CHAPTER 10

COMPONENT, CODE REUSABILITY USING COLLABORATIVE, CONTENT BASED AND TIME VARIANT RECOMMENDATION ALGORITHM

10.1 INTRODUCTION

Along with spurt in technological advancement, the amount of information is increasing tremendously leading to the difficulty for a naive user to extract the desired information efficiently from such a huge collection.

Code reusability basically refers to the technique of reusing available software or component wherever possible as this increases the productivity and even saves the time in testing phase as the available component has been used before and is tested and verified. But though this technique increases the productivity on large scale, it is not easy to apply it efficiently. The success of this technique actually relies upon how quickly the developer can receive the required component, because any delay in the delivery of the component will be considered as an overhead by the developer, and they will try to skip this section and will end up writing their own code for the required functionality, creating a redundancy in the repository and wasting a lot of time in it (Mark & Bryan 2001). This is a current hot-topic in research area of making an efficient agent for code reusability.
In such a predicament Recommendation System plays a vital role. Recommendation Systems are a type of Information Filtering Systems which predict the preference of a particular user towards a particular item, using the characteristics of items and past preferences of the user and his/her similarity with other users, resulting in a quick delivery of required data (Michael & Barry 2007; Batul et al. 2003)

10.2 RECOMMENDATION SYSTEM

Recommendation Systems are broadly classified into two categories: Collaborative filtering, Content Based Filtering and a third category named as Hybrid Systems is being recently introduced which is a combination of the first two.

The following sections will give a brief introduction to the above mentioned categories.

10.2.1 Collaborative Filtering

It is based on user’s activity, behaviour and preferences and tries to predict what the user would like in the future based on similarities with other users. User may provide the preference (or what is referred here as ‘vote’) explicitly or implicitly. For example, a user giving a rating to a movie or a restaurant out of five is considered as an explicit vote. But in many scenarios the explicit voting cannot be applied logically, like in online news feed website; here the user preference is captured by tracing their browsing pattern and their clicks over the feeds available which is technically called as implicit vote (Michael 1999). Recommender System may then perform computation over the votes available from the different users and can conclude the preferred category of the news for the user, so when the user logs in next time, the system can provide a more personalized view than before which is convenient to extract the desired information (Zan et al 2004).
Collaborative Filtering is again subdivided into two categories namely Memory Based and Model Based.

In Memory Based Filtering the entire database about user past preferences is considered for prediction study whereas the Model Based Filtering uses specific algorithms to first model the given user database before processing it for the prediction study.

Collaborative filtering is one of the most widely used technique in recommendation systems as this technique doesn’t use the specific details about items which is its major advantage as this removes the overhead of content analysis (this is very resource intensive and requires high technical skills). This is the major factor behind its success. The traditional algorithms being used in collaborative filtering are: K – nearest neighbors (KNN), along with Pearson Correlation Coefficient or Vector Similarity (Emmanouil & Konstantinos 2003).

10.2.2 Content Based Filtering

Unlike Collaborative filtering, content based filtering is based on the content analysis of the items over which the prediction of the user-choice has to be performed. They are based on the characteristics and information about the item. Basically this algorithm will recommend those items which have the maximum similarity with the items preferred by the user in the past. It relies upon the profile of the item which is prepared by setting the discrete features and attributes after the demographic analysis about the items present in the given set. The traditional algorithms being used under these techniques are Bayesian Classifier, Rocchio Classification and Artificial Neural Networks.
10.2.3 Hybrid Systems

Though the above two mentioned techniques are widely accepted and applied, but recent research has proven that the combination of these two techniques into one system can yield a better result in some cases. The work in this chapter uses the hybrid systems as they can provide more accurate results though they are computationally complex to build and discusses them in greater detail ahead (Robin 2007; Thomas & Robin 2000).

10.3 CODE REUSABILITY

10.3.1 Current Scenario of Code Reusability

Presently there are many “code search engines” which have indexed the code snippets from millions of open source repositories. A user can enter the query about the required component in a natural language and the search engines perform a semantic search over the indexed snippets – a scenario very much similar to the web search. Available code search engines are Google Code Search, Krugle, Koders, jExamples, Codase etc. These search engines also depends largely upon the documentation of the available components, as they are the base of the search when the query entered is in natural language.

