**Association rule mining**

Association rules are conditional statements that help in finding relationship between random objects in a database or information repository. It gives information that how strongly two objects are related to each other which is impossible to detect with naked eyes.

It consists of two parts, an antecedent (if) and a consequent (then).  An antecedent is an item found in the data. A consequent is an item that is found in combination with the antecedent.

The core part of association rule is **apriori algorithm**. It is based on support and confidence. *Support* is an indication of how frequently the items appear in the database. *Confidence* indicates the number of times the if/then statements have been found to be true.

It is widely used now days in industries to find out hidden trend and pattern.

For example “*If a person buys milk then there is 60% of chances that he will buy eggs too*”. They play important role in determining customer behavior in grocery store, in shopping basket data analysis, product clustering, catalog design and store layout.

Programmers use association rules to build programs capable of Machine Learning. Machine learning is a type of artificial intelligence that seeks to build programs with the ability to become more efficient without being explicitly programmed.

**Background for our research**

A number of previous studies have been done on large open projects such as Jedit, Eclipse, Apache Ant, and Mozilla Firefox etc. to detect the coupling between different java files, class and methods. In past Annie Ying has developed an approach which is based on association rule mining to find the recommendation, She generally focused on the files. For example if a developer changes a file then her approach will recommend him other files too that need to be changed for newer version of software. Similarly Zimmermann and his associates developed a tool named rose that too works on association rule mining of CVS. Their main concern was to find the finer grained entities, which was absent in ATT Yings work. They used Association rule mining with minimum support and confidence. Their tool seems to be powerful in terms of suggesting and predicting next likely to be change code elements. It also prohibits and warns about incomplete changes. It can also used for identifying coupling between different code elements.

**Dataset**

We have taken five major releases of Apache ant ranging from 1.5 to 1.9 for our study. We have considered only valid commits, commits that were actually made to code. Neglected commits made for signature changes, comments update etc. we pulled out those commits into three different txt files. We have divided this data into 1:4 ratio and taken 75% one for training and rest for testing. We have taken changes made in one commit as one transaction for association rule mining.

**Methodology**

Those code elements that were changed in single commits are taken as one transaction. We created three different txt files for four different versions of apache ant. We applied apriori algorithm to get the association rule. The minimum support chosen was 9% and the minimum confidence was 50 %.

Why we chose 9% as our support?

As our ‘change file’ contains lots of commits between two releases so, it might be that we miss out lots of relevant rules if we go for higher supports. We calculated rules with 5 different supports ranging from 30% to 9% and we figured it out that 9% gives better results than any other supports. If we go below 9% then we might get a large number of rules but that might be irrelevant.

The rule below suggest a developer that when ever P4Change.java gets changes then there is strong probability that P4Counter.java may get changed.

(/ant/core/trunk/src/main/org/apache/tools/ant/taskdefs/optional/perforce/P4Change.java) => (/ant/core/trunk/src/main/org/apache/tools/ant/taskdefs/optional/perforce/P4Counter.java )

V1 V2 V3 V4 V5

*Code elements that got changed in each version*

From the above diagram it can be easily figured it out that B,D are generally changing together. So, if a developer is trying to change D in current version then he has to pay much more attention on B too.

For calculating the rules we have only considered single dimension rules. i.e. one antecedent and one consequent. Our rules are weighted based on lift the more the value of lift is the more important that rule will be. A lift is the measure of co-relation between two entities. If the value of lift for the two entities is greater than 1 then those entities are positively co-related if the value is less than 1 then they are negatively correlated.

**Evaluation**

We evaluated our approach by precision and recall value. We followed the way what Thomas zimmermann did for his calculation. We took a transaction, which contains a query (Q) and an expected outcome (E). A query is a single entity of a transaction and the expected outcome for a query (Q) is the entities that got changed with that query.

Once we have the query, we move to the association rules generated from apriori and unions all the rules that have same antecedent as query and we figure out how many consequents(C) are matches with the expected outcome (E), which gives us recall (R) and then we counted how many of the consequents were correct for each query. In simple language the precision (P) describes which fraction of the returned items was actually expected.

R = |C ∩

**Results**

After having association rules, we calculated precision and recall for our method. As we didn’t played with enough version history so, the precision and recall for our method was not that much good. The precision comes to be around 10% and recall was petty ugly around 1.5%.