, we are capable to quantify the concept of how similar customers from different clients feel about the provided service. Therefore, if the semantic distance between two clients

is relatively small, in other words, these clients are semantically close, then we could infer that the customers from these clients think of *Promoter*, *Passive* and *Detractor* in a more similar way, comparing to customers from clients that are further away regarding semantic distance. So it is possible that action rules extracted from the dataset covering all these semantically similar clients are useful for further improving the NPS rating of individual client. Based on the semantic distance retrieved, we clustered all the 34 clients using the agglomerative clustering algorithm and generated a dendrogram as shown in Figure 17 which provides us with very efficient way to identify the most similar clients. As mentioned previously, each leaf node of the dendrogram stands for each client correspondingly, so the nodes that are semantically closest should be all the leaf nodes on the sibling side. For instance, if the sibling node is a leaf node, then there is only one node available for being the closest; while the sibling node is a parent node, it complicates the situation since the union set of all the leaf nodes under this sibling node should be the most semantically similar, then certainly, all the leaf nodes on the sibling side should be counted in and be checked one by one in a top down sequence following the depth of these nodes.

However, merging a targeted client with a semantically similar client whose NPS rating is lower won't fully match our expectation, since our goal is to improve the target's NPS rating, not conversely. But what we can be certain of is that, by merging a client with other client whose NPS rating is not lower, we get a dataset with higher or at least the same NPS rating. Let's assume that $NPS[i]$ and $NPS[j]$ are NPS ratings of two clients i and j, and both of them can be computed based on Equation (2).

Also we assume $NPS[j] \geq NPS[i]$ and $NPS[i \cup j]$ is the NPS rating of the union set of client i and j, so we can expect $NPS[i \cup j] \geq NPS[i]$, because

if $NPS[j] - NPS[i] =$

$(\frac{Num[j,Promoter]}{Num[j,*]} - \frac{Num[j,Detractor]}{Num[j,*]}) - (\frac{Num[i,Promoter]}{Num[i,*]} - \frac{Num[i,Detractor]}{Num[i,*]}) \geq 0,$

then $NPS[i \cup j] - NPS[i] =$

$(\frac{Num[j,Promoter]+Num[i,Promoter]}{Num[j,*]+Num[i,*]} - \frac{Num[j,Detractor]+Num[i,Detractor]}{Num[j,*]+Num[i,*]}) -$

$(\frac{Num[i,Promoter]}{Num[i,*]} - \frac{Num[i,Detractor]}{Num[i,*]}) \geq 0.$

Thus, we can surely get a joined dataset with non-decreased NPS rating.

In addition, continually keeping tracking the quality of classifiers resulting from the merged dataset during the entire procedure is advantaging, then we will achieve the best performance of generalization. Classification results show the quality of datasets for mining action rules and poor quality classifiers lead to poor confidence of action rules. Accordingly, we must make sure that the classifiers are under improvement. To evaluate the classifiers, we use F-score that includes both accuracy and coverage of classification into consideration. As a popular measure of assessing the classification performance, F-score offers us a comprehensive and accurate view on our data.

Therefore, the three criteria mentioned above make the foundation of algorithm HAMIS and the procedure of HAMIS is stated thoroughly in next section.

6.2 Presentation of HAMIS

Technically speaking, the purpose of the algorithm HAMIS is to keep expanding the targeted client by unionizing it with all the clients satisfying the conditions. Unless the resulting dataset for chosen client can't be expanded any further, the algorithm

would be repeatedly executed. And the algorithm returns resulting dataset when it ends. As HAMIS is built on the basis of a hierarchical dendrogram regarding semantic distance, its procedure will be described using tree structure related terminology. The algorithm is designed as presented in Algorithm 1.

Algorithm 1 Hierarchical Agglomerative Method for Improving NPS

Input: N_{target}: the target node.
Output: N: the node processed from N_{target}.

```
N ← N_target
repeat
  N_0 ← N;
  retrieve N_c which is a list of candidates N_c[1], N_c[2], ..., N_c[n];
  while N_c ≠ ∅ do
    get next available candidate N_c[i] ∈ N_c (i ∈ {1, 2, ..., n});
    if NPS[N_c[i]] ≥ NPS[N_target] then
      N_m ← {N_c[i]} ∪ N
      if Fs[N_m] ≥ Fs[N] then
        N ← N_m
      end if
    end if
    N_c ← N_c \ {N_c[i]}
  end while
  if N_0! = N then
    N climbs to its parent node in upper level;
  end if
until N_0 = N
return N
```

In the procedure HAMIS, the resulting node is defined as N and it is initialized with the input targeted node N_{target}. Once N has been given, the nodes that are semantically closest to it are retrieved and stored in a list naming N_c. Accordingly, N_c contains all the leaf nodes on the sibling side of current N in the dendrogram and they are the candidates for being mergeable with N. It is apparent that at least one candidate is required to proceed, otherwise, it means the node N has reached the root and there are no more nodes available for merging. When proceeding, the following

part is the main part in HAMIS and it iterates through all the candidates in N_c on the foundation of other two merging criteria mentioned above: NPS rating and F-score. If a candidate $N_c[i]$ is with not less NPS rating than the targeted node N_{target}, then the candidate is qualified for merging. And the merged result is temporarily stored as N_m. N_m can't become the new resulting node N yet unless its F-score is greater or at least equal to F-score of current N. Thus, if the resulting node N is replaced by the merged result N_m, it suggests the merging process for current candidate succeeds and the new N will be used for next candidate in N_c if there are still any. When a candidate fails merging with N, the same resulting node N will be used for another generalization attempt with next available candidate. The main part will not end until all the candidates have been checked. If there are more than one candidate found in N_c, they will be checked in a top down order based on their depth in the dendrogram, smaller the depth of a candidate is, earlier the candidate will be checked. So candidates are stored ascendingly with regards to their depth in dendrogram, saying that for each candidate $N_c[i](i \in \{1, 2, ..., n\})$, $depth[N_c[i+1]] > depth[N_c[i]]$, where $depth[N_c[i]]$ is the depth of node $N_c[i]$ in dendrogram and $i \in \{1, 2, ..., n-1\}$.

Each candidate is examined in almost the same way, while the only difference would be a new resulting node iteratively generated by successful merging process. Evey time a new node is merged with the resulting node, the newly updated resulting node will be taken to the following parts.

When the main part is completely finished, the resulting node could be updated with some nodes added in. If this is the case, it implies the resulting node has been generalized further at current depth, so the resulting node will climb up one level in

the dendrogram and become the parent node of previous one. With a new resulting node at a new depth, HAMIS will continually get all of its candidates and repeat the main part once more. HAMIS will keep going until the resulting node at certain level is not changed at all after main part ends or it has reached the root.

6.3 Experiments on HAMIS with Semantic Similarity-Based Dendrogram

To show the running process of algorithm HAMIS in our domain, we are going to take Client 2 as a target for example, and the relevant data used during this procedure is shown in Table 2. As the semantic similarity based clustering dendrogram has been given in Figure 17 and the part of it relating to our example is shown in Figure 18, we observe that node $\{2\}$ representing Client 2 is labeled in green at the bottom and it is the initial resulting node. As a sibling node to node $\{2\}$, node $\{4\}$ is the only candidate in N_c which is most semantically similar to node $\{2\}$, and NPS rating of node $\{4\}$ shown in Table 2 is higher. In addition, F-score calculated by J48 in WEKA for the merged node $\{2, 4\}$ is also higher than current resulting node $\{2\}$, which are 0.788 comparing to 0.783, hence the merged node $\{2, 4\}$ successfully replaces $\{2\}$ and becomes the new resulting node. Meanwhile, there is no more unchecked candidates in N_c, so HAMIS is done with current depth and will continue with the new resulting node by climbing up to the parent node which is labeled in blue. At a new position, because the sibling node of the current resulting node is not a leaf node, leaf nodes $\{16\}, \{8\}, \{24\}$ and $\{34\}$ on the sibling side should be included in candidate set as we defined, and they are labeled in blue as well. According to the depth of each candidate in dendrogram, they will be checked following the top down sequence which is node

Table 2: NPS rating and F-score of relevant nodes in Figure 18

	$N\{2\}$	$N\{4\}$	$N\{16\}$	$N\{8\}$	$N\{24\}$	$N\{34\}$
NPS rating	0.765	0.803	0.767	0.802	0.724	0.779
	$N\{2\}$	$N\{2,4\}$	$N\{2,4,16\}$	$N\{2,4,8\}$	$N\{2,4,24\}$	$N\{2,4,34\}$
F-score	0.783	0.788	0.776	0.786	NA	0.778

{16} first, then node {8} and {24}, and {34} at the end. Then HAMIS attempts to merge these candidates to resulting node individually, but it turns out none of them can successfully merge with $\{2, 4\}$.

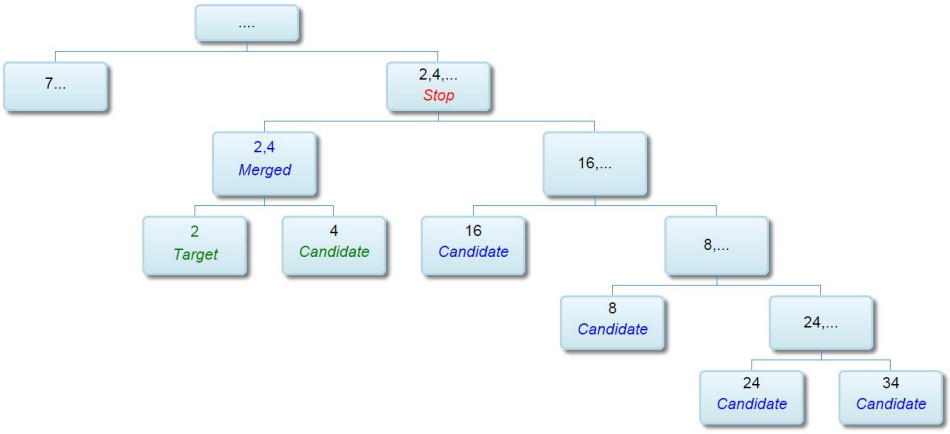


Figure 18: Example of running HAMIS on Client 2 with semantic dendrogram

When it comes to node {16}, although its NPS rating is just a little bit higher than the targeted node {2}, the F-score of $\{2, 4, 16\}$ is much lower that $\{2, 4\}$, so the merging of {16} and $\{2, 4\}$ fails and the main part goes to the next one, which is node {8}. The case for node {8} is exactly the same as for the node {16}, so $\{2, 4\}$ is still the resulting node without being changed and it keeps going to node {24} and {34}. But neither of them can join $\{2, 4\}$ due to either low NPS ratings or lower F-score of joined nodes. Consequently, node $\{2, 4\}$ has not been replaced with any new merged node after all the candidates have been checked, which suggests $\{2, 4\}$ is the most

generalized in our program for Client 2. Thus, HAMIS ends here and returns $\{2, 4\}$.

Next step, we are going to generate action rules for both generalized dataset and original dataset of Client 2. Before the program starts, we need to specify the necessary attributes. Certainly that promoter status should be the decision attributes and the transitions we are interested in are from *Detractor* to *Promoter*. The customers' personal information related attributes should be seen as stable attributes, in our experiment, attributes like customers' name, location and contact number are set as stable attributes. Then the attributes about customers' feeling and comment are selected as flexible attributes, and these are the keys for improving NPS ratings since they tell us about the actions we should adopt. For example, attributes evaluating if "the job is done correctly" and "the time frame of technician's arrival" are flexible attributes. Based on our personal knowledge about the dataset, we expect that a huge number of action rules will be generated and we only pay attention to the one with sufficiently high confidence, so we intend to get the action rules with at least 80% confidence.

Figure 19 shows the results of comparing action rules extracted from dataset $\{2, 4\}$ to dataset $\{2\}$ alone. In the figure, blue bars display the number of exact same rules with same support and confidence extracted from both datasets, red bar represents the initial rules extracted from dataset $\{2\}$ which are not found exactly the same as the ones extracted from dataset $\{2, 4\}$ but the action sets of these rules are contained in the action rules from $\{2, 4\}$ with higher confidence or support, which is marked using orange bar on the bottom. Last but not the least, green bar and pink bar show the unique rules in both action rule sets respectively that don't exist in the

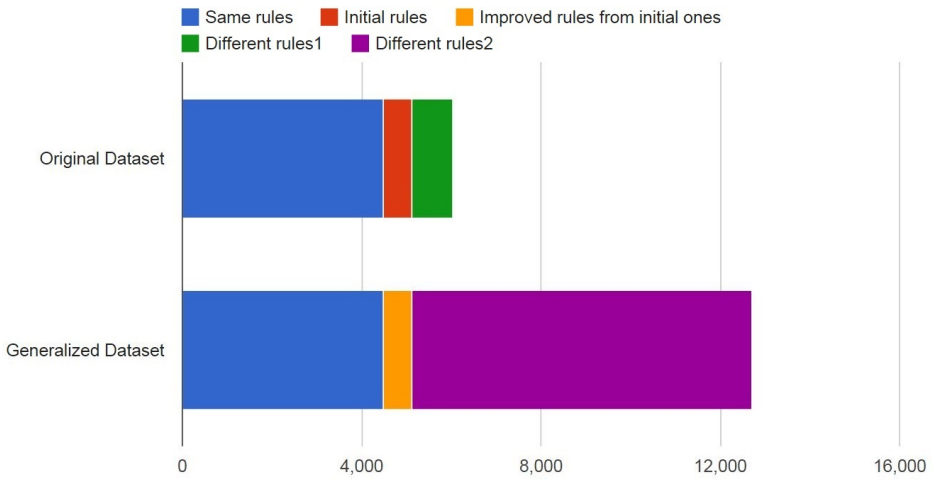


Figure 19: Comparison of sets of action rules generated from dataset of Client 2 before/after HAMIS

other action rule set. Firstly, we can easily see that there are twice as many as rules generated from the expanded dataset. More specifically, we found $12,715$ action rules from the larger dataset while $6,026$ from the original dataset. At the same time, nearly 75% of action rules from dataset of Client 2 can be found in the set of action rules from the more generalized dataset $\{2,4\}$ with same support and confidence. Over 10% of action rules found in original dataset can be found in the set of action rules from generalized dataset with higher support or confidence. Furthermore, a lot of new action rules have been discovered and over 70% of the new action rules have remarkably high confidence.