10.3.2 Limitations in the Current Scenario for Developers

While a developer is tending to reuse an available component from a framework or a library; usually developer is aware of only the type of component required, but developer is unaware of the particular name of the object, method or the structural context. Figure 10.1 shows the average statistics of the search performed for the component on various code search engines (Hoffmann et al 2007).
Figure 10.1  Average component search by developers on various search engine

The statistics clearly depicts that the code search engines alone are not sufficient to serve the requirements of the developer as their result quality is highly dependent on the knowledge of the component being searched for. Even the results generated by the query are not that easy to parse and fetch the required component, as the search result also consists of the source code which is either partial or not compilable, because the engines retrieve the individual source files instead of the entire projects. The result also includes many code components which might be not compatible with the current working environment version of the developer. This creates an overhead to the developers which will ultimately lead to these two situations: either the developer will intensively search for the required component refining the query each and every time after watching the results, or the developer will quit the search and will end up writing own code for the required functionality. Both of these situations will highly affect the efficiency and will decrease the overall productivity of the development generating many redundant components, which might have been reused if efficiently supplied to the developer in time. Besides this the current techniques also have one
more limitation, the repositories of the organizations are constantly updated by expert developers, the newly added component usually outperforms the previous one in terms of efficiency; though it is highly recommended to use the new component but the naïve developers rarely come across them as they are unfamiliar and tend to use the old ones with which they are acquainted of unless some expert recommends them the new one.

### 10.4 Recommendation System for Code Reusability

So it can be concluded from the above section that what is actually required is something more than just a code search engine. A system which can infer the requirement of the developer and suggest the developer the required component at that particular time is required (Kostas et al. 2012). The System should be intelligent enough to find the pattern from the developer’s recent history and also the context of the present scenario before suggesting any components. The system should also remove the overhead from the developer to explicitly find the required components. This situation lays a perfect platform for the landing of the Recommendation Systems (Frank et al. 2005).

Recommendation Systems are capable of gathering the information about the histories of the component being used by the developer and can use this information to infer the need of a particular component. The alterations in the present context which are going to be proposed will also solve the problem of unfamiliarity of the newly available components to a great extent.

The next section explains the mathematical details of a generic recommender system and also maps its usage for the component reusability in software development.
10.5 GENERIC RECOMMENDER SYSTEM

Recommendation Systems are easier to understand if the context of users and items are considered where the aim is to recommend an item to a user, much like those systems which are already being used in many e-commerce web-sites. For using these systems in component reusability, the traditional equations can be applied and the items can just be replaced with software components of the repository and users with the classes developed by the software developers tending to reuse the components. So further in this chapter if users is mentioned then it actually means a class and when items is mentioned there it means a software component which can be a method or a set of function, this will keep the context more generic and easy to visualize. Proceeding forward, now there is a need for a system to be built that would recommend a software component to an active class being developed by the active developer (one who is currently involved in the development phase and coding) which is most probabilistic to be used by the developer, because it is obvious in object oriented programming that the class is being developed for a specific purpose, which can be either Database Connection, GUI handling, Server Socket Connection, etc. In each case the component used to develop this class will be very much related and will also be similar to the one used by the past developers building the class for the same tasks. Mapping this scenario can save a lot of time in the development phase and will increase the efficiency. Now each part of collaborative, content based and time variance factor will be picked individually and each of them merged appropriately to obtain the final equation.

