BIG DATA ANALYSIS

– A CASE STUDY

Du, Haoran
Supervisor: Dr. Wang, Qing

COMP 8790: Software Engineering Project

AUSTRALIAN NATIONAL UNIVERSITY
Semester 1, 2014
Acknowledgement

Many people have assisted me in carrying out this project. Firstly I would like to thank my academic supervisors Dr. Qing Wang for her ideas, support, patience, encouragement and feedback. I would also like to appreciate the course co-ordinator, Dr. Weifa Liang, for sharing his valuable experiences in writing report and giving presentations. In addition, I would also like to thank Elly who provided my project many helpful advice. Lastly, I would like to thank my family for their support and encouragement.
Abstract

Big data analysis attracts more and more attention from a wide range of fields nowadays. Although it can bring many benefits on how to obtain knowledge from data for support of decision making, still many challenges remain to be solved, such as ambiguities semantics, huge data volume, and noisy data. This report provides case study for big data analysis based on an industrial project with the Australian company Piction. In this report, I present the key techniques I used to develop a recommender system that can recommend products to customers. In order to efficiently handle high dimensional data, I have combined the Locality-Sensitive Hashing (LSH) method with fast hamming search algorithm. Then I conducted several experiments to evaluate the scalability and quality of the unified method. The experimented results show that this unified method performs efficiently with an accurate results. Finally, I develop a simple recommender system tool with user interface based on this method.
# Content

1. Introduction .......................................................................................... 1  
   1.1 Objective ......................................................................................... 1  
   1.2 Outline ............................................................................................ 1  
2. Project Backgroud ............................................................................... 3  
   2.1 Piction Company ............................................................................ 3  
   2.2 E-comm tool project ....................................................................... 4  
3. Methodology ......................................................................................... 6  
   3.1 Distance Measures .......................................................................... 6  
      3.1.1 Cosine distance ........................................................................ 6  
      3.1.2 Hamming distance .................................................................... 7  
   3.2 Locality sensitive hashing ................................................................ 7  
      3.2.1 LSH functions ......................................................................... 8  
      3.2.2 LSH for cosine distance ............................................................ 10  
      3.2.3 Discussion ............................................................................... 14  
      3.2.4 Algorithm details ..................................................................... 17  
   3.3 Fast hamming search ....................................................................... 17  
      3.3.1 Algorithm details ..................................................................... 17  
      3.3.2 Discussion ............................................................................... 19  
   3.4 Complexity analysis ....................................................................... 20  
      3.4.1 Time Complexity ..................................................................... 20  
      3.4.2 Space Complexity .................................................................... 21  
4. Implementation ...................................................................................... 23  
   4.1 Data set ........................................................................................... 23  
   4.2 Data pre-processing ......................................................................... 25  
      4.2.1 File segmentation .................................................................... 25
4.2.2 Database import ................................................................. 25
4.2.3 Target investigation ............................................................. 26

4.3 Core algorithm implementation ................................................. 27
  4.3.1 Measurements ................................................................. 27
  4.3.2 Initial value preset ............................................................. 28
  4.3.3 Experiment .......................................................................... 29

4.4 A simple tool ............................................................................ 30

5. Conclusion and Future work ......................................................... 32
  5.1 Conclusion ............................................................................. 32
  5.2 Future work ........................................................................... 32
Chapter 1

1. Introduction

Big data refers to technologies and initiatives that involve data that is too diverse, fast-changing or massive for conventional technologies, skills and infrastructure to address efficiently (Zikopoulos, Paul, and Chris Eaton, 2011). A wide variety of techniques have been developed to visualize, analyze, manipulate and aggregate Big data. These techniques range over many areas, such as Database and data warehouse technologies, machine learning and artificial intelligence, statistics and numerical mathematics. From an enterprise perspective, Big data has the potential to help companies improve operations and make faster, more intelligent decisions. Many companies have applied Big data techniques to market themselves better. For example, Amazon has used Big data analysis tools to develop marketing solutions for optimizing customer service. However, there are many challenges in using Big data analysis because Big Data is often noisy, dynamic, heterogeneous, inter-related and untrustworthy. Thus, traditional data process methods do not work.

Recently, an Australian company named Piction launched the e-comm tool project, which aims to obtain beneficial advise on their commercial decision via Big data analysis. I undertake this project as my master project. This report will give a detailed description about how I address the challenges of noisy data and the trouble of big data volume in the development of a recommender system tool.

1.1 Objective

The initial requirements of this project are mainly two aspects. One is to analyze the data set from Piction pre-process. The task is to investigate what kind of useful knowledge can be extracted from the data set. The other is to build a useful tool so that knowledge we discovered in the project can be used in practice. After analyzing data, I set the objective of this project on developing a recommender system tool that can recommend products to customers. The desired requirements for such a tool are effectiveness and efficiency.

1.2 Outline

The rest of the report is organized as follows:
• In Chapter 2, I briefly introduce the company Piction and describe the general background.

• In Chapter 3, I introduce the main techniques used in developing the recommender system tool in this project.

• In Chapter 4, I describe the characteristics of the data set given by Piction, and discuss how to pre-process the noisy dataset, how to extract useful data to be analysed, how to choose the algorithms, and how to conduct the experiments to verify the efficiency and quality of the tool.

• Finally, I conclude the report and discuss some possible extensions in the future in Chapter 5.
Chapter 2

2. Project Background

This project is a joint project of ANU and an industrial company Piction. This chapter will provide a general background of this project.

2.1 Piction Company

Piction is an Australian company which supplies software solutions within the areas of Digital Asset Management, E-Business, Ticketing and Venue/Educational booking systems. It has a strong focus on the museum, government and corporate sectors. The most well-known customers of it include Australian War Memorial, and Australian National Botanic Gardens.

Piction company has developed an award winning platform called The Piction Digital Media eXchange (DMX). This platform has been applied to satisfy different requirements for different sector producers. The software solutions based on this platform allow organizations to centralize, control, distribute and commercialize a huge volume of digital data such as images, PDFs, audio, video and so on. This platform has features such as fast, scalable, and rich-domain-oriented. The following graph is the homepage about this company.
2.2 E-comm tool project

As the volume of customer data in Piction is incredibly increasing, also as a response to the challenge of big data, this E-comm project is launched. The purpose of this project is to build a processing tool that can extract data from the Piction e-comm system and generate analysis report of user purchase behaviours. As shown in Graph 2, this project consists of five stages.