In order to get more convincing results, we apply HAMIS to all 34 clients individually and retrieve the generalized datasets. From the results in Figure 20, we get 18 out of 34 clients who are generalized by HAMIS, and in average, the generalized dataset for each client is three times as large as the original dataset. The largest generalized dataset is from Client 7 which is far more larger than other expanded datasets. For

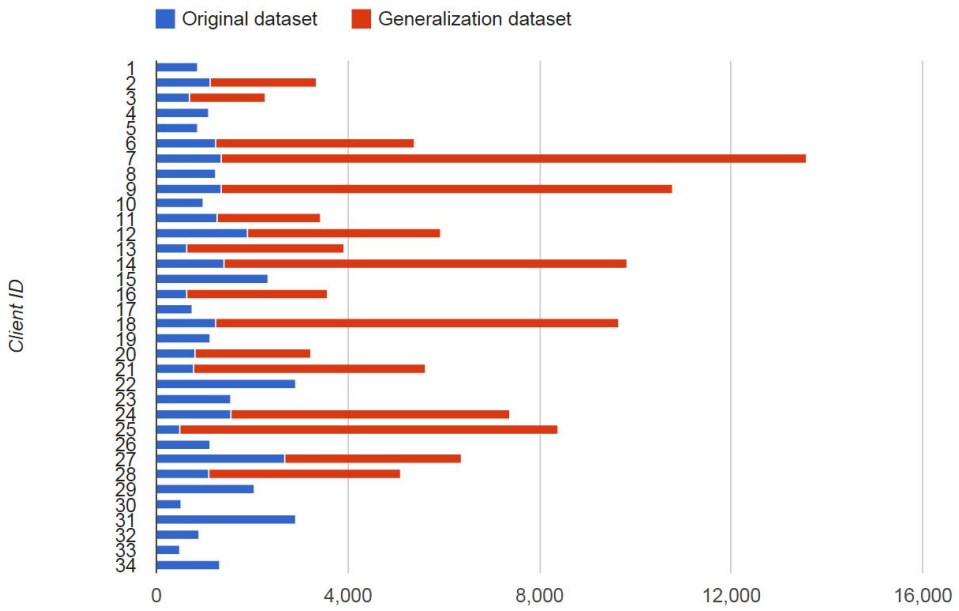


Figure 20: Performance of HAMIS on 34 clients with semantic dendrogram

comparing action rules, the results vary with different generalized datasets associated with clients, and the set of action rules generated from those generalized datasets of clients is at least two times larger than that from single client alone, which is still within our expectation.

CHAPTER 7: FURTHER EXPANSION WITH GEOGRAPHICAL DISTANCE

In Section 5, the NPS rating of each client is computed and shown in Figure 15, and we are provided with the locations of each client, so 34 clients' approximate locations along with their NPS ratings are collected and shown in Figure 21.

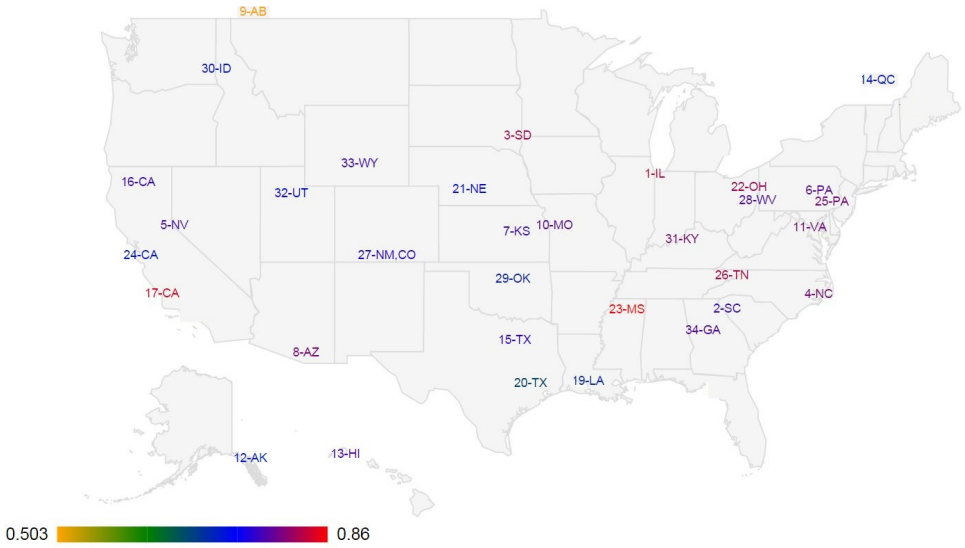


Figure 21: Approximate locations of 34 clients and their NPS ratings

This figure is a map of U.S. with the regions that clients cover and their NPS ratings. On the map, the text shows client's ID number and the state that client locates, and the color of texts displays clients' NPS rating. From the color axis on the bottom, the minimum NPS rating in our domain is 0.503 while the maximum is 0.86, and all the text markers are labeled by color in the order of yellow, green, blue and red which indicates NPS rating in an ascending sequence. So Client 23 in Mississippi has the highest NPS rating and Client 9 in Alberta of Canada has the

lowest. Also we can find NPS rating of clients in Canada region relatively lower than the ones in US. After discussing with our data experts, we have been informed that it possibly results from high requirement or different thoughts on service from customers in Canada, or just simply because really bad service was provided.

The fact that clients can collaborate with each other by exchanging their knowledge hidden in their datasets is our foundation for designing HAMIS, but we still believe that clients can get helpful recommendations not only from the clients whose knowledge hidden in dataset is similar, but also from those clients who target the same groups of customers. Thus, we assume that clients locating nearby actually may target same customers, at least groups of customers who share common thoughts about how they think of the ideal service and how they are served in real world, because they are sharing similar neighborhood, similar regional culture and similar environment, which could affect the way clients deal with customers to some extent. From the experiments of applying HAMIS with semantic similarity based clustering dendrogram to all the clients, it turns out that there are still some clients who can't benefit from HAMIS due to the failure of generalization. 90% of them are caused by the lower NPS rating of their most semantically similar clients. For example, as one of the clients who failed in generalization, Client 5 can't merge with Client 15 because the NPS of Client 15 is 0.762, which is slightly lower than the NPS of Client 5. However, Client 5's geographical neighbor Client 17 has been expanded with Client 23. We can't help thinking that the knowledge hidden in dataset of Client 5 could be similar to the knowledge hidden in Client 17 to a certain degree. Therefore, it is an interesting idea to think that Client 5 can benefit with the advices from Client 17

and even from Client 23.

Table 3: Classification results from expanded Client 24 with different clients

Clients	**Location**	**Precision**	**Coverage**
Client 24	CA	80.86%	99.03%
Client 34	GA	83.41%	96.95%
Client 16	CA	82.68%	95.42%
Client 24&34	N/A	81.58%	95.74%
Client 24&16	CA	82.17%	99.44%

What's more, the case that clients are semantically similar is not absolutely equivalent to the case that they are geographically close to each other, sometimes, they could be even far away from each other. Hence, it is not absolutely true that merging with the most semantically similar clients is the best option for us. The most representative example mentioned previously is Client 24 and Client 34. From figure 21, we can see that Client 24 is in California while Client 34 stays at Georgia, they are physically far away, but they are treated as most semantically similar nodes as it is shown in Figure 17. Although Client 24 can successfully merge with Client 34, it can also merge with its neighbor Client 16. As the figures show in Table 3, both precision and coverage of classifiers on union set 24, 34 is lower than the union set 24, 16, which is 82.17% and 99.44% comparing to 81.58% and 95.74%. Another words, F-score of classifiers on 24, 34 is lower than F-score of classifiers on 24, 16, so it seems that merging with Client 16 maybe a better choice for Client 24 than merging with Client 34. Following the same idea, Client 34 can't be generalized with Client 24 because of its lower NPS rating in previous experiments. If we want to help Client 34, we need to offer other options like the clients nearby Client 34.

Additionally, through looking into the neighboring clients around Client 34, we

discover another interesting phenomenon, for Client 2, although it has been generalized using semantic similarity, it is surrounded by several clients with higher NPS ratings, which implies probably customers living in this area are more satisfied with other clients instead of Client 2. Making a living in such competitive environment, customers around here could have stricter requirement to clients and be harder to satisfy, so if Client 2 doesn't know how to serve its customers' in a better way, its situation could get worse and worse. Concerning all the cases above, we believe that geographical distance could be very helpful for some clients in certain circumstances and taking geographical distance between clients into our consideration for improving the hierarchical structure is a promising movement.

7.1 Definition of Geographical Distance Between Clients

To build a geographical distance based hierarchical dendrogram, defining the distance between two clients is our first step. Besides knowing that each client owns a few shops, we consulted with our data consultants to get the latest location of shops for each client. Finally, a list including all the shops for each client is confirmed and it is used for the estimation of the distribution of shops owned by each client. Figure 22 shows the distribution of shops crossing the entire U.S. as well as parts of Canada, and each point in it represents one shop and points in same color belong to same client. And the size of a point tells the NPS rating of its corresponding shop, larger a point is, higher its NPS rating is. From that figure, we can easily notice that all the clients own at least five shops respectively, and shops in the east of U.S. locate more densely than in the west of U.S., shops in U.S. locate more centrally than in

Canada. For example, distribution of shops belonging to Client 14 and Client 9 which are Canada clients is relatively sparser. From the distribution of shops, we can see that company mainly focuses on the business in US, but probably more efforts are needed in Canada due to the remarkably lower NPS rating regarding the apparently smaller size of several points in Canada area.

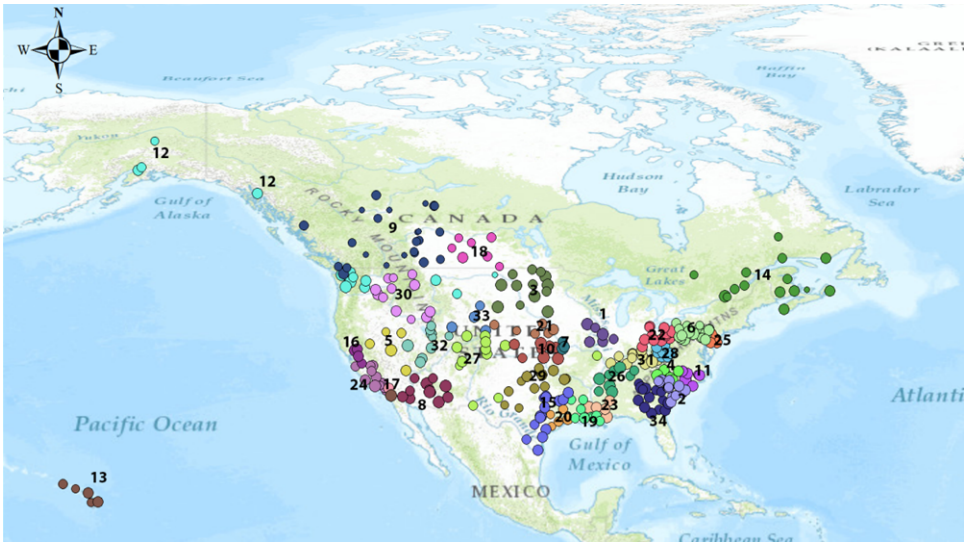


Figure 22: Approximate locations of shops for 34 clients

As a result, computing the geographical distance between two clients is not as simple as we thought of computing the distance between two points. All the 34 clients shown in Figure 22 can be seen as 34 clusters, and non-Euclidean distance is applied here instead of Euclidean distance. Since the core idea of Euclidean distance is to find the centroid for each cluster, retrieving the centroid for clusters that are sparsely distributed can't suit the purpose of representing the general distance between clusters. While non-Euclidean distance resolves such issue, it treats each shop as one point independently and there is no "average" of two points. There are several popular approaches in the area of calculating non-Euclidean distance and a key concept ap-

plied in these approaches is *clustroid* which is defined as the point "closest" to other points, and the clustroid is used as a centroid to compute the intercluster distance. The "closest" point means differentially depending on the actual circumstances, it could be the smallest maximum distance to other points, smallest average distance to other points or smallest sum of square of distances to other point [27]. Intercluster distance is the minimum distance between any two points from two clusters respectively [15]. In order to use a good approach that is most fitting in our situation, basic ideas from these approaches are borrowed and combined together to develop our following definition. With regards to locations of shops in two clusters, every shop in both clusters is selected as a clustroid for once during the computation process, and one cluster are processed at a time, not crossly. The meaning of clustroid is defined as the minimum distance to points in the other cluster. To compute the minimum average distance to points in the other cluster, which is defined as the distance from current cluster to the other one, intercluster distance should be calculated for every clustroid in current cluster and then the average intercluster distance is what we need. So the final physical distance between two clusters would be the average distance of the distance from one cluster to the other and its distance computed in the opposite direction.