10.5.1 Collaborative Filtering in reusability

As discussed earlier, the collaborative filtering analyses the similarity between the users and considers that more similar users will share preferences in the future as they did in the past. Core working of a
collaborative filtering actually depends upon how efficiently the similarity between the two users can be found. Before that the system should have the required data i.e. the preference of the users over the available items. For this it is essential that recommender system should be installed in each of the systems of the developers to trace the software components used by the each developer and should collect the combined data of all the developers in a single database. After sufficient data is collected the system is now ready to predict the next required component to the current or active class considering the components which the developer has already used in the development phase. It is expected that the active class will most probably use the same components as other classes which have also used the components that the active class has used till now. There would be many classes which might have used one or more components that the active class has used till now. But the recommender system should prioritize that class which has the most similarity with active class, because the most similar class will tend to produce a more accurate recommendation result. So now the question ends with, how to find the priority of each user towards the active class. This priority is technically called as weight and Pearson Correlation Coefficient is one of the traditional equation in (10.1) being used in collaborative filtering to calculate the weight.

\[ P_{x,y} = \frac{\text{COV}(X,Y)}{\sigma_x \sigma_y} \quad (10.1) \]

where \( P_{x,y} \) denotes the relation between the variables \( X \) and \( Y \), \( \text{COV}(X,Y) \) is the covariance while \( \sigma_x \) and \( \sigma_y \) is the standard deviation of \( X \) and \( Y \) respectively. This formula can be expanded as the following equation (10.2)

\[ w = \frac{\sum_{i=1}^{n} (x_i - \overline{X})(y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n} (x_i - \overline{X})^2 \cdot \sum_{i=1}^{n} (y_i - \overline{Y})^2}} \quad (10.2) \]
where \( w \) is the weight or in this case it denotes the similarity. \( \bar{X} \) and \( \bar{Y} \) denote the average votes of users X and Y respectively. The formula to calculate the average vote \( \bar{v}_i \) for any person \( i \) is as in equation (10.3).

\[
\bar{v}_i = \frac{1}{|C_i|} \sum_{j \in C_i} v_{i,j}
\]

(10.3)

where \( C_i \) is the set of items user \( i \) has voted for and \( v_{i,j} \) is the vote of user \( i \) over the item \( j \).

Besides Pearson correlation coefficient there are many other equations which can be used for the calculation of the similarity between users, one of them being Vector Similarity.

The formula for calculating the weight using Vector Similarity as in equation (10.4)

\[
w_{a,j} = \sum_{i} \frac{v_{a,i} \bar{v}_{a,i}}{\sqrt{\sum_{k \in C_a} v_{a,k}^2 / \sum_{k \in C_a} v_{a,k}}} \frac{\bar{v}_{i,j} \bar{v}_i}{\sqrt{\sum_{k \in C_i} v_{i,k}^2 / \sum_{k \in C_i} v_{i,k}}}
\]

(10.4)

It has been found that the vector similarity outperforms the Pearson Correlation when the number of users is very large.

After the calculation of weight (similarity) between the users and the average or the mean vote of the individual user, now the Predicted Vote of an active user \( a \) over a particular item \( i \) can be calculated by the formula of the collaborative filtering given below in equation (10.5).

\[
P_{a,j} = \bar{v}_a + N \sum_{i=1}^{n} w_{(a,i)}(v_{i,j} - \bar{v}_i)(10.5)
\]

where \( N \) is the normalizing factor having the value as

\[
N = \frac{1}{\sum_{i \in \text{sim}(a,i)}}
\]

(10.6)
The predicted vote calculated by the above formula serves the need of recommending items by analysing the vote of the similar users, but one problem still persists, that is the unfamiliarity of newly added items. In above formula the old items will be recommended first as they will have more weightage and thus the recommendation of the newly available items though used by small number of users will always be suppressed by the old item. So there is a need to find a solution by which this recommendation system can recommend the newly available if it is used by some user, because this newly available component will surely outperform the older one as it has arrived later. So this work is proposing the inclusion of a time variant factor in the above formulas so that the vote over the older items should decay with the passage of time and that new item gets a chance to appear in the results of recommendation even though its users are less. Therefore a time dependent factor is being multiplied which will degrade the vote of a particular user over the item so that the trend in the system can be maintained.

The formula for the average vote appending in equation (10.3) the time dependent factor will become as following equation (10.7)

\[ \bar{\theta}_i = \frac{1}{|C_i|} \sum_{j \in C_i} (\hat{\theta}_{i,j} \ast 2^{-\lambda(t-t_{i,j})}) \] (10.7)

where \( t \) is the current time stamp and \( t_{i,j} \) is the time stamp when the user \( i \) voted for the item \( j \).