- In the first stage, the Piction Server runs on Oracle which contains a wide variety of data.
- In the second stage, the extraction process will extract information from the e-commerce system and store the data as XML file in a file directory.
- In the third stage, the XML files will be ziped or unzipped, all of which satisfy the schema recorded in the Piction financial tables.
- In the fourth stage, a tool can tackle with XML files to support the analysis on the data record in those files.
- In the last stage, multiple results are expected to obtain. For example, visual analysis of the data which can be interpreted by management, data mining analysis looking for patterns which can supply the evidence for: recommendations for selling products better, selling statistics and usage, security incursions or holes and so forth.

The stages A, B and C are conducted at Piction and my main tasks in this project are stages D and E.
Graph 2
Chapter 3

3. Methodology

In this chapter, I first present the definition of cosine distance, which is the basic metric used in this project. Next, I briefly introduce the basic theory of locality-sensitive hashing, and discuss how I apply this theory to the data in this project with cosine distance in a high dimensional space. Moreover, I describe the algorithm of fast hamming search. Lastly, I discuss the complexity of combination of LSH and fast hamming search in aspects of time and space.

3.1 Distance Measures

3.1.1 Cosine distance

Assume that we have a space that contains a collection of points. If there is a function $d(x, y)$ on this space, which takes two points in this space and generates a real number, satisfying the following conditions, we call such a function a distance measure on this space:

1. $d(x, y) \geq 0$, and $d(x, y) = 0$ if and only if $x = y$ (distance is positive between two different points and is zero from a point to itself).

2. $d(x, y) = d(y, x)$ (distance is the same from either direction, namely distance is symmetric).

3. $d(x, y) \leq d(x, z) + d(z, y)$ (the distance between two points is the shortest distance along any other path, which is called triangle inequality).

Then, the cosine distance between two points is defined as the angle between the corresponding vectors representing those points. The cosine of the angle between the two points is defined as the cosine similarity between them.

It is easy to prove that the cosine distance is indeed an distance measure. Firstly, the range of the angle will be from 0 to 180 degree, in any dimensional space, so there is no negative value. Besides, the angle between two vectors is 0 if and only if the two vectors are in the same direction. Secondly, the angle from vector one to vector two is the same as the angle from vector two to vector one. Lastly, for physical reasoning, the angle displacement moving from vector one directly to vector two cannot be more than the angle displacement from vector one to move to vector three and thence to vector two (Rajaraman, Anand, and Jeffrey, 2012).
To calculate the cosine distance between two points, we first compute the cosine of the angle, and then use the arccosine function to translate the cosine value to an angle valuing between 0 and 180. This is also the reason why this distance is called cosine distance. Recall that the cosine of the angle between two vectors is the dot product of two vectors divided by the L2-norms of those two vectors (Roberts, Gareth, and Richard, 2001).

**Example**: Assume we have two points, \( x = [0,2,2] \) and \( y = [2,0,-2] \). The dot product of \( x \) and \( y \) is \( x \cdot y = 0 \times 2 + 2 \times 0 + 2 \times -2 = -4 \). The L2-norms of \( x \) is \( \sqrt{(0)^2 + (2)^2 + (2)^2} = 2\sqrt{2} \) and the L2-norms of \( y \) is \( \sqrt{(2)^2 + (0)^2 + (-2)^2} = 2\sqrt{2} \). So we can get the cosine of the angle between \( x \) and \( y \) is \( 4/(2\sqrt{2} \times 2\sqrt{2}) = 0.5 \). By applying the arc-cosine function, the angle whose cosine is 0.5 is 60 degrees. Thus, the cosine distance between \( x \) and \( y \) is 60. So, the cosine similarity is 0.5.

### 3.1.2 Hamming distance

In a space, the points can be represented as a vector with coordinates. The **hamming distance** between two points is defined as the number of coordinates, that differ from each other. Hamming distance is also a distance measurement since it follows the three conditions of distance measurement. Specifically, the minimum value of hamming distance is 0, if and only if the two points are identical. It is symmetric because the distance does not depend on which point we take first. The difference between two points cannot be bigger than the sum of respective difference between those two points and the third point, so the triangle inequality is satisfied.

**Example**: Assume we have two points, \( x = [1,0,1] \) and \( y = [0,1,0] \). The hamming distance between \( x \) and \( y \) is 3. If \( xx = [1,0,1] \) and \( y = [1,1,0] \), then the hamming distance between \( x \) and \( y \) is 1.

Usually, the hamming distance is used with binary vectors, i.e., the coordinates of which are either 0 or 1. The calculation of the hamming distance of binary vectors is extremely fast. So, this kind of distance is applied in many area to enhance efficiency.

### 3.2 Locality sensitive hashing

Assumed we have \( n \) data points in a database. Given a data point, our task is to find those data points which are most similar to the given data point. The naive way is to compare each pair of points to find the points which satisfy the condition. However, the first problem we may confront is that the space needed for storing all the pairs to be compared is also not affordable to the main memory. A possible solution is that we compress the huge set by a much smaller set without losing the expected similarity of any pair in the database. This representation of the original set is called **signature**. Furthermore, regardless of the space issue, finding the pairs with certain similarity efficiently may still be computational expensive. This is because \( n \) can be quite large. Nevertheless, often we just attempt to get the most similar
pairs or get the pairs which have the similarity above certain lower bound. Thus, we should just focus on the pairs that have high possibility to be similar, other than compare all points. Locality-sensitive hashing (LSH) is a general theory to provide such kinds of idea.

3.2.1 LSH functions

The notion of LSH is first introduced by Indyk and Motwani (1998) to solve nearest neighbour search problem. The basic idea of LSH-based approaches is to apply LSH functions. Those functions map close items, which are represented in a d-dimensional space, into the same hash bucket with high probability. In other words, the similar objects are more likely to have the same hash value than the dissimilar ones. There are mainly two steps to process similarity search on a LSH.