Assume the distance that we are going to compute is the one between clients $C1$ and $C2$, and both of them own a few shops. We also assume that $\{S[1,i] : i \in I_{C1}\}$ is a collection of all the shops belonging to $C1$, and $\{S[2,j] : j \in I_{C2}\}$ is a collection of all the shops belonging to $C2$. Firstly starting with $C1$, for each shop $S[1,i]$ where $i \in I_{C1}$, when a shop becomes a clustroid, its intercluster distance - the minimum

distance between it and any shop in $C2$, is calculated and represented as:

$$d(S[1,i],C2) = min\{d(S[1,i],S[2,j]) : j \in I_{C2}\}, i \in I_{C1}. \quad (4)$$

In Equation(4), $d(S[1,i],S[2,j])$ is the distance between a shop $S[1,i]$ in $C1$ and a shop $S[2,j]$ in $C2$, where $i \in I_{C1}$ and $j \in I_{C2}$. $d(S[1,i],C2)$ is the minimum distance from a shop $S[1,i]$ to any shop in $C2$, which is understood as the distance from this shop to $C2$. And the distance from every shop in $C1$ to $C2$ should be computed the same way.

Similarly, when we are done with $C1$ and shops in $C2$ are chosen to be clustroids, we have the distance between any shop $S[2,j]$ in $C2$ and any shop $S[1,i]$ in $C1$, where $i \in I_{C1}$ and $j \in I_{C2}$, and the minimum distance from a shop in $C2$ to any shop in $C1$ will be:

$$d(S[2,j],C1) = min\{d(S[2,j],S[1,i]) : i \in I_{C1}\}, j \in I_{C2}. \quad (5)$$

And distance from every shop in $C2$ to $C1$ are computed as shown in Equation(5).

Distance from $C1$ to $C2$ is the minimum average distance from $C1$ to $C2$ which is calculated by averaging the total of the distances from shops owned in $C1$ to $C2$. The distance from $C2$ to $C1$ is the minimum average distance from $C2$ to $C1$ which can be calculated by averaging the total distances from shops owned by $C2$ to $C1$. Finally, the geographical distance between $C1$ and $C2$, $GeoDist(C1,C2)$, is a balanced distance from $C1$ to $C2$ and $C2$ to $C1$. The complete Equation is:

$$GeoDist(C1,C2) = \frac{1}{2}\times(\frac{\Sigma\{min\{d(S[1,i],S[2,j]) : j \in I_{C2}\} : i \in I_{C1}\}}{|C1|} + \frac{\Sigma\{min\{d(S[2,j],S[1,i]) : i \in I_{C1}\} : j \in I_{C2}\}}{|C2|}). \quad (6)$$

In Equation(6), $|C1|$ and $|C2|$ mean the number of shops owned by $C1$ and $C2$ respectively.

7.2 Construction of Geographical Distance-Based Hierarchical Dendrogram

Based on Equation(6) given in last section, geographical distance between any pair of clients can be retrieved now and a 34×34 distance matrix is generated and used to build our geographical distance based dendrogram. Figure 23 is generated by applying agglomerative clustering algorithm to the distance matrix. Similarly as the hierarchical dendrogram based on semantic similarity in Figure 17, greater the depth of the earliest common ancestor of two clients in the dendrogram, closer they are to each other in geographical distance. The figure clearly demonstrates the division of clients from the aspects of physical location in real world.

This dendrogram is another skeleton for the collection of hierarchically structured recommender systems, and we also run HAMIS with it, exactly like it has been done with the semantic distance based dendrogram. Same two criteria are still applied: NPS rating and F-score, so the resulting dataset have the highest NPS rating and F-score. The only minor difference is that geographical distance replaces semantic similarity as first key measurement, instead of looking for most semantically similar clients, geographically closest clients are needed to continue the procedure.

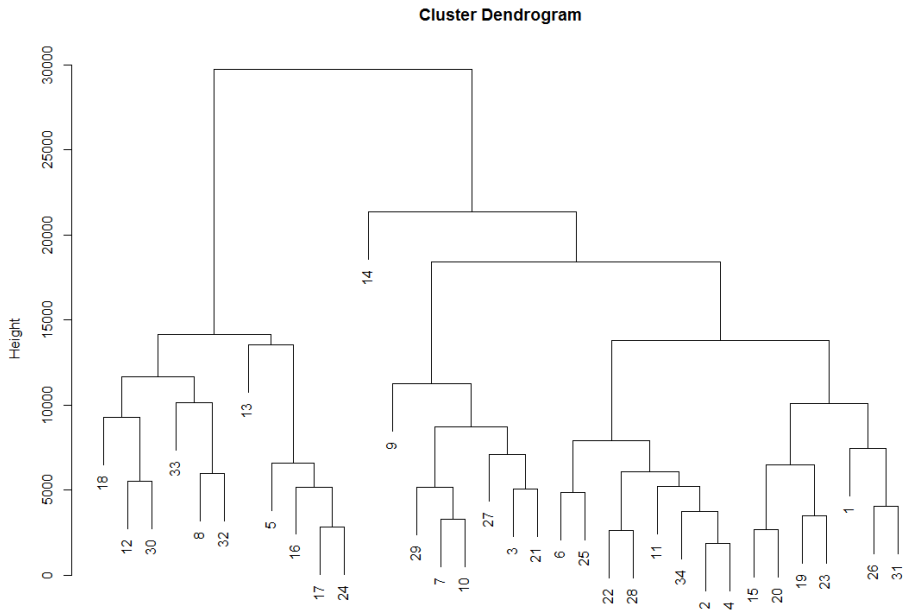


Figure 23: Hierarchical clustering of 34 clients based on geographical distance

7.3 Experiments on HAMIS with Geographical Distance-Based Dendrogram

To show the running process of algorithm HAMIS with geographical distance based dendrogram in our domain detailedly, Client 5 is chosen to become our target client since it has been mentioned as an example of motivating extension with geographical distance earlier. Relevant data used during this procedure is shown in Table 4. As the geographical dendrogram has been given in Figure 23 and the part of it relating to our example is shown in Figure 24, we observe that node {5} representing Client 5 is labeled in green and it is the initial resulting node. Node {5} will be considered to be merged with all the leaf nodes on its sibling side, which are nodes {16}, {17} and {24}. And the strategy of checking these three nodes follows a top down sequence, the earliest node to be checked is the one with the smallest depth. Accordingly, node {16} will be checked first, and nodes {17} and {24} follow. It turns out that node

{16} fails merging with node {5} due to slightly lower NPS rating, which is 0.767 comparing to 0.77, but node {17} is successfully merged with node {5} because NPS rating of node{17} is significantly higher than node {5} and the F-score of merged dataset $\{5, 17\}$ is also greater than the original node {5}. Thus new resulting node $\{5, 17\}$ will be used in the following procedure. Still node $\{5, 17\}$ stays at the same hierarchy until the last candidate node {24} for current hierarchy is checked. The case for node {24} is the same as for the node {16}, thus resulting node $\{5, 17\}$ climbs to its parent node and starts the next round of generalization at a new hierarchy level in the dendrogram. For the node $\{5, 17\}$ at a new position, there is only one candidate on the sibling side, which is node {13}. Despite the NPS rating of node {13} is 0.007 over the initial node {5}, the F-score of newly merged dataset $\{5, 17, 13\}$ is less than $\{5, 17\}$, hence HAMIS stops here with $\{5, 17\}$ returned since it can't be expanded any further.

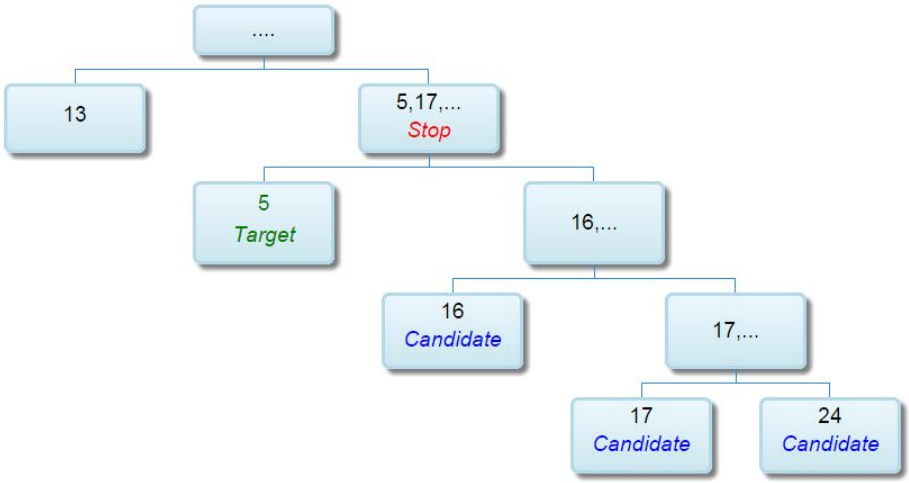


Figure 24: Example of running HAMIS on Client 5 with geographical dendrogram

HAMIS with geographical distance based dendrogram is also performed on the datasets of 34 clients respectively. Figure 25 shows the comparison in size of original

Table 4: NPS rating and F-score of relevant nodes in Figure 24

	$N\{5\}$	$N\{16\}$	$N\{17\}$	$N\{24\}$	$N\{13\}$
NPS rating	0.77	0.767	0.848	0.724	0.777
	$N\{5\}$	$N\{5,16\}$	$N\{5,17\}$	$N\{5,17,24\}$	$N\{5,17,13\}$
F-score	0.743	NA	0.794	NA	0.787

dataset, dataset generalized with semantic dendrogram and dataset generalized with geographical dendrogram which are represented by orange, green and red bars successively. Longer the bar is, larger the dataset is. Overall speaking, applying HAMIS with geographical dendrogram solves the issue left by applying it with semantic dendrogram to some extent, clients 5, 19, 29, 30, 31, 32, 33, 34 who can not be extended by following semantic dendrogram, actually get generalized with geographical dendrogram. It is interesting to find that Client 2 ends with same result of merging with Client 4 with either distance measure, which could possibly imply that Client 2 is a perfect example of showing consistency in semantic similarity and geographical distance. For those clients who achieved remarkably larger resulting dataset by either distance measure, we could speculate that the corresponding distance measure gains more importance to this client than the other distance measure. For example, these 8 clients who can not be generalized by semantic dendrogram but achieved expansion by geographical dendrogram, they may all locate in a competitive environment, as quite a few neighboring clients gain higher NPS rating which is the main reason of allowing these 8 clients get generalized.

By comparing the number of generalized clients with different distance measures, we have a more comprehensive view about how they perform in our experiments.

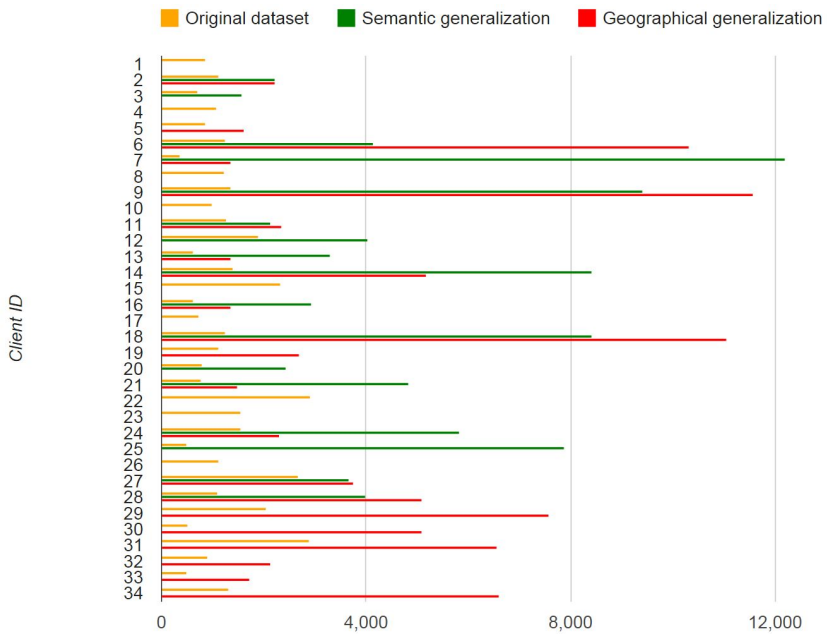


Figure 25: Comparison of performance of HAMIS on 34 clients with semantic/geographical dendrogram

Table 5: Number of generalized clients with specified distance measures

With Geo-distance		**Non-with Geo-distance**	
21		13	
withSemSim	Non-withSemSim	withSemSim	Non-withSemSim
13	8	5	8

Table 5 provides the relevant data to help us do the comparison. It shows that there are 21 clients being generalized with geographical distance and more than the number of clients being generalized with semantic similarity, and 13 out of 21 clients get benefits from both distance measures. Also there are 8 out of 13 clients who failed in generalization with geographical distance, neither with semantic similarity. This inspires us to think if it is true that when a client couldn't get help from neighboring clients, then it is less likely that he/she can get ideas of how to improve its business from semantically similar clients. The relationship between semantic similarity and geographical distance is hard to figure out clearly, but still interesting to look into.