\( \lambda \) is the decay rate. Higher the value of \( \lambda \) faster will be the degradation of the vote.

Similarly the formula for the weight calculation by the Pearson correlation co-efficient will be modified as the following
\[ w_{a,j} = \frac{\sum_{i} \left( \theta_{a,i} \left( z^{-\lambda(t-i,a,j)} \right) \bar{y}_{a} \right) \left( \theta_{t,j} \left( z^{-\lambda(t-t_i,j)} \right) \bar{y}_{t} \right)}{\sqrt{\sum_{i} \theta_{a,i} \left( z^{-\lambda(t-i,a,j)} \right) \bar{y}_{a}^2 \sum_{j} \theta_{t,j} \left( z^{-\lambda(t-t_i,j)} \right) \bar{y}_{t}^2}} \]  

(10.8)

The formula for the predicted vote will also be modified to equation (10.9).

\[ P_{a,j} = \bar{y}_{a} + N \sum_{i=1}^{n} w_{a,j} \left( \{ \theta_{t,j} + 2^{-\lambda(t-t_i,j)} \} - \bar{y}_{t} \right) \]  

(10.9)

But still this formula has a limitation in the sense that it does not consider the context of the component. While it is important to consider the context of component or item before recommending it, as it might be possible that though the two users share the maximum similarity but the present context of the active user is different from that of the second user. So there is a need of understanding the component first and this is called the demographic analysis of the component. So now the Content Based Filtering is introduced.

10.5.2 Content Based Filtering in reusability

Content based filtering methods are based on information to be recommended. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past (or is examining at present).

Two popular algorithms are available for content based filtering:

- Rocchio’s Algorithm
- Bayesian Classifier.

Both the algorithms have a common short-coming i.e. they require pre-specification of the number of terms used in the profile. Hence a latest ML algorithm called Winnow Algorithm is going to be used for the purpose of calculating the content based filtering. It is very similar to perceptron
algorithm but uses multiplicative scheme instead of additive scheme in weight-update, hence converges quickly.

10.5.3 Working Principle of Winnow Algorithm

In this application there is a need to create a profile for each software component which can also be created through a system after semantically analyzing the documentation and component description. After this analysis a number of tags corresponding to each component will be obtained, precisely defining it. Initially random weights are given to the tags for the particular component. Then the predicted vote is recursively calculated using the following formula

\[ \sum_{i=1}^{n} w_i x_i > \theta \]

where \( \theta \) is the threshold

i) If the sample is correctly classified and do nothing.

ii) If the prediction is 1 but result was 0 perform a demotion step (like divide by \( a \))

iii) If the prediction is 0 but the result was 1 perform a promotion step (like multiply by \( a \))

This content based filtering vote can be used in an analogous form instead of the digital 0 or 1. It will be appended to the collaborative filtering equation by multiplying it by a constant.

\[ \sum_{i=1}^{n} w_i x_i = \mu \]  

(10.10)
10.5.4 Combining the Content based Equation with the Collaborative Filtering

Though this is not considered as the novel approach to combine the two different filtering algorithms but it fits well in this context. So after the content based filtering is appended the following final formula is obtained as (10.11).

\[ p_{a,j} = \bar{d}_a + N \sum_{i=1}^{n} w_{a,i} \{ (v_{i,j} \cdot 2^{-\lambda t_{i,j}}) - \bar{d}_i \} + \psi \mu_{a,j} \]

(10.11)

10.6 CONCLUSION

Hence a formula which is Collaborative, Content based and time variant has been obtained successfully. This promises to provide more accurate result though it is computationally complex.

When applied to the field of Component reusability the predicted vote of an item for a particular user can be obtained. This recommendation is crucial to component reusability as it solves the problem of the developer having to search extensively for existing components that match developer requirements and also prevents the developers from going back to writing redundant lines of code. Clearly, this enhances productivity of the development phase and reduces the development time significantly and thus ML algorithms can be used in code or component reusability.