- Firstly, using LSH functions to build hash tables in which related points are allocated in the same bucket. Any pair in the same bucket is considered to be a candidate pair.

- Next is the query time. The k-nearest neighbour search is performed by hashing the query point to one bucket per hash table. All the objects in those discovered buckets will be ranked by their distance to the query point. The closest k objects are outputted as the final result.

**Definition:** Let $d_1 < d_2$ to be two distances under some distance measure $d$. LSH function family, i.e. $F=\{f : S \rightarrow U\}$ is called $(d_1, d_2, p_1, p_2)$-sensitive for $d$ if for every $f$ in $F$, any $x, y \in S$:

1. If $d(x, y) \leq d_1$, then the probability that $\Pr\{f(x) = f(y)\}$ is at least $p_1$.

2. If $d(x, y) \geq d_2$, then the probability that $\Pr\{f(x) = f(y)\}$ is at most $p_2$.

$\Pr\{f(x) = f(y)\}$ stand for the probability of that $x$ and $y$ would be candidate pairs. $S$ is the domain of the objects. $U$ is the domain of signature generated by corresponding hash function.

We prefer $p_1 > p_2$. Informally, this definition can be explained as follows: the nearby items within distance $d_1$ have a greater chance of being hashed to the same bucket than distant items those are at a distance greater than $d_2$. Figure 1 illustrates the behaviour of a $(d_1, d_2, p_1, p_2)$-sensitive family about how likely two items can be a candidate pair.
For consistency, we can transform the horizontal axis metrics from distance to similarity. Because the more the distance, the less the similarity, we can easily flip the x axis horizontally as shown in the figure 2.

Then, it would be more natural if we rotate the whole graph 180 degrees centring on y axis. As Figure 3 illustrated, we can rewrite the feature of LSH functions family from the view of similarity. The items which have similarity above s1 have more probability to be declared by LSH functions as candidate items than those whose similarity below s2. If we let s1 and s2 close enough to be regarded as a value s approximately, we can state that the pairs with similarity above s are very probable to become candidates, meanwhile those
with similarity below $s$ are unlikely to become candidates. The curve forms an “S” shape. This is the iconic sign for the LSH. We also call the value $s$ **threshold**. The threshold is roughly where the rise is steepest in the $s$-curve, which ensure nearby items have high probability to be allocated in the same bucket.

![Figure 3](image)

Different distance functions, such as the Jaccard distance and Hamming distance, can create different LSH families. However, the essential part in LSH technique is to obtain an appropriate LSH function family.

### 3.2.2 LSH for cosine distance

Recall that we can express an object by a point in a data space. The cosine distance of two points is the angle between the two corresponding vectors while the cosine similarity is the cosine of the cosine distance. For instance, we see in Graph 3 two vectors $u$, $v$ make an angle $a$ between them. We note that the two vectors may be in a space of many dimensions, however, they can determine a plane. $31$ is a top-view of the plane defined by $u$ and $v$. 
Suppose we take any vector from this space. It will always have a projection on any plane in the space. Specially, if the vector is on the space, the projection is itself. In Graph 4, the vector $X$ is projected into the plane containing vector $u$ and $v$, whose projection is represented by the dot vector $x$.

Graph 3

We can simply classify all the vectors in the space by this method. One class of vectors are those whose projection on the $u$-$v$ plane can separate the two vectors $u$ and $v$ on different sides, which is the blue area of the following Graph 5. At the same time, the other class of vectors, those whose projection cannot are located in the red area. First, we consider the first class of random vectors, that is the dot product of the vector with $u$ will have different sign to the dot product with $v$. As the vector $X$ shown in the 5, the dot product $X \cdot u$ is positive while the dot product $X \cdot v$ is negative. If we assume the projection of vector $X$ extends in the opposite direction, then $X \cdot v$ is positive while $X \cdot u$ is negative. However, those signs are still different. On the other hand, if the vectors $u$ and $v$ are located in the same side of the projection of the vector selected randomly, both the dot product of the vector with $u$ and the dot product with $v$ will have the same sign. Taken random vector $Y$ in the
for instance, the dot products \( Y \cdot v \) and \( Yu \) are both negative. In the opposite direction, \( Y \cdot v \) and \( Y \cdot u \) will be both positive.

Graph 5

All the locations for the projection of a randomly chosen vector is equally likely. Thus, the probability that a random vector separates two vectors \( u \) and \( v \) (i.e. the vector in first class) is directly proportional to the angle between the two vectors, namely \( 2a/360 = a/180 \). Similarly, the probability that we choose the second-class random vector will be \( (180-a)/180 \).

Intuitively, if we take a vector randomly and the two vectors are located just the same side of this random vector, then those two vectors have some probability to be the same vector. Therefore, we build every function \( f \) in our locality-sensitive family from a randomly chosen vector \( r \). Given two vectors \( u \) and \( v \), \( f(u) = f(v) \), if and only if \( u \) and \( v \) are at the same side of \( r \). This can also be interpreted that the dot product \( r \cdot u \) and \( r \cdot v \) have the same sign. From previous analysis, we know that the probability that \( r \) is the second class vector in the \( u-v \) plane is \( (180-d)/180 \), \( d \) is the angle between \( u \) and \( v \). In summary, the probability that \( u \) and \( v \) can be declared as a candidate is \( (180-d)/180 \). We assign \( d \) using value \( d1 \) and \( d2 \), where \( d1 < d2 \). Then the corresponding probability is \( (180-d1)/180 \), \( (180-d2)/180 \), where \( (180-d1)/180 > (180-d2)/180 \).