CHAPTER 8: MIXTURE DISTANCE-BASED HIERARCHICAL CLUSTERING

Recommender systems based on semantic similarity and geographical distance have been built separately for all 34 clients with HAMIS applied, but building only one comprehensive recommender system covering both semantic and geographical areas will better meet our goal. To construct such recommender system, a mixture distance measure combining semantic similarity and geographical distance is proposed to generate a new hierarchical clustering dendrogram which is the foundation of applying HAMIS. In order to balance the importance of semantic similarity and geographical distance in constructing the combined measure, assigning weight factors is used and the mathematical model of the new weighted distance between any pair of clients is defined as below:

$$d_{new}(C1, C2) = W_{sem} \times SemSim(C1, C2) + W_{geo} \times GeoDist(C1, C2), \quad where \quad W_{sem} + W_{geo} = 1. \tag{7}$$

In Equation(7), $d_{new}(C1, C2)$ is the new distance between clients $C1$ and $C2$ that we want to compute, and $SemSim(C1, C2)$ and $GeoDist(C1, C2)$ are semantic similarity and geographical distance between $C1$ and $C2$ respectively. Through assigning weight factors W_{sem} and W_{geo} that are appointed to weight semantic similarity and geographical distance respectively, the general distance between $C1$ and $C2$ will be balanced to present the overall difference between them. Traditionally, weight fac-

tors are assigned by designers and appropriate factors are not given randomly, but determined empirically with regards to the actual experiments and experience. As experiments with HAMIS have been done and shown in previous sections, relevant figures can give us some ideas about how to develop our own factors.

Since F-score occupies a key position through our whole research, from evaluating the performance of classification to being a primary condition in designing HAMIS, F-score has been always used as a measurement for guarding the quality of our process, higher the F-score is, better the classifier has been built. Additionally, after applying HAMIS to all 34 clients with either geographical dendrogram or semantic dendrogram, the extended dataset for each client has not only the highest NPS score, but also the best F-score, and the final F-scores for each client retrieved in semantic way and geographical way are not the same which tells us the quality of knowledge hidden in dataset to some extent. Therefore, using the final F-score provided by the generalized dataset to build weight factors is our best choice. The following F-score based weight factors are proposed to calculate the new distance.

$$W_{sem} = \frac{FS'_{semantic}}{FS'_{semantic} + FS'_{geographical}}; \quad W_{geo} = \frac{FS'_{geographical}}{FS'_{semantic} + FS'_{geographical}}; \qquad (8)$$

Where

$$FS'_{semantic} = \sum_{k=1}^{34} FS_{semantic}(k) \quad \text{and} \quad FS'_{geographical} = \sum_{k=1}^{34} FS_{geographical}(k). \qquad (9)$$

Equation(9) shows $FS'_{semantic}$ and $FS'_{geographical}$, the overall F-score of semantic generalization and geographical generalization respectively, are computed by summing up the F-scores from extended datasets of all 34 clients with semantic dendrogram

and geographical dendrogram respectively, and the overall F-score $FS'_{semantic}$ and $FS'_{geographical}$ are used to build our weight factors as shown in Equation(8). Generally speaking, for one single client, if the overall classifiers built from semantic generalization are better than from geographical generalization, then its semantic similarity should occupy more percentage comparing with geographical distance to construct the new distance between it and all the other clients.

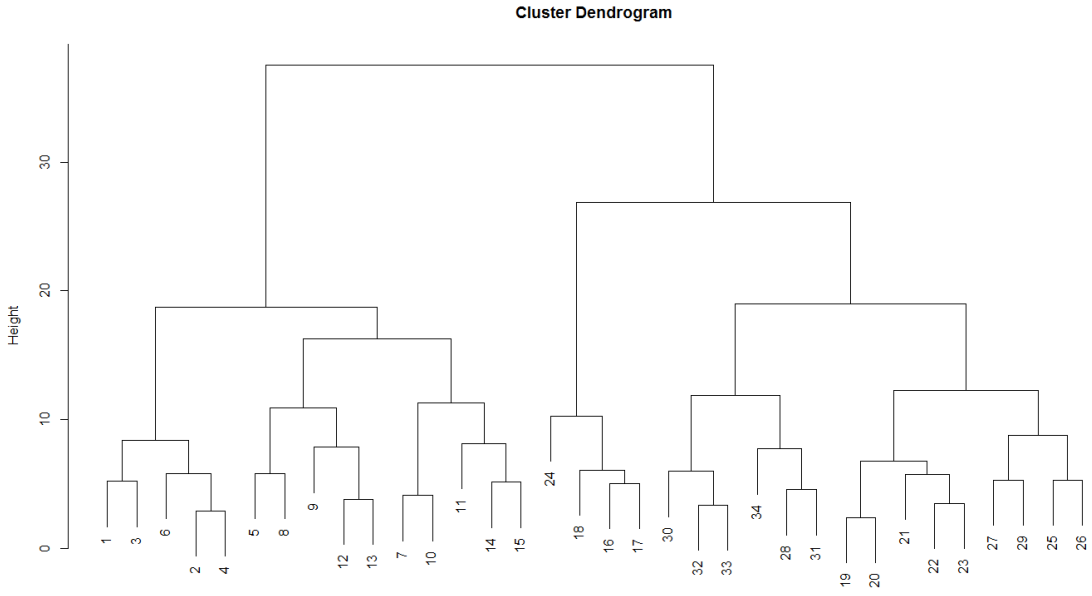


Figure 26: Hierarchical clustering of 34 clients based on mixture distance

Therefore, a new hierarchical dendrogram has been generated based on the proposed mixture distance and it is shown above. Then HAMIS is applied to 34 clients' datasets based on the new dendrogram and the extended dataset for each client will be used to extract action rules and meta-actions.

CHAPTER 9: EXTRACTION OF META-ACTIONS

Our ultimate task is to extract useful action rules and meta-actions for helping clients to improve NPS rating. Meta-actions are the actions that should be executed by clients to trigger the specific group of action rules, and they can be called triggers as well [40] [57] [56]. In our domain, comments left by customers are stored as texts after the questionnaires and they are our resources of generating proper triggers that are applicable to clients. In this chapter, we mainly focus on mining meta-actions which are the actual solutions to trigger action rules and ulteriorly improve NPS ratings, as meta-actions plays the most critical role in our recommender system and the process of extracting them turns out to be one of the most challenging problems in the entire project.

The strategies for discovering action rules have been quite mature so far. In early research, since action rules can be constructed from two classification rules [41], pre-existing algorithms for discovering classification rules can be used to constructed action rules, such as *LERS* [16] and *ERID* [12]. Later in [18], an algorithm for direct extraction of action rules from a data table was proposed. However, the area of mining meta-actions is still blossoming during these years. Although [56] presents a methodology of mining meta-actions in medical related field, it concentrates on selecting meta-actions to achieve preferable effect under the assumption that meta-actions are already known. The process of discovering and identifying proper meta-

actions is what we need and have accomplished in this stage.

9.1 Introduction of Meta-Actions

In Chapter 4.1, the concept of action rule has been introduced, and it can be seen as a set of atomic actions that need to be made happen for achieving the expected result. Meta-actions are the actions that need to be executed in order to trigger corresponding atomic actions. The concept of meta-action was initially proposed in [58].

Table 6: Meta-actions influence matrix for S

	a	b	d
$\{M_1, M_2, M_3\}$		$b_1 \rightarrow b_2$	$d_1 \rightarrow d_2$
$\{M_1, M_3, M_4\}$	a_2	$b_2 \rightarrow b_3$	
$\{M_5\}$	a_1	$b_2 \rightarrow b_1$	$d_2 \rightarrow d_1$
$\{M_2, M_4\}$		$b_2 \rightarrow b_3$	$d_1 \rightarrow d_2$
$\{M_1, M_5, M_6\}$		$b_1 \rightarrow b_3$	$d_1 \rightarrow d_2$

Table 6 shows an example of influence matrix which describes the relationships between the meta-actions and atomic actions influenced by them. In the table, a, b and d are stable attribute, flexible attribute and decision attribute respectively in decision system S, and $\{M_1, M_2, M_3, M_4, M_5, M_6\}$ is a set of meta-actions which hypothetically trigger action rules generated from S. Each cell in a row tells an atomic action that can be invoked by the set of meta-actions listed in the first column of that row. Take the first row for instance, it shows that the atomic actions $(b_1 \rightarrow b_2)$ and $(d_1 \rightarrow d_2)$ can be activated by executing meta-actions M_1, M_2 and M_3 together. Unlike the traditional influence matrix in [57] and [56] which involves only one single meta-action in each row, here the transaction of atomic actions in our domain relies on one or more meta-actions.

Clearly, an action rule can be triggered with the set of meta-actions in one single row in influence matrix as long as all of its atomic actions are listed at the same row. Otherwise, selecting a proper set of meta-actions combined from multiple rows becomes necessary. If we would like to activate action rule r assumed as $r = [\{(a, a_2), (b, b_1 \rightarrow b_2)\} \Rightarrow \{(d, d_1 \rightarrow d_2)\}]$, which is the composition of two association rules r_1 and r_2, where $r_1 = [\{(a, a_2), (b, b_1)\} \rightarrow (d, d_1)]$ and $r_2 = [\{(a, a_2), (b, b_2)\} \rightarrow (d, d_2)]$, it is quite obvious that the combination of meta-actions listed in the first and second row in Table 6 could trigger r, as meta-actions $\{M_1, M_2, M_3, M_4\}$ cover all required atomic actions (a, a_2), $(b, b_1 \rightarrow b_2)$ and $(d, d_1 \rightarrow d_2)$ in r. On the other hand, one set of meta-actions could possibly trigger multiple action rules, like the meta-action set $\{M_1, M_2, M_3, M_4\}$ triggers not only r as mentioned but also action rule $[\{(a, a_2), (b, b_2 \rightarrow b_3)\} \Rightarrow \{(d, d_1 \rightarrow d_2)\}]$ by following the second and forth row in Table 6, if such action rule exists in S. Also some action rules can be invoked by more than one set of meta-actions. By selecting a proper set of meta-actions we could benefit in triggering larger number of action rules.

Supposing a set of meta-actions $M = \{M_1, M_2, ..., M_n : n > 0\}$ can trigger a set of action rules $\{r_1, r_2, ..., r_m : m > 0\}$ that covers certain objects in S with no overlap, so the coverage or support of M is the summation of the support of all covered action rules, which is the total number of objects at the initial state that are going to be affected by M in dataset. And the confidence of M is calculated by averaging the confidence of all covered action rules.

- $sup(M) = \sum_{i=1}^{m} sup(r_i)$

- $conf(M) = \frac{\sum_{i=1}^{m} sup(r_i) \cdot conf(r_i)}{\sum_{i=1}^{m} sup(r_i)}$

Furthermore, the effect of applying M, that is going to be used in next chapter, is defined as the product of its support and confidence ($sup(M) \cdot conf(M)$), and it represents the number of objects in the system that are expected to be improved by applying M, which is the foundation of calculating the increment of NPS rating caused by it. Therefore, greater the effect of M is, more the increment of NPS rating will be. On the other hand, there could be overlaps in the objects covered by a set of action rules in reality. In such case, the support (coverage) of a set of meta-actions M is actually the number of distinct objects involved with triggered action rules, which requires us to keep tracking the objects invoked by certain meta-actions and keep out of redundant ones.

9.2 The Process of Generating Meta-Actions

As mentioned earlier, triggers aiming at different action rules should be extracted from respectively relevant comments which are the sources for generating meta-actions in our domain. Let's assume that action rule $r = [\{(a, a_2), (b, b_1 \rightarrow b_2)\} \Rightarrow \{(d, d_1 \rightarrow d_2)\}]$ mentioned earlier is our target, and two classification rules r_1 and r_2 have been used to construct r, so the clues for generating meta-actions are in the comments stored in database satisfying the description (a, a_2), (b, b_1) and (d, d_1) or (a, a_2), (b, b_2) and (d, d_2).