Hence, \( F \) is a \( (d1, d2, (180-d1)/180, (180-d2)/180) \)-sensitive family of hash function for cosine distance. Charikar (2002) proposed this kind of LSH function based on the following theorem:

**Theorem:** Suppose we are given a collection of vectors in a \( k \) dimensional vector space (as written as \( \mathbb{R}^k \)). Choose a family of hash functions as follows: Generate a spherically symmetric random vector \( r \) of unit length from this \( k \) dimensional space. We define a hash function, \( h_r \), as:
Then for vectors \( u \) and \( v \),

\[
Pr[h_r(u)=h_r(v)] = 1 - \frac{\theta(u,v)}{\pi}
\]

(1)

\( Pr[h_r(u)=h_r(v)] \) means the probability that both of the dot product of \( u \) with \( r \) and \( v \) with \( r \) are non-negative or negative. In this situation, \( u \) and \( v \) can be hashed into the same bucket to be declared as a candidate. This is identical to the probability that \( u \) and \( v \) are on the same side of the random vector \( r \), which is \( 1 - \frac{\theta(u,v)}{\pi} \). The rigorous mathematical proof of this theorem was provided by Goemans and Williamson (1995).

From Equation (1) we have,

\[
\cos(\theta(u,v)) = \cos((1-Pr[h_r(u)=h_r(v)])\pi)
\]

(2)

This gives us a way of calculating the cosine similarity. This equation is based on probability. Thus, the more random vectors we use, the more accurate cosine similarity between \( u \) and \( v \) we can obtain. Hence, we can use a large number of random vectors and then calculate the hash signatures with all the random vectors for \( u \) and \( v \). Applying Equation (2), we can estimate the cosine similarity between \( u \) and \( v \).

**Example**: Suppose that in a 4-dimensional space, we have three random vectors: \( v_1=[1,0,-1,0] \), \( v_2=[1,1,0,0] \), \( v_3=[0,0,-1,1] \). For \( x=[-1,1,0,1] \), the signature will be \([0,1,1]\). That is \( v_1 \cdot x = -1+0+0+0=-1 \). So, the first component is 0. Similarly, \( v_2 \cdot x =0 \) and \( v_3 \cdot x =1 \), the second and third signature are thus both 1. Consider another vector \( y=[1,2,3,4] \). We can obtain that the signatures of \( y \) is \([0,1,1]\) in the same way. Comparing the signatures of \( x \) and \( y \), we observe that they are identical. Hence, we estimate the angle between them is 0.

However, the conclusion is not accurate. We can calculate the cosine of the angle between \( x \) and \( y \) to be \( x \cdot y \), which is \(-1\times1+1\times2+0\times3+1\times4=5\) divided by the magnitudes of the two vectors, which are

\[
\sqrt{(-1)^2 + (1)^2 + (0)^2 + (1)^2} = \sqrt{3}
\]

and

\[
\sqrt{(1)^2 + (2)^2 + (3)^2 + (4)^2} = \sqrt{30}
\]

Thus the cosine of the angle between \( x \) and \( y \) is \( 5/\sqrt{30} \times \sqrt{3} = 0.527 \), and this angle is about 58.2 degrees. If we take more vectors, the estimation will become more accurate.
3.2.3 Discussion

In common case, we always focus on a range of the cosine similarity, especially above some certain value, other than some specific value of the cosine similarity. Hence, we can directly estimate exalt the value of the cosine similarity by comparing the signature and choose those satisfying conditions. Anyhow, we can have a better method, in which we set a proper threshold for the maximum number of different components of two points to be compared. We directly choose the points which is under the threshold. The process just contain the behaviour of comparison which is not like the original method in which we choose the points after calculating the estimation of the cosine similarity. Later, we will analyse this method is indeed effective and efficient to obtain neighbours for a query vector.

Denote cosine distance between two vectors is \( \theta \) and cosine similarity is \( c \). Let the \( p \) stand for \( \Pr[h_r(u)=h_r(v)] \). We choose \( d \) random vectors to get a \( d \)-bit binary signature vector for each vector to be compared. Suppose two vector can be declared as candidate with the hamming distance between two vectors is less than \( b \). Then, we can calculate the probability that two vectors would become a candidate pair. For every bit, the signatures will be identical otherwise different. So in total of \( d \) bits the event that there are \( b \) bits different follows binomial distribution where \( b \) successes (in here, it is more proper called fails) occur in sequence of \( d \) trials. We can get the probability of this event by the probability mass function \( C_d^b \cdot (1-p)^b \cdot p^{(d-b)} \). This relationship is shown by the blue line in the following Graph 6. Then we can just simply sum up all the probabilities to get the probability that there are no more than \( b \) bits different, which is also the probability that two vectors would become a candidate pair. According to the cumulative distribution function of binomial distribution, we get \( \sum_{n=0}^{b} C_d^b \cdot (1-p)^b \cdot p^{(d-b)} \).

![Graph 6](image)

It is obvious that the function of the probability to be a candidate forms an s-shape curve, as illustrated by the red line in Graph 4. We choose the mean of the binomial distribution \( \text{d}(1-p) \) as the threshold of \( b \), which is the peak of blue line around value 11, and also the part of red line rising most steeply in the graph. Holding the value of 1-\( p \) and \( d \) unchanged, if we just choose \( b \)
above the value $d(1-p)$, then the pairs with number of bits different form each other no more than $b$ are very likely to be candidate pairs.

But can we get the result that pairs with similarity above certain value can have high probability to become candidate? Firstly, we know that in the Graph 7 which demonstrates the cumulative function of binomial distribution, the curve will move back when the $(1-p)$ (in graph denoted $p$) get smaller while keeping the $b$ and $d$ the same. For instance, in the Graph 7, the blue line have the same $b$ and $d$ ($N$ in the graph) to the green line. But, the $(1-p)$ of blue line is 0.5 which is smaller than the $(1-p)$ of green line, which is 0.7. As demonstrated, the blue line is in the left of green line. This means that if we choose 0.7N to be the value of $b$, this is above the threshold of any curve of which the $(1-p)$ is smaller than $(1-p) = 0.7$, taking the blue line whose $(1-p)$ is 0.5 for example, then all the pairs which have the real probability $(1-p)$ under 0.7 will be very likely to be declared as candidate pairs. By the way, we should make a note that $(1-p)$ under 0.7 means that $\frac{\theta(u,v)}{\pi}$ is under 0.7, namely cosine distance $\theta(u,v)$is under 0.7$\pi$. Because cosine function is a decreasing function, cosine similarity is above $\cos(0.7\pi)$. Then we can answer the question above that as long as we have chosen proper $d$ and $b$, pairs with similarity above certain value can have high probability to become candidate. This is exactly the situation we want. Hence, this method is effective to obtain neighbours for a query vector.