To generate meta-actions from a determined set of comments, five steps are designed to accomplish this task: *1*) Identifying opinion sentences and the orientation with localization; *2*) Summarizing each opinion sentence using discovered dependency

templates; *3*) Identifying feature words in opinion summarizations; *4*) Aggregating opinion summarizations based on feature words; *5*) Generating meta-actions with regards to given suggestions. There are characteristics differing from others in each step that will be presented in following sections. The whole process involves not only the sentiment analysis, text summarization and feature identification, but also generation of appropriate suggestions as meta-actions, which is more representative for our purpose. Before going to details of each step in the following subsections, adjusting the order of steps 1-4 is another uniqueness of this approach. This adjustment is based on two reasons: *a*) Data is uncleaned. Unlike reviews on particular products or experience mentioned in other research, quite a number of comments in our domain are useless due to lack of opinion orientation for our purpose, even though they are mentioning certain aspects; *b*) Former steps can benefit latter steps. After comparing the methods of extracting aspects (features) from different formatted comments, it turns out that dealing with the reviews in short segments is more efficient without compromising their effectiveness. Therefore, ordering the first four steps in such way accelerates the process by eliminating useless information and makes preparing the data easier to handle for the next step. Most crucially, the accuracy will not be damaged, but improved.

9.2.1 Identification of Opinion Sentences and the Orientation with Localization

To identify an opinion sentence which expresses customers' sentiment, the presence of opinion words is considered as a standard sign. Initially adjectives are usually used as the main opinion words, like Hu and Liu have used only adjectives in [23], and

he has generated two sets of opinion words expressing positive and negative feelings. Although these sets of opinion words are still growing continually, using them as the only references to detect opinion words and their orientation is not sufficient due to its generality. In some local scenarios, the lists of opinion words must be expanded more broadly concerning some neutral words with implicit polarity. For example, a comment *"the charge was too high"* can not be identified with the given lists, since the adjective *"high"* is not recognized as a positive nor negative word by them, but it definitely presents a useful message reflecting customers' negative opinion about the price, so the word *"high"* can be treated as a negative opinion word in this case. Similarly, other special neutral words or phrases can be added as opinion words if they reflect oriented meanings under certain circumstances without confusion, but such addition strongly relies on designers' knowledge about the domain.

Hence based on our own learning, some extra adjectives and verbs which are not included in provided lists are added into our library of opinion words with clarified orientations. Totally speaking, four types of words that could have orientations are used to filter the appearance of opinion words and they are: *verb, adjective, adverb* and *noun*. As long as a word in a sentence tagged as any one of the four types exists in either extended list of opinion words, this sentence is an opinion sentence and the orientation of a tagged opinion word depends on its ascription to which list. The orientation of a sentence is determined by following the basic principles summarized in [29]. However, we care more about the orientation of segments summarized in next step with related opinion words, so only the orientation of opinion word is marked in this part.

9.2.2 Summarization of Opinion Sentence based on Dependency Relationships

With opinion sentences identified, shortening them into segments is an important procedure for following steps. Relevant research like [61] and [53] construct feature-opinion pairs with grammatical rules describing the relationships between features and opinion words. Without pre-identified features in opinion sentences, extracting summaries from every sentence by following certain grammatical relations associated with opinion words solely is also applicable and sufficient for two reasons. Firstly, unlike other relevant works, there is no need of Part-of-Speech (POS) [31] tagging for us in this step, as the grammatical structure of a sentence is the only factor that we depend on; Secondly, the grammatical relations of the expected portion most closely connecting to the opinion words in a sentence can be summarized based on the knowledge of linguistics and used to extract a short but meaningful segment from a complete sentence.

The foundation of this step is the grammatical relations defined by Stanford Typed Dependencies Manual [32] and generated by Stanford Parser. A dependency relationship describes a grammatical relation between a governor word and a dependent word in a sentence and it is represented as *d(G, D)*, where d is one type of dependency among approximately 50 defined dependencies in [32], G and D are the governor and dependent respectively. With the comprehensive representation of dependencies, the nearest necessary relations associated with opinion words can be identified. Moreover, the types of dependencies that could link to opinion words straightforwardly rely on the type of opinion words, and Table 7 demonstrates all the discovered dependency

templates for four types of opinion words respectively.

Table 7: Dependency templates for extracting sentence segments

Type of opinion words	Dependency template	Segment result
Noun	nsubj(op , W_{nsubj})	W_{nsubj} + op
	prep(op, W_{prep}) + pobj(W_{prep}, W_{pobj})	op + W_{prep} + W_{pobj}
	dobj(W_{dobj}, op)	W_{dobj} + op
Adjective	nsubj(op , W_{nsubj})	W_{nsubj} + op
	amod(op, W_{amod}) + vmod(W_{amod}, W_{vmod}) + dobj(W_{vmod}, W_{dobj})	op + W_{amod}+ W_{vmod}+ W_{dobj}
	xcomp(op, W_{xcomp}) + dobj(W_{xcomp}, W_{dobj})	op + W_{xcomp} + W_{dobj}
	prep(op, W_{prep}) + pobj(W_{prep}, W_{pobj})	op + W_{prep} + W_{pobj}
	pcomp(op, W_{pcomp}) + dobj(W_{pcomp}, W_{dobj})	op + W_{pcomp} + W_{dobj}
	vmod(op, W_{vmod}) + dobj(W_{vmod}, W_{dobj})	op + W_{vmod} + W_{dobj}
Adverb	advmod(op, W_{advmod})	W_{advmod} + op
Verb	dobj(op, W)	op + W
	prep(op, W_{prep}) + pobj(W_{prep}, W_{pobj})	op + W_{prep} + W_{pobj}
	xcomp(op, W_{xcomp}) + dobj(W_{xcomp}, W_{dobj})	op + W_{xcomp} + W_{dobj}
	advcl(op, W_{advcl}) + dobj(W_{advcl}, W_{dobj})	op + W_{advcl} + W_{dobj}

[1] *op* denotes opinion word.

In Table 7, for each type of opinion words, all the other words linked with opinion words directly or indirectly are labeled as W_* regarding the dependency type. In one template, there could be more than one dependency due to the existence of phrases, and the segment result is the combination of all involved words in a sequence as shown in the third column. When there are two or more templates associated with one opinion word, words from all detected templates will be combined sequentially as the words appear in the sentence. For example, if only dependency *nsubj* is discovered in a sentence associated with a *noun* opinion word, then the final segment is extracted

by simply following the first row in Table 7, which gives us "W_{nsubj} *op*" with space between words. If additional dependencies *prep* and *pobj* are discovered along with *nsubj* for a *noun* opinion word in a sentence, then the final segment result becomes "W_{nsubj} *op* W_{prep} W_{pobj}" by combining the results from both templates and ordering the words according to their locations in the sentence. During the process of exploring the dependencies in a sentence, it is extraordinarily necessary to detect the existence of a negation relation linked to a opinion word, if there is, then the opinion word *op* will be changed to *not op* before being used in the final segment.

9.2.3 Identification of Feature Words in Opinion Summarizations

Identifying feature words from opinion summarizations is a simpler case now, because there is at most one feature in each segment with one opinion word, sometimes there is no valid feature existing in invalid segments. As the supervised pattern mining method - label sequential rule mining in [30] and [29] is proposed to handle reviews formated similarly as our segments, the similar idea has been broadened with our own observations to fulfill this step.

In the training dataset, the column is used to mark sequence of words in each segment. For example, the longest segment contains five words, then there are five columns in training dataset and each of them is assigned name "*word#*" to indicate the position of values in segments. In each row, every word in one segment is put in its corresponding column from the beginning, along with their POS tags. Last but not the least, every feature word in each segment will be identified and replaced with label [feature] manually, so a segment like "pleased with attitude" will be finally

represented as "pleased_VB with_IN [feature]_NN" in our training dataset, where tags VB, IN and NN denote for verb, preposition or conjunction and noun respectively.

Then association rules are mined from the training set with assistance of WEKA, and only the ones with label [feature] at the right hand side are kept and transformed into patterns. Following the example given above, the association rule generated for it is: word1=pleased_VB ==> word3=[feature]_NN and the pattern transformed from it becomes: <pleased_VB> <>< [feature]_NN>. Inspiring by summarizing quite a lot of valid association rules and patterns retrieved from them, we cannot help thinking that the tags actually help generalize the recognition of features, especially there are limited kinds of tags appearing in our segments. For instance, <excellent_JJ> <[feature]_NN> and <hard_JJ><[feature]_NN> (JJ denotes adjective) are two patterns which form a more general one <JJ><[feature]_NN> indicating that the noun appearing right after the adjective could be the feature under certain possibility. With regards to such observations, we are more inclined to build general patterns with only tags to predict features, which remarkably decreases the number of useful patterns and increases the efficiency.

9.2.4 Aggregation of Opinion Summarizations

In many sentiment analysis works, this step is to generate a final review summary for all discovered information about features and opinions, and also rank them by their appearances in the reviews. Besides that, more of our attention in this step is put on avoiding the redundancy of extracted segments and clustering segments into different classes. To fulfill this task, five feature classes are defined with a list of seed

words or phrase as shown in Table 8, and the number of seed words for each class is consistently growing larger and larger to enlarge its coverage as more learning process has been done. According to the existing flexible attributes that are introduced in previous chapters, "Service" and "Communication" classes shown in Table 8 are quite understandable and obvious. Meanwhile, classes "Staff" and "Technician" are separated with regards to the suggestions from our consultants from the company, since "Staff" should be more inclined to represent the administrative personnel or dealers while "Technician" is more related with the person who is responsible for fixing machines on the field or in shop. Classes "Invoice" and "Price" are also divided into two parts concerning the difference between concepts of billing procedures and the actual amount charged at the end of the service, which is beneficial for more distinctly distinguishing problem laid in different classes. To cluster a segment into the corresponding class, its feature word or the base form of its feature word is going to be checked to see whether it exists in any list of seed words.

Table 8: Feature classes and relevant seed words

Feature Class	Feature Seed Words
Service	service, experience, performance, job, work, part(s), equipment(s), do, done, complete, completed, fix, fixed, run
Communication	communication, communicate, respond, responded, response, call, phone, contact, reply, return, returned, receive, receiving, answer, speak
Staff	staff, staffed, dealer, manager, guy, team, employee(s), friendly, attitude
Technician	technician, tech, mechanic, friendly, skill(s), diagnosis, diagnosing, misdiagnoses, supervision
Invoice	invoice, invoicement, bill, billed, billing, match, account
Price	price(s), pricey, pricing, charge, charged, amount, money, discounts, discount, quote, pay, paid, expensive

However, in order to generate meta-actions, each class has been broken into several subclasses regarding more specific aspects in a class that segments involves and their are defined based on our common sense and learning of extracted opinion summarizations as Table 9 shown.

Table 9: Feature classes, its subclasses and their seed words

Feature Class	Subclasses	Examples of Seed Words
Service	completeness	correctly, properly, well, completed, fixed
	timeliness	timely, earlier, time(s), days, weeks, months
Communication	proactiveness	would like, respond, heard back
	ease of contact	difficult, hard, better, poor
	timeliness	timely, delay, quicker, slow, nobody
	quality (effectiveness)	effectively, failed
Staff	kindness	good, better, great, nice, friendly, gracious
	knowledgeability	knowledgeable, clueless, diagnosis, skill, professional, trained, inexperienced
	resource	enough, available, resourceful, staffed
Technician	kindness	good, better, great, nice, friendly, gracious
	knowledgeability	knowledgeable, clueless, diagnosis, skill, professional, trained, inexperienced
	resource	enough, available, resourceful, staffed
Invoice	accuracy	wrong, right, correct, incorrectly, correctly, incorrect, accurate, accurately
	expectation	match, matching, matched, expected
	timeliness	quick, quicker, quickly, slow, slowly, late, timely, time
Price	competitiveness	high, expensive, outrageous, fair, fairly, good, competitive, poor, excellent, reasonable, excessive

9.2.5 Generation of Personalized Meta-Actions

The positive opinions indicate the satisfying behaviors that should be kept doing, so the meta-actions for them are called *keeping* actions. While negative opinions show the undesirable behaviors that should be fixed, so their solutions are referred as *fixing* actions. Sometimes it is straightforward to create *keeping* actions, since the positive

segments can be used directly and they are explicit enough for users to understand and adopt. However, for negative segments, reversing them literally or removing the negation is not always helping. To provide the most suitable fixing actions, consulting with company partners who have expertise in this field is necessary and useful. After detailed discussions, meta-actions targeting positive or negative aspects in each feature class are determined and shown in Table 10.

In Table 10, two meta-actions have been generated to cover positive and negative sides of reviews for every subclass in feature classes. By subclasses, we mean the more specific aspects that could be designated by segments and have been introduced in last section. For example, staff's attitude is one of the subclasses in class *"Staff"*, and the *keeping* meta-action for positive reviews and *fixing* meta-action for negative reviews concerning staff's attitude are "nice staff" and "staff show care and respect to customers", which reaches our goal of suggesting the shop what to do for improvements.

9.3 Experiments

To implement our system for mining meta-actions, several existing tools from other projects have been used. Stanford NLP part-of-speech tagger and lexicalized parser are used for generating POS tags [31] and identifying the dependency relations [32]. The lists containing positive and negative words respectively from Liu [30] are applied to detect opinion words and their polarities, ulteriorly the orientation of the segments.