Furthermore, we analyse this method from the view of maximum likelihood. Recall the fundamental basic idea of likelihood is using a bundle of random sample from certain distribution to estimate the unknown parameters of this distribution, the estimation of which can make the sample likelihood, the product of individual probabilities of the sample, maximum. Also we can say this estimation is the most probable parameter under the condition with the occurrence of these samples.

For instance, in our cosine LSH functions, whether the signatures of two vectors, generated by a same random vector, are equal follows the
Bernoulli distribution, denoted as $B(1, p)$, where the $p$ stands for $\Pr[\text{hr}(u) = \text{hr}(v)]$. The $p$ is the parameter of the mean of the Bernoulli distribution, which is unknown, which is also what we want to estimate. We take $d$ random vectors to get $d$-bit signature vectors, namely, $d$ samples. Moreover, we just regard those vectors who have no more than $b$ bits different between each other as candidate pairs. This means the candidate pairs we chosen are under the condition that there are more than $(d-b)$ same bits in $d$. As mentioned, the maximum likelihood estimator can be calculated in the general formula $t / n$ for any sequence of $n$ Bernoulli trials resulting in $t$ 'successes'. So, the estimators of $p$ of all the candidate are bigger than $(d-b)/n$. From the equation (2), we know the higher the $p$, the higher the cosine similarity. Thus, we get the result that all the candidates we chosen have high probability to have a similarity above certain value. Thus, this method is efficient to obtain neighbours for a query vector.

In conclusion, this method is not only effective but also efficient to quality of the near neighbour search result. This method can also reduce the dimensionality, where we reduce the original dimension of vectors to that of signature vectors, while holding the cosine distance. Besides, based on the equation (2), when we have a large number of random vectors, we just use the different number of bits between two points to estimate the probability that those two would be different, then calculate the cosine distance. As we known, finding the different number of bits between two points is just finding the hamming distance between the two points. So, this method converts calculating cosine distance to calculating binary hamming distance. As mentioned above, finding binary hamming distance is very efficient both in time and space. Also, there are abundant fast algorithms to address hamming distance finding, which enhance the efficiency further. In this report, we will introduce one of them, which is described below.

Digressively, two point should be mentioned. Firstly, because the binomial distribution is discrete distribution, we cannot get a continuous probability. Then, the accuracy of our selection will be affected. To smooth the $s$-curve, we always choose $d \geq 20$, $b \geq 5$ and $d-b \geq 5$, which is derived from the condition that the binomial distribution can be approximated as Gaussian distribution. Secondly, we can observe that the curve will move forward as the $N$ becomes bigger, keeping the $p$ and $b$ unchanged, just as the location of blue curve and red curve in the Graph 7. When the $N$ becomes bigger, to the same threshold $b$, the probability to be declared as candidate of the same $p$ will decrease, some situation very severely. This will results that the pairs those should be declared as candidates are not be chosen. Only the pairs with higher similarity keep the high probability to be selected. So, as we take more random vectors for analysis, there will be less pairs to be chosen and the result will be stricter. But the candidate pairs will have higher similarity. In other words, the accuracy will be enhanced. If we want to keep the probability of original pairs to be selected, we can choose a bigger $b$. As a summary, both $b$ and $d$ are adjustment parameter for the accuracy or strictness of the result. The bigger the $d$ or the smaller the $b$, the more accurate the result and vice versa.
3.2.4 Algorithm details

1. Initially, we have n vectors in k dimensions. Each of vector represents a customer and each of the k coordinates in the vector records the number of corresponding product that the customer bought. For convenience, the n vectors can be also regarded as a whole matrix.

2. In this step, I introduce how to generate random matrix which consist of d random vectors. Every random vector also has k dimensions, which is exactly the same to the vectors we got in the step 1. As an effective method, for one random vector, we can sample the standard Gaussian distribution with mean of 0 and variance of 1, by k times independently. Each of these k numbers is used to assign one of the k dimensions of the random vector. By repeating d times, we finish assign all the d random vectors and get the random matrix finally. As a note, the process of sampling Gaussian distribution can be achieved by the Box-Muller transformation (Box and Muller, 1958) or the algorithm detail can be found in The Art of Computer Programming (Knuth, 2005).

3. In this step, we attempt to get a signature matrix for the n vectors in step 1. For every vector, there will be a corresponding d-dimensional signature vector. So, there are n signature vectors. Those n signature vectors compose the signature matrix we attempt to obtain. We call one dimension in the signature vector a signature. Each signature of all original vectors can be determined by the hash function. Specifically, according to the definition of the hash function \( h_r(u) \) in subsection 3.2.2, we can get a signature of the vector \( u \) by calculating the scalar product of the random vector \( r \) and the vector \( u \). If the product is negative, the signature is assigned 0. Otherwise, the signature is 1. We traverse all the d random vectors for one vector and get d signatures for it. This is the reason why the signature vector is d-dimensional. Then, after finishing the signature vector generation for every vector, we can get the signature matrix.

3.3 Fast hamming search

To calculate fast hamming distance, we randomly permute the components of all the vectors and sort the new vectors to find the close neighbours of certain vector. In the sorting of vectors, we regard vectors as binary numbers and sort them according the numerical order. This search algorithm is called Point Location in Equal Balls (PLEB), which is first published by Indyk and Motwani (1998), and further developed by Charikar (2002) to solve the problem of nearest neighbours search for a given vector. We modify the Charikar's version to suite our problem to find the top B nearest neighbour of the query point. This report will skip the mathematics explanation of this method and just describe the process detail.

3.3.1 Algorithm details
1. I use the signature matrix in subsection 3.2.4, which is a binary matrix, as input of this algorithm. To efficiently calculate the hamming distance between signature vectors, we can also represent the signature vector in the form of bit streams of length d. Consequently, the signature matrix can be regarded as n parallel bit streams. Firstly, we shuffle the original bit index of the bit streams and get a random permutation of it. For every signature vector in the matrix, we jumble signatures by the order of the permuted bit index. This process is indicated in the Figure 4. As a result, we obtain a permuted signature matrix. By choose q different permutations of the original bit index, there will be q different permuted signature matrix.