The system is built on JAVA and the sample used to test our approach contains 116 sentences which are manually labeled with relevant information including all the

Table 10: Feature classes, its subclasses and their meta-actions

Feature Class	Subclasses	Meta-Actions
Service	completeness	(k) service done correctly (f) ensure service done correctly
	timeliness	(k) service done timely (f) complete service in timely manner
Communication	proactiveness	(k) keep proactive communication (f) improve proactive communication
	ease of contact	(k) keep ease of contact (f) improve ease of contact
	timeliness	(k) keep responding to customers timely (f) decrease dealer response time
	quality (effectiveness)	(k) keep quality of communication (f) improve quality of communication
Staff	attitude	(k) nice staff (f) staff show care and respect to customers
	knowledgeability	(k) knowledgeable staff (f) improve staff's knowledge
	resource	(k) sufficient staff (f) have a proper number of staff
Technician	attitude	(k) nice technician (f) show care and respect from technician
	knowledgeability	(k) knowledgeable technician (f) improve technicians' knowledge
	resource	(k) sufficient technician (f) need more technicians
Invoice	accuracy	(k) accurate invoice (f) make invoice accurate and clear
	expectation	(k) reasonable invoice (f) properly set invoice expectations
	timeliness	(k) invoice billed timely (f) more timely invoicing
Price	competitiveness	(k) competitive price (f) justify price

[1] k: keeping action.
[2] f: fixing action.

expected results for each step, such as opinion sentence (or not) and opinion words orientation for the first step, and so on. In order to evaluate the performance, three common measurements are defined as shown below, where *Num(*)* represents the

number of records as $*$ described.

$$precision = \frac{Num(generated\ results\ matching\ our\ expectation)}{Num(all\ generated\ results)} \tag{10}$$

$$recall = \frac{Num(generated\ results\ matching\ our\ expectation)}{Num(all\ expected\ results)} \tag{11}$$

$$F - score = \frac{2 \times precision \times recall}{precision + recall} \tag{12}$$

Table 11: Experiment results of major steps

	Precision	Recall	F-score
Opinion Sentence Identifier	0.833	0.696	0.758
Opinion Sentence Summarizer	0.883	0.8	0.839
Feature-Opinion Pair Mining in [61] using method in [23]	0.403	0.617	0.488
Feature-Opinion Pair Mining in [61] using method in [61]	0.483	0.585	0.529
Feature Words Identifier	0.81	0.71	0.757
Feature Mining in [61] using method in [23]	0.457	0.708	0.555
Feature Mining in [61] using method in [61]	0.595	0.682	0.636
Feature Aggregator	0.78	0.733	0.753
Meta-Action Generator	0.78	0.75	0.764

After the sample data is processed with the proposed procedures, precision, recall and F-score are computed and shown in Table 11, and also some data from other work are included. Firstly, although there is no other comparable results for *Opinion Sentence Identifier*, its performance is very satisfying, its F-score is over 0.75 and the precision is over 0.8. Secondly, if comparing the performance of *Opinion Sentence Summarizer* and *Feature Words Identifier* in our work to the average results of feature-opinion pair mining and feature mining in [61] using approaches from [23] and [61] respectively, it is quite obvious that our approach achieves much better results in

all three measurements, F-score of our Summarizer reaches as high as 0.839 while F-score of aspect identifier is over 0.75. The accuracy of *Feature Aggregator* shown in the last row of Table 11 is very optimistic. For *Meta-Action Generator*, there are 30 *fixing* actions provided in the list, and the performance of mapping them to specific segments is very acceptable, and its F-score is 0.764. Thus, the experiments confirm our expectation concerning the proposed method.

Generally speaking, the typical procedure of feature-based sentiment analysis in [23], [24] and [7] proceeds without opinion sentence summarization. Later, although [61] and [53] have completed some work on opinion summarization by mining feature-opinion pairs, our approach accomplishes opinion summarization by following the discovered templates of dependency relations involving expanded opinion words solely, and features in the summarized opinion segments are recognized by applying tag-dominated patterns transformed from association rules. Compared with other relevant work, our *Sentence Summarizer* and *Feature Words Identifier* achieve higher accuracy in the experiments, which proves the effectiveness of the atypically ordered and accordingly adjusted procedures. Besides adapting the traditional sentiment analysis into our project, designing the unique procedure - generation of meta-actions - resolves the demands for providing proper solutions to exposed problems, and the experiments also demonstrate its impressive effect. Moreover, we believe this process can be applied to other areas for solving the discovered problems with their personalizations.

CHAPTER 10: IN SEARCH FOR BEST META-ACTIONS TO BOOST BUSINESS REVENUE

In previous chapter, the method of mining meta-actions from customers' reviews in text format has been proposed and implemented. However, it turns out that the discovered action rules need more than one meta-action to be triggered, and each action rule is triggered by different groups of meta-actions. In the ideal scenario, all discovered action rules should be triggered. The way and the order of executing triggers causes new problems due to the commonness, differential benefit and applicability among sets of meta-actions, so finding the smallest sets of meta-actions triggering all action rules becomes important. Each meta-action has some parameters assigned to it covering its effect and so on, but the applicability or feasibility of meta-actions in practice should be judged by professionals in the field, our concentration is put on designing a strategy to hierarchically sort and arrange so called meta-nodes (used to represent action rules and their triggers in a tree structure) in a certain way as well as to compute the effect of each meta-node. Based on that, users will have more concrete clues for adopting groups of meta-actions in reality by following the paths in trees built from these mate-nodes, as long as the increment of NPS by triggering them is above certain threshold. In another word, some meta-actions may not be executed at all due to their unsatisfying impact.

10.1 Background

The importance of a well organized and informative result is mentioned briefly earlier. Now, the detailed reasons for pursuing that are explained below:

1. The commonness between sets of meta-actions tells about an enhancement. Although actions rules are triggered by different sets of meta-actions, there are some meta-actions which exist in different sets. The commonness between two sets of meta-actions is defined as the percentage of the common meta-actions both sets occupy in the smaller set. If the commonness between two sets equals to 1, then the smaller set belongs to the larger one. In this case, with the larger set of meta-actions applied, besides the action rules covered by it, the action rules covered by the smaller set are also invoked. So the effect of executing the larger set of meta-actions is expected to be enhanced with the effect of smaller one added.

2. The effect of sets of meta-actions tells about a preference. Generally speaking, in contrast to the smaller set of meta-actions, it is understandable that the larger set to which a smaller one belongs is more preferable to clients due to its greater effect. On the other hand, for disjoint sets of meta-actions, the selection preference is no longer built on the commonness, but to some extent directly on their effect. Another word, the ones with more significant effect are certainly more preferable to clients.

3. The applicability of certain meta-actions tells about a rejection. In our domain,

an action rule will not be effective unless all required meta-actions have been taken. So, clients are encouraged to perform all given meta-actions to achieve the expected improvement. But in real life, we can foresee that some of meta-actions would be rejected by clients under great possibility as they are more difficult or pricey to be executed than other ones and they are not worth it in clients' opinion. For example, if clients are recommended with a set of meta-actions in which lowering the price is included, clients will not like it according to our consultants, even though they are advised by the system to do so for pleasing the customers who complain about the charge. So meta-actions with poor applicability will lead to rejections from clients.

Considering these three reasons together, the decision of choosing meta-actions to apply does not solely depend on us any more, as our system can calculate the effect of sets of meta-actions, but we are not the experts in estimating the difficulty or cost of adopting certain meta-actions in the field. The decision must be done by clients and technicians who have professional knowledge about this practice. Also, it should be mentioned that clients probably would ignore a set of meta-actions if its effect and cost are remarkably unbalanced and there is an alternative. So the final decision should be determined by our clients who weight the effect and cost of applying some actions with their experience and expertise, which proves the necessity of developing this strategy.

The strategy aims to create actionable paths that are well classified from groups of meta-actions and lead to certain increment of NPS rating. These actionable paths

can be seen as a forest of trees where each tree is made of one or more meta-nodes that are hierarchically structured regarding their relation. A meta-node is a potential choice for clients to consider and it represents a set of action rules and their triggers. So, the effect of a meta-node is the effect of the group of meta-actions in it. In a tree (or path), the connection between two nodes is a parent-child relationship and it indicates that the commonness between the groups of meta-actions contained in them equals to 1. So the node with smaller set of meta-actions is defined as the parent of the node with larger set and it is put on an upper level. Lower a meta-node is located in a tree, greater its effect is. The parent-child relationship is a many to many mapping, as one node can have more than one parent, and a parent node can have more than one child. Details concerning the process of generating these trees are illustrated thoroughly in the following sections.

10.2 Advanced Matrix: Transformation of Influence Matrix

To find triggers for action rules, influence matrix introduced in Section 9.1 is a semi-product for us and it should be transformed into an advanced representation to demonstrate the triggers for each rule straightforwardly. Table 12 is an example of advanced matrix. In the table, there are n meta-actions in total and M_i represents every single meta-action, where $0 < \mathrm{i} < \mathrm{n} + 1$. There are m action rules and rule j denotes every generated action rule, where $0 < \mathrm{j} < \mathrm{m} + 1$. For each row, each cell corresponding to the meta-action that is responsible for triggering rule j is filled with 1, while other cells are filled with 0.

Advanced matrix is transformed from influence matrix. Based on advanced matrix,

Table 12: Advanced matrix: action rules and their triggers (meta-actions)

	M_1	M_2	M_3	...	M_n
rule 1	0	1	1	...	1
rule 2	1	0	1	...	1
rule 3	0	0	1	...	1
rule 4	0	1	0	...	1
...	...	...	...	...	...
rule m	0	0	1	...	1

the association between meta-actions and action rules becomes more apparent and it is easy to sort all the meta-actions by their popularity, which is the basis of the following procedures.

10.3 Presentation of the Strategy

As mentioned earlier, the goal of our strategy is to build a forest of trees where each tree is composited by meta-nodes that are linked via a parent-child relationship. Algorithm 2 is designed to organize the advanced matrix in a hierarchical manner and generate a list of meta-nodes T, so it is obvious that an advanced matrix M should be given. At the beginning of Algorithm 2, meta-list and rules are prepared, which are a list of meta-actions descendingly sorted by their popularity in M and a set of all action rules in M respectively. Meanwhile, a map *MetaMap* and a list T are initialized as well. *MetaMap* is created to store the mappings (entries) from sets of meta-actions to their sets of action rules during the entire process. Generally speaking, the content in an entry or mapping in *MetaMap* is identical to a meta-node at the end of the algorithm and each action rule can be involved with only one mapping. The list T will be used to store the continually generated meta-nodes and it will be our final product.

Algorithm 2 Hierarchically Organize Triggers Algorithm

Input: M: advanced matrix containing action rules and their corresponding triggers.
Output: T: a list containing all generated meta-nodes.

```
Generate meta-list: a list of meta-actions that are descendingly sorted by their popularity in M;
Generate rules: a set of all action rules in M;
Initialize MetaSets (meta-action-set, mapped-rules);
Initialize a list T for storing all generated meta-nodes;
for meta ∈ meta-list do
  for r ∈ rules do
    if meta is one of r's triggers then
      retrieve entry E (meta-action-set, mapped-rules) for r ∈ mapped-rules from MetaSets;
      if E is valid then
        add a new entry E'(meta-action-set ∪ {meta}, {r}) to MetaSets;
        mapped-rules = mapped-rules \ {r};
        if mapped-rules is empty then
          remove entry E (meta-action-set, mapped-rules) from MetaSets;
        else
          keep entry E (meta-action-set, mapped-rules) in MetaSets;
        end if
      else
        retrieve entry E' ({meta}, mapped-rules);
        put entry E' ({meta}, mapped-rules ∪ {r}) to MetaSets;
      end if
      if meta-action-set ∪ {meta} trigger r completely then
        T = TreeEditor(meta, r, meta-action-set, T);
      end if
    end if
  end for
end for
Sort meta-nodes in T by their size ascendingly and effect descendingly.
return T;
```

With a sorted list of meta-actions, the algorithm repeatedly runs the main part with one meta-action at a time from the most frequent to least frequent one. Given a meta-action (which is represented as *meta*) in each round, all the action rules are iterated one by one and only the ones associated with *meta* in M will be continued to the following procedures. For each continued action rule (which is denoted as r), the

existence of a mapping established in *MetaMap* involving r differs the way of handling it and we can easily tell its existence by attempting to retrieve a mapping E associated with r from *MetaMap*. If the mapping is valid, it indicates that other triggers for r have been processed and stored in E before *meta*, so a non-empty set of meta-actions (which is denoted as *meta-action-set*) is retrieved along with its *mapped-rules* set from E; Otherwise, a new mapping from meta to r should be established instead. In terms of a valid mapping, it is defined as a mapping with a non-empty key (set of meta-actions) in our domain, so an empty entry will be found if a mapping is invalid. For the former situation with a valid existing mapping discovered, it is apparent that a new mapping E to r has to be built due to the enlargement of its triggers, and the existing mapping E has to be updated accordingly to keep the distinctness of action rules. Building a new mapping in *MetaMap* by creating a new entry E with *meta* added into the retrieved *meta-action-set* and mapped to r is straightforward, so is the removal of r from mapped-rules in the existing entry E. But there is no necessity of keeping E if its action rule set becomes empty after the removal, with regards to the definition of a valid mapping. Similarly, building a new entry E with only $\{meta\}$ mapped to $\{r\}$ in *MetaMap* for the latter situation is simple as well. However, it is possible that a mapping from *meta* already exists. If this is the case, which implies another set of actions rules that can be triggered by *meta* have already been stored, then r should be added into the existing set of action rules mapped from *meta*, instead of creating a new mapping.