<table>
<thead>
<tr>
<th>Original index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit stream</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Permuted index</th>
<th>1</th>
<th>0</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit stream</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 4

2. For each permuted signature matrix, we sort every permuted signature vector lexicographically. In other words, every vector in the permuted signature matrix can be addressed as a binary number. We convert the binary number in decimal base and define the decimal number as the value of the vector. Then we sort the vectors by the ascending order of their value in every permuted signature matrix and finally get q sorted signature matrix.

**Example:** Table 1 and 2 show how to sort group of binary vectors lexicographically. To be simple, we take a matrix which consists of three vectors. Table 1 shows the matrix before sorting and the value of each vector while table 2 is the result of sorting.

<table>
<thead>
<tr>
<th>Vector 0</th>
<th>Vector 1</th>
<th>Vector 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Value</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1

<table>
<thead>
<tr>
<th>Vector 0</th>
<th>Vector 1</th>
<th>Vector 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Value</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 2

3. In this step, we introduce two new concepts, window and bucket. Window is used like filter. Its centre is always the query vector. The radius of the window determines how many vectors to be chosen in either direction,
backward or forward, from the query vector. For instance, if we just set the radius of window on 2, the previous two vectors and the following two vectors of the query vector will be filtered out. So we can get the closest four neighbour vector from the query vector. All the selected vectors would be put in the bucket. The Graph 8 below demonstrates the details of the given example. However, this graph gives a special case, in which the query vector is the second last of the matrix. As a result, the window can only filter out three vectors other than four as expected. One thing should also be noticed is that the bucket has the feature of set. This means that if we put a same vector into an empty bucket two times, there is still only one vector in the bucket. For every query vector, we select all the vectors filtered by the window with the radius B in every sorted matrix and put them in one bucket. Then, we check the hamming distance of the query vector with all the vectors in the bucket. If the distance is above the predetermined threshold b, normally B=b, we abort this vector. After analysing all the vectors in the bucket, we eventually get all the candidate vectors.

![Graph 8](image)

### 3.3.2 Discussion

We can have a more detailed view on the fast hamming search. We have already know that the whole algorithm is that we permute the original matrix and sort all the vectors in one matrix. Then, we just choose the nearest neighbours directly from the index without any calculation about the distance.
Actually, the effect of sorting for binary can put the vectors which have small distance from each other together. For example, we can look at all the binary vectors at length 3 in Table 3.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3

Those vectors have been sorted. We observe that the neighbours often have small hamming distance. Nevertheless, we can also see that some cases are not so expected. For example, [1,0,0] and [0,1,1], although they are adjacent, all the coordinates are different. This is the reason why we permute the vectors. First, we should keep in mind that under the same permutation, the hamming distance between the same two points will stay the same. Then, by permutation, we enhance the probability that vectors with small hamming distance will be put together while reduce the probability that the vectors with big hamming distance to be put together. Consider [1,0,0] and [0,1,1] in Table 3, if we invert them from the end to start, we can get [0,0,1] and [1,1,0] and they will be far away. Similarly, if we take [0,0,1] and [1,0,1], they are quite far away but have small distance. If we permute them inversely and get [1,0,0], [1,0,1], they will be located adjacently. In step 3, we choose the near neighbour just according to the location rather than the hamming distance. This is because the hamming distance between adjacent vectors is at least 1. By the triangle inequality, the b nearest neighbour chosen by location can ensure the distance from all vectors being chosen to the query vector will be no more than b.

3.4 Complexity analysis

In this section, we give some complexity analysis for LSH and fast search from the aspects of time and space.

3.4.1 Time Complexity

We first discuss the time complexity of the combination of LSH and fast hamming search.

- In the step 2 of LSH, we build d random vectors, the dimension of which are all k. So, we generate total of k*d random numbers and the complexity is O(d*k).
- In step 3 of LSH, we calculate the signature vector for n original vectors with d random vectors. Every component of a signature vector is the dot
product of the k-dimensional original and random vectors, which is obtained after k calculations. So the complexity is O(k*n*d).

- In step 2 of fast hamming search, we acquire q permuted random matrices. For each permuted random matrix, it takes O(n*d). So, the complexity is O(q*n*d).

- In step 1 of fast hamming search, in each permuted random matrix, we sort each binary vector according to their decimal value. The most quick method will take O(n*logn). Thus, the complexity of step 5 is O(q*n*logn).

- In step 3 of fast hamming search, we just choose the 2b nearest neighbour for the query vector in each sorted matrix. However, we should find the query vector first. If we choose sequential search algorithm, the complexity is O(n). Then we select 2b neighbours, which is O(2b). So, the complexity of step 6 is O(q(2b+n)).

By summing up all the steps, the complexity of the combination of LSH and fast hamming search is O(d*k+k*n*d+q*n*d+q*n*logn+ q(2b+n)). We know that d<<k. Besides, q and b are quite small. So, the main part of complexity is O(k*n+ n*logn). Generally, logn << k. This implies n*logn << n*k. So, the complexity of this algorithm is approximately O(k*n).

3.4.2 Space Complexity

Now, we discuss the space complexity of the combination of LSH and fast hamming search.

- In the step 2 of LSH, we should store d k-dimensional random vectors. So, the space complexity is O(d*k).

- In step 3 of LSH, we obtain the signature matrix. Every component in the signature matrix is the dot product of the k-dimensional original and random vectors. However, we can reuse the space of the storage for random and original vector in each signature computation. So the complexity is the units used to store the result, which is O(n*d).

- In step 1 of fast hamming search, our target is q permuted random matrices. There is not any calculation. The process is just get q permutation and copy the original signature matrix to another place according the permuted order. So, the complexity is O(q*n*d).