Every time a new mapping (or entry) E' is put into *MetaMap*, no matter if it is from an existing mapping or not, Algorithm 3 will be called on the condition that current

Algorithm 3 TreeEditor

Input: *meta*: a single meta-action.
r: an action rule.
meta-action-set: a set of meta-actions.
T: a tree storing the current added meta-nodes and their relationship.
Output: T
retrieve meta-node N *(meta-action-set* $\cup$ $\{meta\}$, *)* from T;
if N is valid **then**
set N as (*meta-action-set* $\cup$ $\{meta\}$, * $\cup\{\mathrm{r}\}$);
set *effect(N) = effect(N) + num(new objects triggered by r)* $\cdot$ *conf(r)*;
if N is someone's parent **then**
update N's children's effect.
end if
else
set N as (*meta-action-set* $\cup$ $\{meta\}$, $\{r\}$);
set *effect(N) = num(distinct objects triggered by r)* $\cdot$ *conf(r)*;
add N into T;
attempt to retrieve N's parent meta-nodes P in T;
if P is not empty **then**
set every one of P as N's parent;
set *effect(N) = effect(N)* + $\sum_{i=1}^{|P|}$ *num(new objects from* P_i*)* $\cdot$ $conf(*)$;
end if
end if
return T;

action rule r is fulfilled. By r is fulfilled, we mean that the set of meta-actions in the new entry E' is all we need to trigger r according to advanced matrix M. Algorithm 3 is responsible for two tasks: constructing meta-nodes and building trees by linking meta-nodes regarding their relation, which requires relevant information including current meta-action *meta*, current action rule r, *meta-action-set* from E and of course T for storing meta-nodes. To construct a meta-node for a most recently fulfilled rule r, the first step is to check whether there is a meta-node in T which contains the same set of meta-actions as r does but fulfilling other rules. Since every meta-node can be seen as a mapping in *MetaMap*, the validity of a meta-node is evaluated in a similar way as we did for validating a mapping previously. As shown in Algorithm

3, if the meta-node N retrieved for the purpose of validation from T is non-empty, then r should be added into the action rule set along with other rules represented as * that have already been stored in N. Otherwise, it means that N is empty, so a new entry ($meta\text{-}action\text{-}set \cup \{meta\}$, $\{r\}$) should be assigned to N and added into T as a new meta-node. With a newborn meta-node N, the last but not least step is to set up the parent-child relationship through looking for its parent nodes. The parent nodes will not exist unless the mapping E' is extended from an existing mapping with one meta-action less in previous procedures in Algorithm 2. Therefore, as long as the meta-nodes retrieved by looking for meta-nodes each of which has a set of meta-actions with any one meta-action dropped from ($meta\text{-}action\text{-}set \cup \{meta\}$), they are N's parents. On the other hand, it is unnecessary to set up the relationship for a newly updated meta-node because its parent must be found when it was made.

Besides organizing sets of meta-nodes hierarchically as a tree, computing the expected effect of each meta-node during the process is another characteristic step in our strategy. It provides clients with concrete clues to evaluate the worth of those meta-actions. Since the effect of triggering a single action rule is defined as the product of its support and confidence, the effect of a meta-node is the summation of the effect of triggering all action rules in it, in another word, it is the product of the number of distinct objects without duplicate ones and their respective confidence from related action rules. So whenever a new action rule is activated and added into an existing meta-node, its effect must be assigned into the meta-node by tracking the newly triggered objects and their confidence. Additionally, if this existing meta-node has any children, its children's effect need to be updated with the same amount as its

parent does. And for a newborn meta-node, besides its own influence computed on the basis of our regulations, its effect is strengthened with each of its parent's help unless its parents do not exist. Similarly, the portion of effect inherited from parent meta-nodes that enhance current meta-node's performance should be caused by the number of objects from each parent node which are new to current meta-node and their corresponding confidence.

The size of a meta-node implies the number of meta-actions stored in it. Ultimately Algorithm 2 will sort the meta-nodes in T by their size ascendingly and their effect descendingly, and then return T after all the meta-actions in *meta-list* have been processed. An example of printed result will be shown in the next section.

10.4 Experiments

To test its performance, experiments are carried out with the implementation of the proposed algorithm in JAVA. Firstly, we take a small sample of data as an example to show the procedures in Algorithm 2 and the representation of final results. Table 13 is the advanced matrix in our example, and there are six action rules and four meta-actions involved in it. These four meta-actions are descendingly sorted by their popularity in *meta-list* as $\{M_3, M_4, M_2, M_1\}$ and they will be checked one by one in such order.

Hence, with M_3 as the most frequent meta-action in the list, all the action rules associated with it will be stored in an entry in *MetaMap* at the first round, which is $(\{M_3\}, \{r_1, r_2, r_3, r_4, r_6\})$. There is no action rule that has been fulfilled, so no meta-node will be made yet. When it comes to M_4, the action rules involving it are handled

Table 13: Advanced matrix for the sample data

	M_1	M_2	M_3	M_4
r_1	0	1	1	1
r_2	1	0	1	1
r_3	0	0	1	1
r_4	0	1	0	1
r_5	1	1	1	0
r_6	0	0	1	1

in different ways. Action rules that have already been stored in *MetaMap*, like r_1, r_2, r_3 and r_6, should be moved to another mapping with $\{M_3, M_4\}$, and the previous mapping becomes $(\{M_3\}, \{r_4\})$. For action rules that are new to *MetaMap*, a new entry is built and it is $(\{M_4\}, \{r_5\})$ in this example. In terms of building meta-nodes, when iterating action rules top down, we would find that r_3 is fulfilled with $\{M_3, M_4\}$. Obviously a meta-node $(\{M_3, M_4\}, \{r_3\})$ should be built and it parent who is suppose to be $(\{M_3\}, *)$ doesn't exist, so no parent is affiliated and its effect is $sup(r_3) \cdot conf(r_3)$. And the same meta-actions trigger r_6 as well, so the meta-node becomes $(\{M_3, M_4\}, \{r_3, r_6\})$, and its effect is updated as $sup(r_3) \cdot conf(r_3) + sup(r_6) \cdot conf(r_6)$. We should take care of its children's effect as well, but there isn't any, so nothing needs to be done. As all the action rules are appeared in *MetaMap* now, we focus on creating new mappings from existing ones and updating the existing mappings by following the requirements. Along with the same procedures being performed to the rest meta-actions, more meta-nodes will be built, such as $(\{M_3, M_4, M_2\}, \{r_1\})$, $(\{M_2, M_4\}, \{r_4\})$ and so on. Whenever a new meta-node is made, it is necessary to look for its parent. For instance, meta-node $(\{M_3, M_4, M_2\}, \{r_1\})$ is a mapping extended from $\{M_3, M_4\}$, and meta-node $(\{M_3, M_4\}, \{r_3, r_6\})$ does exist, thus these

```
meta-node0= (m3, m4)
its effect = 6.15
meta-node0's parent is not valid
meta-node0's children include: meta-node2(m3, m4, m2)
meta-node0's child's effect is 8.15
meta-node0's children include: meta-node3(m3, m4, m1)
meta-node0's child's effect is 7.95
meta-node1= (m4, m2)
its effect = 1.6
meta-node1's parent is not valid
meta-node1's children include: meta-node2(m3, m4, m2)
meta-node1's children's effect is 8.15
meta-node2= (m3, m4, m2)
its effect = 8.15
meta-node2's parent is meta-node0(m3, m4)
meta-node2's parent's effect is 6.15
meta-node2's parent is meta-node1(m4, m2)
meta-node2's parent's effect is 1.6
meta-node3= (m3, m4, m1)
its effect = 7.95
meta-node3's parent is meta-node0(m3, m4)
meta-node3's parent's effect is 6.15
meta-node4= (m3, m2, m1)
its effect = 4.35
meta-node4's parent is not valid
```

Figure 27: Experiment result with given example

two meta-nodes should be connected with a parent-child relationship as the former is a child of the latter. Meanwhile, the effect of the children node is potentially its own effect plus its parent's effect.

When the algorithm comes to the end, a list of meta-nodes will be printed as Figure 27 shows. There are five meta-nodes generated and they are listed in an descending order according to their effect in a group of nodes with same size, which is basically the priority that clients follow to judge them. In the group of meta-nodes involving two meta-actions, *meta-node0* has the greatest effect and it has two children with size 3 who are *meta-node2* and *meta-node3*. In the group of meta-nodes involving three

```
meta-node0= (ensure service done correctly)
its effect = 11.174404001042282
meta-node0's parent is not valid
meta-node0's children include: meta-node5(ensure service done correctly,
improve price competitiveness)
meta-node0's child's effect is 22.73196781593094
meta-node0's children include: meta-node6(ensure service done correctly,
nice technician)
meta-node0's child's effect is 21.948903984344167
meta-node0's children include: meta-node7(decrease dealer response time,
ensure service done correctly)
meta-node0's child's effect is 15.287128712871292
meta-node0's children include: meta-node9(ensure service done correctly,
service done correctly)
meta-node0's child's effect is 12.283870967741931
```

Figure 28: Partial experiment result with a larger sampling

meta-actions, *meta-node2* has the greatest effect 8.15, which is also the greatest effect among all generated meta-nodes.

Similar as running the program with the example, we run it for a larger sample data with 802 action rules generated and 5 meta-actions extracted, and the program sorted the result in a format as shown in Figure 27, which consists of 19 meta-nodes, their relations and their effect. Figure 28 shows the simplest meta-node that has the greatest effect. From Figure 28, it is clear that *meta-node0*'s effect is 11.17 and it have four children including *meta-node5*, *meta-node6*, *meta-node7* and *meta-node9*. Among its children, *meta-node5* has the most significant effect, which indicates that the effect will be doubled with one more meta-action *improve price competitiveness* applied, in contrast to applying *meta-node0* only. However, *meta-node6* is probably more preferred by clients in reality, because it has almost the same effect as *meta-node5*, and the second meta-action in *meta-node6 nice technician* is more applicable and acceptable by clients than *improve price competitiveness* in *meta-node5*. From

the example result shown in Figure 28, it is clear that the strategy helps clients select actions at top priority by considering the predicted effect as a partial reason for the final decision. What's more, the experiment with larger dataset proves the efficiency of organizing the meta-actions in the proposed way, because the number of generated meta-nodes is much less than the number of action rules or the possible subsets of meta-actions for triggering action rules. On the other hand, the effect of applying sets of meta-actions is calculated in a more accurate way with the definition of parent-children relationship. Therefore, this strategy absolutely provides a more effective way for clients to make the maximum profit out of the generated suggestions, and conclusively fits our anticipation.

CHAPTER 11: CONCLUSION

In this dissertation, a hierarchically structured recommender system driven by action rules and meta-actions has been designed to improve customers' satisfaction, in another word, improve clients' NPS scores by applying proper actions. During the process of building the system, we have confronted following issues: *1*) The given dataset contains human made mistakes and inconsistently formatted features with poor representation; *2*) Initial classification result is too weak for further analysis; *3*) It is useful to find similar clients and merge them together because they can learn from each other to improve their service; *4*) Suggestions to clients need to be built from the valuable information hidden in customers' reviews stored in text format; *5*) Selecting proper set of meta-actions is needed for better performance of applying them. To respectively address different problems, following procedures have been proposed and realized: *a*) Data preprocessing procedures involving data transformation, feature construction and feature selection have been conducted to result in better data representation and quality; *b*) Proper classification methods and experiments have been deployed to handle the issue of poor classification result, and experiments of mining action rules have been designed to prepare the settings for future purpose; *c*) Hierarchical structure has been built based on the mixture distance and used as the foundation of proposed hierarchical agglomerative algorithm HAMIS which follows a bottom-up path in the dendrogram and merges similar clients' datasets under certain

conditions. *d*) A new strategy for extracting meta-actions from customers' comments in text format has been designed and implemented. *e*) A method for boosting the performance of applying generated meta-actions has been developed and achieved satisfying results.

11.1 Initial Work for Data Analysis

To provide a well represented dataset with high quality and get ready for our future analysis, the original dataset has been preprocessed with three major steps:

1. Important features in various types have been transformed into consistent and more convenient formats with better representation, new features have been created to present extended information derived from existing ones, and the most representative features have been selected to build our dataset;

2. The classification algorithm in WEKA that achieved the best result in our domain has been selected. Based on that, construction of hierarchical features, normalization, discretization and addition of new features have been used to improve the classification performance.