- In step 2 of fast hamming search, we sort each binary vector according to their decimal value. Because we just exchange the location of each vector when it is necessary, only a temporary space for a vector is needed. Also, the temporary space can be reused. However, we need another space to store the decimal value of each vector. So, the space complexity for this step is O(n+d).
In step 3 of fast hamming search, we just choose the $2b$ nearest neighbour for the query vector in each sorted matrix. The space is just for the result again. So, the complexity is $O(2b*q)$.

By summing up all the steps, the complexity of this algorithm is $O(d*k + n*d + q*n*d + n + d + 2b*q)$. We know that $d<<k$. Besides, $q$ and $b$ are quite small. So, the main part of complexity is $O(n)$. 
Chapter 4

4. Implementation

4.1 Data set

This section will give a description and a rough analysis on the data set, obtained from the company Piction.

There are totally 14 XML files which is given by Piction. Most of the XML files are well named. One thing that should be noted is that some files are relatively huge. For instance, the file named basket_umo_metavalue is nearly 1GB. By contrast, the file basket_comments is just 2MB. Moreover, there is a file named financial_tables.sql. The content of this file is the SQL command to create the corresponding tables for the 14 XML file in the Oracle database of Piction.

When we check the content of some small-size files (because big files cannot be opened by applications), we can even observe some problems of those files:

- Duplicate values could appear across multiple files.
- Not all the attributes have values.
- Some attributes have a multi-level structure.
Not all the attributes created by the SQL command appear in every record.

Because of data privacy, nearly all the values are artificial values, which are encrypted keeping the basic statistic features but hiding the content details.

We can roughly infer what are those files used for by their file names and the structure given by the SQL commands in the financial_tables.sql file. There are more tables in the database than the XML files we get. However, we can obtain a sketch of the story the whole data set describes.

This data set is taken from an online shopping system. Some customer uses an account to log in the system to make some purchases in one session. A customer has a basket to put products in during every purchase. Each basket is identified by a structural object. This structural object includes the identifier of the customer, the identifier of this purchase and the identifier of audit tables. Although there are some other attributes, such as date, comments, we omit them because they do not always appear in every basket.

In every purchase, there are corresponding tables that record various aspects of details relative to the purchase behaviour, for instance, price, comments, status of this purchase, contact detail of the customer who makes this purchase, etc. This is demonstrated in Graph 9. But there are still some files, the meaning of which are not so clear. Thus we further investigate the content of each file to confirm the relationship among them and their specific roles in the database.

Because the size of some files is so large that they cannot be opened in an XML viewer application, furthermore we also need to do some statistic and dependent analysis, we should pre-process those XML files, which corresponds to the part D of the whole project.
4.2 Data pre-processing

In this section we discuss how to handle the problems mentioned in previous section, also we will decide which subset of data to be used to retrieve worthy knowledge.

4.2.1 File segmentation

To address the problem that the size of some files are too huge to be open in an XML viewer application, we have to split the original file into multiple small-size files. Each of the split XML files should be syntactical valid XML fragments, and each fragments should also be validated against the XML schema.

I used the tool proposed by Koen Serneels (2013). He describes in his blog that this tool utilizes the streaming application programming interface (API) for XML (StAX) techniques. This parses the XML line by line triggering different events, such as opening tag, closing tag and so on, other than loads the complete XML content into memory to handle it. Besides, this API has other advantages, such as can write XML documents, easy to code, and only passes data when needed rather than anytime. It turns out to be a proper solution in my project.

4.2.2 Database import

To conduct statistic analysis and detect dependency relationships, we import the whole data set into the database. As required in the project, tools should be open source, we choose Postgres which is an object-relational database management system with the features like extensibility and standards-
compliance, and Java as the file or database manipulation programming language. All the steps is shown in the Graph 10.

- First, using the JDBC to connect the database. JDBC is a Java-based data access technology (Java Standard Edition platform) from Oracle. It provides an API for the Java programming language that defines how a client can access a database. It also provides methods for querying and updating data in a database.

- Secondly, using JDOM to parse the XML files. JDOM is an open source Java-based document object model for XML. It allows random access and manipulation to an XML file. It is quite flexible and easy to use.

- Finally, using JDBC to execute the insert command based on the content of the XML files.

![Graph 10](image)

### 4.2.3 Target investigation

In this subsection, I describe the whole process of how to determine the task of this project.

Initially, I want to choose the purchase table and its relevant detail tables as the data source to resolve entities. For example, from the basket_contacts table, we can do some entity resolution analysis to figure out what kind of customer will buy what kind of product. However, because many contact details are fake values, we were stoped to carry on this task.

However, I find the table basket umo metavalue, where 100360 distinct customers and 72 different products can be confirmed. In total there are 2578273 distinct purchase behaviours. Based on customers and products, we can build a recommender system for customers to recommend products
which are bought most frequently among the similar customers of this customer, namely those product should also be suitable for him.

Our task is to build a collaborative recommender system. This sort of systems base on similarity measures between users. The computation of similarity is always time-consuming, especially we should calculate many times. As mentioned above, the data volume is huge in the data set. There are 100360 distinct customers. So, we should not compare the similarity for each pair customers exhaustively. Furthermore, the data are all in high dimensions. There are 72 different tag names. For efficiency, we should reduce the dimensionality. As a popular method, we select LSH as core algorithm to solve the problem of similarity computation and dimensionality reduction. Because a customer may buy a product many times, we use LSH for cosine distance. To address the exhaustive similarity comparison, I combine the cosine LSH with fast hamming search technique to gain further efficiency.

4.3 Core algorithm implementation

I compare the LSH method with the naïve method in this project. The procedure of original naïve method is as follows:

- We calculate the cosine similarity between the query customer and every other customer. The similarity is viewed as a weight of purchase times for products

- We calculate the product of purchase times and cosine similarity to get weighted purchase times for that product. For example, a customer has 0.5 similarity with the query customer and he buys product A 3 times. Then, the weighted purchase times of this customer for product A is 1.5 times.

- To a product, every customer will have a weighted purchase times. We sum up all the weighted purchase times to get the total purchase time of this product.

- We sort all products by their total purchase times.

- We output the top k products to recommend. Of course, they should not be bought by the query customer.