3. To get prepared for mining action rules for each client in the future, initial experiments have been conducted and the setting for mining action rules has been determined.

Through the analysis work that is done afterwards, it is proved that the preparation work has built a firm foundation for our subsequent work, which makes it more efficient and smooth.

11.2 Hierarchical Agglomerative Method for Improving NPS

We know the fact that clients can learn from others who are similar and have better performance in NPS. To evaluate the similarity between clients, semantic similarity and geographical distance have been calculated. Semantic similarity measures the difference of knowledge hidden in datasets concerning *Promoter*, *Passive* and *Detractor* between clients. Beyond that, weight factors have been assigned to semantic similarity and geographical distance for balancing them and resulting in a mixture distance. Given the mixture distance between clients, an agglomerative clustering algorithm has been applied to generate a hierarchical clustering dendrogram, which forms the fundamental architecture built in our system.

Based on the hierarchical dendrogram, Hierarchical Agglomerative Method for Improving NPS (HAMIS) has been proposed to maximally enlarge a client's dataset by merging it with other clients who are clustered closely in dendrogram and have better performance in NPS and classification. Through our experiments on processing all clients with HAMIS, we conclude that more action rules with high confidence have been extracted from the extended dataset.

11.3 Extraction of Meta-Actions and Performance Boost

Action rules and meta-actions are used to advise clients for improving their performance, as the area of mining meta-actions is quite mature so far, but the method for extracting meta-actions is still under development, therefore, we mainly concentrate on designing strategies for mining meta-actions as the actual solutions returned to clients. To mine meta-actions from customers' comments in text format, a strategy

involving sentiment analysis, opinion summarization, feature identification and aggregation and most characteristically meta-action generation have been developed, and the experiments have been accomplished to prove its effectiveness and flexibility.

Besides generating meta-actions, providing clients with more concrete suggestions concerning the order, effect and applicability of executing them is also an important topic in our work. In our strategy of boosting the performance, we aims to create actionable paths that are well classified from groups of meta-actions and lead to certain increment of NPS rating, these actionable paths are a forest of trees where each tree consists of one or more meta-nodes which are hierarchically structured regarding the parent-child relationship between them. In the end, a list of meta-nodes containing sets of meta-actions will be returned and they are sorted ascendingly by the number of meta-actions in it and descendingly by their effect.

11.4 Future Work

From the perspective of applications, our recommender system has great potential and continuing our work in following aspects makes it more adaptive and feasible:

1. The extraction of meta-actions can be improved continually with no doubt. Currently, our strategy greatly relies on our knowledge about the domain and in natural language processing, as more learning process is achieved in the future, more opinions will be identified and extracted for generating meta-actions. On the other hand, regardless of the limitation on manually proposing meta-actions to respective features, there is great possibility that more meta-actions will be designed when new features are discovered.

2. Our dataset is continually growing over the years, and it is inevitable that new datasets will be collected and expected to show new insights on the most recent performance. However, current system is not capable of processing new datasets without loading it into the dendrogram. Constructing a new dendrogram whenever retrieving a new dataset is not an ideal solution due to low efficiency and time issue, so developing an advanced adaptive module to handle new datasets seems to be a better choice.

3. For the best usage of our system, building a program with intuitive interface which allows clients to use it in reality becomes a crucial part on our way of proving the effectiveness and applicability of our system, and the system is now under construction. To build our recommender system in an user-friendly way, we keep discussing with our partners from the company and modifying the frame as advised. Currently the system prototype has been completed and it will become more powerful with the newly embedded or enhanced modules.

REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions*, 17(6):734 – 749, 2005.

[2] M. Balabanovic and Y. Shoham. Fab: Content-Based, Collaborative Recommendation. *Communications of the ACM*, 40(3):66 – 72, 1997.

[3] C. Basu, H. Hirsh, W. Cohen, et al. Recommendation as classification: using social and content-based information in recommendation. In *AAAI/IAAI*, pages 714–720, 1998.

[4] J. Bazan. A comparison of dynamic and non-dynamic rough set methods for extracting laws from decision table. *Rough sets in knowledge discovery*, 1:321–365, 1998.

[5] J. G. Bazan, M. S. Szczuka, and J. Wroblewski. A new version of rough set exploration system. In *Rough Sets and Current Trends in Computing*, pages 397–404. Springer, 2002.

[6] D. Billsus and M. J. Pazzani. User modeling for adaptive news access. *User modeling and user-adapted interaction*, 10(2-3):147 – 180, 2000.

[7] S. Blair-Goldensohn, K. Hannan, R. McDonald, T. Neylon, G. A. Reis, and J. Reynar. Building a sentiment summarizer for local service reviews. In *WWW Workshop on NLP in the Information Explosion Era*, volume 14, 2008.

[8] L. Breiman. Random forests. *Machine learning*, 45(1):5 – 32, 2001.

[9] R. Burke. Knowledge-based recommender systems. *Encyclopedia of Library and Information Science*, 69, 2000.

[10] Y. H. Cho, J. K. Kim, and S. H. Kim. A personalized recommender system based on web usage mining and decision tree induction. *Expert Systems with Applications*, 23(3):329 – 342, 2002.

[11] M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, and M. S. D. Netes. Combining content-based and collaborative filters in an online newspaper. In *Proceedings of ACM SIGIR workshop on recommender systems*, volume 60. Citeseer, 1999.

[12] A. Dardzinska and Z. Ras. Extracting rules from incomplete decision systems. In *Foundations and Novel Approaches in Data Mining*, pages 143–153. Springer, 2006.

[13] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Machine learning: proceedings of the twelfth international conference*, volume 12, pages 194–202, 1995.

[14] U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Join Conference on Artificial Intelligence*, 1993.

[15] V. Ganti, R. Ramakrishnan, J. Gehrke, A. Powell, and J. French. Clustering large datasets in arbitrary metric spaces. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 502–511. IEEE, 1999.

[16] J. Grzymala-Busse. A new version of the rule induction system LERS. *Fundamenta Informaticae*, 31(1):27–39, 1997.

[17] R. Gutierrez-Osuna and H. T. Nagle. A method for evaluating data-preprocessing techniques for odor classification with an array of gas sensors. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 29(5):626 – 632, 1999.

[18] A. Hajja, Z. Ras, and A. Wieczorkowska. Hierarchical object-driven action rules. *Journal of Intelligent Information Systems*, 42(2):207 – 232, 2014.

[19] M. A. Hall. *Correlation-based feature selection for machine learning.* PhD thesis, The University of Waikato, 1999.

[20] M. A. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *Knowledge and Data Engineering, IEEE Transactions on*, 15(6):1437–1447, 2003.

[21] Z. He, X. Xu, S. Deng, and R. Ma. Mining action rules from scratch. *Expert Systems with Applications*, 29(3):691 – 699, 2005.

[22] W. Hill, L. Stead, M. Rosenstein, and G. Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 194–201. ACM Press/Addison-Wesley Publishing Co., 1995.

[23] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.

[24] M. Hu and B. Liu. Mining opinion features in customer reviews. In *AAAI*, volume 4, pages 755 – 760, 2004.

[25] R. Jensen and Q. Shen. Fuzzy-rough sets assisted attribute selection. *IEEE Transactions on Fuzzy Systems*, 15(1):73 – 89, 2007.

[26] S. B. Kotsiantis, D. Kanellopoulos, and P. Pintelas. Data preprocessing for supervised leaning. *International Journal of Computer Science*, 1(2):111 – 117, 2006.

[27] S. Krolak-Schwerdt, P. Orlik, and A. Kohler. A regression analytic modification of Wards method: A contribution to the relation between cluster analysis and factor analysis. In *Classification, Data Analysis, and Knowledge Organization*, pages 23–27. Springer, 1991.

[28] J. Kuang, A. Daniel, J. Johnston, and Z. Ras. Hierarchically structured recommender system for improving NPS for a company. In *Rough Sets and Current Trends in Computing*, pages 347–357. Springer, 2014.

[29] B. Liu. Sentiment analysis and subjectivity. *Handbook of natural language processing*, 2:627–666, 2010.

[30] B. Liu, M. Hu, and J. Cheng. Opinion observer: analyzing and comparing opinions on the web. In *Proceedings of the 14th international conference on World Wide Web*, pages 342–351. ACM, 2005.

[31] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2), 1993.

[32] M. D. Marneffe and C. Manning. Stanford typed dependencies manual. Technical report, Technical report, Stanford University, 2008.

[33] W. N. H. W. Mohamed, M. N. M. Salleh, and A. H. Omar. A comparative study of Reduced Error Pruning method in decision tree algorithms. In *Control System, Computing and Engineering (ICCSCE), 2012 IEEE International Conference on*, pages 392–397. IEEE, 2012.

[34] H. S. Nguyen and S. H. Nguyen. Discretization methods in data mining. *Rough sets in knowledge discovery*, 1:451 – 482, 1998.

[35] M. F. Othman and T. M. S. Yau. Comparison of different classification techniques using WEKA for breast cancer. In *3rd Kuala Lumpur International Conference on Biomedical Engineering 2006*, pages 520–523. Springer, 2007.

[36] R. Paul and A. Hoque. Mining irregular association rules based on action and non-action type data. In *Digital Information Management (ICDIM), 2010 Fifth International Conference on*, pages 63–68. IEEE, 2010.

[37] Z. Pawlak. Information systems theoretical foundations. *Information Systems*, 6(3):205 – 218, 1981.

[38] M. Pazzani. A Framework for Collaborative, Content-Based, and Demographic Filtering. *Artificial Intelligence Review*, 13(5-6):393 – 408, 1999.

[39] Y. Qiao, K. Zhong, H. Wang, and X. Li. Developing event-condition-action rules in real time active database. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 511–516. ACM, 2007.

[40] Z. Ras and A. Dardzinska. Action rules discovery based on tree classifiers and meta-actions. In *Foundations of Intelligent Systems*, pages 66–75. Springer, 2009.

[41] Z. Ras and A. Dardzinska. From data to classification rules and actions. *International Journal of Intelligent Systems*, 26(6):572–590, 2011.

[42] Z. Ras and A. Wieczorkowska. Action-rules: how to increase profit of a company. In *Principles of Data Mining and Knowledge Discovery*, pages 587–592. Springer, 2000.

[43] Z. Ras, E. Wyrzykowska, and H. Wasyluk. ARAS: Action rules discovery based on agglomerative strategy. In *Mining Complex Data*, pages 196–208. Springer, 2008.

[44] J. Rauch and M. Simunek. Action rules and the GUHA method: Preliminary considerations and results. In *Foundations of Intelligent Systems*, pages 76–87. Springer, 2009.

[45] F. Reichheld. The one number you need to grow. *Harvard Business Review*, 81(12):46–55, 2003.

[46] P. Resnick, N. Iakovou, M. Sushak, P. Bergstrom, and J. Riedl. GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[47] SATMETRIX. Improving your net promoter scores through strategic account management. *White paper*, 2012.

[48] A. Schein, A. Popescul, L. Ungar, and D. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002.

[49] B. Schowe. Feature selection for high-dimensional data with RapidMiner. In *Proceedings of the 2nd RapidMiner Community Meeting And Conference (RCOMM 2011), Aachen*, 2011.

[50] M. Shanker, M. Y. Hu, and M. S. Hung. Effect of data standardization on neural network training. *Omega*, 24(4):385 – 397, 1996.

[51] U. Shardanand and P. Maes. Social information filtering algorithms for automating "word of mouth". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217. ACM Press/Addison-Wesley Publishing Co., 1995.

[52] I. Soboroff and C. Nicholas. Combining content and collaboration in text filtering. In *Proceedings of the IJCAI*, volume 99, pages 86–91, 1999.

[53] G. Somprasertsri and P. Lalitrojwong. Mining feature-opinion in online customer reviews for opinion summarization. *J. UCS*, 16(6):938–955, 2010.

[54] V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan, and B. P. Feuston. Random forest: a classification and regression tool for compound classification and QSAR modeling. *Journal of Chemical Information and Computer Sciences*, 43(6):1947 – 1958, 2003.

[55] M. Tiwari, M. Jha, and O. Yadav. Performance analysis of data mining algorithms in Weka. *IOSR Journal of Computer Engineering*, 6(3):32 – 41, 2012.

[56] H. Touati, J. Kuang, and A. H. Z. W. Ras. Personalized action rules for side effects object grouping. *International Journal of Intelligence Science*, 3(1A):24 – 33, 2013.

[57] A. Tzacheva and Z. Ras. Association action rules and action paths triggered by meta-actions. In *Granular Computing (GrC), 2010 IEEE International Conference on*, pages 772–776. IEEE, 2010.

[58] K. Wang, Y. Jiang, and A. Tuzhilin. Mining actionable patterns by role models. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 16 – 16. IEEE, 2006.

[59] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *ICML*, volume 97, pages 412–420, 1997.

[60] H. Zhang, Y. Zhao, L. Cao, and C. Zhang. Combined association rule mining. In *Advances in Knowledge Discovery and Data Mining*, pages 1069–1074. Springer, 2008.

[61] L. Zhuang, F. Jing, and X. Y. Zhu. Movie review mining and summarization. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 43–50. ACM, 2006.