4.3.1 Measurements

I choose the following measurements to evaluate the methods used in my experiments from two aspects:

- Efficiency:
  - Time cost: the time taken by the system to get the recommend results.

- Result quality:
– Mean: the mean of the real ranking number of recommended products in the naïve method. The recommendation of the naïve method is the most accurate. The top 1 product is exactly the most popular product. Every product has its own ranking number in the naïve method. Then, we average the ranking numbers of the recommended products in the LSH method to evaluate the result quality of this methods. For example, we get three recommended products in the LSH method. The ranking numbers are the 4th, the 5th and the 6th. So, the mean of this result is 5.

– Standard variation: the standard variation of the real ranking number of the recommended products in the naïve method. We may get a recommended result in the LSH method with a good mean. Continuing the example above, the mean of 5 is acceptable. But one situation happens that the ranking numbers are 1th, 4th and 10th. Obviously this is not as good as the previous result. So, I also take the standard variation into account.

4.3.2 Initial value preset

As explained in the methodology Chapter 3, holding the value of 1-p and d unchanged, if we just choose b above the value d(1-p), then the pairs with number of bits different form each other no more than b are very likely to be candidate pairs. We choose the mean of the binomial distribution d(1-p) as the threshold of b. In fact we already know the threshold of similarity c. Because c = cos((1-p) π), the threshold of b is \( \frac{\arccos c}{\pi} d \).

Furthermore, to guarantee the accuracy, we always choose d \( \geq 20 \), b \( \geq 5 \) and d-b \( \geq 5 \). So, if 5 \( \geq \frac{\arccos c}{\pi} d \), then we set the value of b = 5, which is bigger than the threshold. As discussed in the methodology chapter, this will keep high probability to choose the eligible candidate. For the same consideration, if d - 5 \( \geq \frac{\arccos c}{\pi} d > 5 \), we should set b = \( \frac{\arccos c}{\pi} d \). However, we always set c in the range (0.0, 1.0], so it is impossible \( \frac{\arccos c}{\pi} d \geq d - 5 \), under d \( \geq 20 \).

For efficiency, we take as small as d. The reason for this is that except k and n, d is the dominant factor for time cost. Under test, d = 10 can hold the basic quality. So, we always set d = 10. Then the threshold of b is 10 \( \frac{\arccos c}{\pi} \), which is always less than 5. So, we set the value of b equal 5.

As to q, we test four cases q=1, q=4, q=8 and q=12. The results is following:
As we can see the mean becomes steadily around 6 from q=4. Similarly, the standard variation also converges to 5 from the point q=4. Thus, we set the initial q value of 4.

4.3.3 Experiment

The experiments are carried out on a computer with an Intel(R) Core(TM) i7-2670 QM 2.20 GHz dual CPU and 8.0 GB memory which runs Windows 7 Ultimate with Service pack 1(64-bit). All the methods are tested in single thread mode.

Under the initial values, I conduct the experiments on 10%, 60%, and 100% of the data set separately. My first experiment is to evaluate the efficiency of the LSH method. The Graph 12 shows the results.
As demonstrated in graph 12, the time cost of the naïve method is increasing quite quickly as the rise of data proportion. In contrast, the LSH method performs well. The increase is approximately linear.

My second experiment is concerning the quality of the results given by the LSH method. The Graph 13 depicts that the mean varies around 6 in different data proportion. Compared to the optimal 2, it is not bad. Moreover, The Graph 14 tells us that the results also do not distribute in a wide range, roughly around 4. With comparison with the optimal 1, it is still acceptable.

As a conclusion, the LSH method is scalable. Even handling huge amount of data, it is also efficient. Certainly, the quality of the result is also good. Thus, LSH method is a proper method for the recommender system in this project.

4.4 A simple tool
Based on the core algorithm implementation, I also develop a graphical tool, in which we can recommend the top three products to a customer according to certain similarity requirements. In the user interface, we can enter a customer ID in the customer textbox and set the threshold of similarity in the threshold textbox. When clicking the button recommend, the top three product will be demonstrated in the textboxes before the stars. When clicking the clear button, all the information you input will be cleared.

However, this tool also have the functionality to detect fault input. If you enter a customer ID which does not exist nor a numerical type, you will get an alert message, "Enter the customer ID in integer type", in the customer textbox. Similarly, if you input a threshold which is out of range \([0.0, 1.0]\) or not a real number, you will get another alert in the threshold textbox," Enter the threshold ranged in (0.0-1.0)".
Chapter 5

5. Conclusion and Future work

5.1 Conclusion

As Rutherford D. Rogers stated, We are drowning in information and starving for knowledge (Friedman, Hastie, and Tibshirani, 2001). When facing a huge amount of noisy and unstructured data, it is a challenge to clear up those data, get a well-regulated data set, and then analysis them to acquire certain beneficial knowledge to tackle some problems.

In this project, I confront the same challenge. Firstly I solve the problem caused by big volume of data file by dividing them into small pieces and import the data files into a database for further analysis. I get a relationship between different data group. After analyzing the characteristics of the records and their relationship, I put the main direction of this project on developing a recommender system tool that recommend products to customers based on the similarity between customers purchase behaviors. To address the efficiency problem brought by the high-dimensional and huge-amount data, I investigate the LSH method for cosine distance and furthermore improve it by combining with the fast hamming search technique. Through experiment, I verify that LSH method is indeed efficient and scalable meanwhile guarantee the quality of results. Finally, I use LSH method as the core algorithm for developing an efficient and accurate e-commercial tool for recommending products to customers.

5.2 Future work

There are still some places that need to be improved in the future. Firstly, the time taken by connecting a database and generating the customer-item matrix is the bottleneck of efficiency in this tool. These two task together take nearly a third of the whole process time. Secondly, in this project, I only verify one core algorithm for this tool. There are also other efficient machine learning algorithms such as SVD (Friedman, Hastie, and Tibshirani, 2001). Moreover, this tool only processes artificial data so far. If there is an opportunity, I hope this tool can be integrated into the Piction's platform to test real-world data sets and obtain a real meaningful result. Last but not least, this tool can have a further improvement on the user interface and can be added more analysis functionalities such as associate analysis, statistic visualization.
References


