Feature Selection and Case Selection Methods Based on Mutual Information in Software Cost Estimation

Shihai Shi
Abstract

Feature Selection and Case Selection Methods Based on Mutual Information in Software Cost Estimation

Shihai Shi

Software cost estimation is one of the most crucial processes in software development management because it involves many management activities such as project planning, resource allocation and risk assessment. Accurate software cost estimation not only does help to make investment and bid plan but also enable the project to be completed in the limited cost and time. The research interest of this master thesis will focus on feature selection method and case selection method and the goal is to improve the accuracy of software cost estimation model.

Case based reasoning in software cost estimation is an immediate area of research focus. It can predict the cost of new software project via constructing estimation model using historical software projects. In order to construct estimation model, case based reasoning in software cost estimation needs to pick out relatively independent candidate features which are relevant to the estimated feature. However, many sequential search feature selection methods used currently are not able to obtain the redundancy value of candidate features precisely. Besides, when using local distance of candidate features to calculate the global distance of two software projects in case selection, the different impact of each candidate feature is unproven.

To solve these two problems, this thesis explores the solutions with the help from NSFC. In this thesis, a feature selection algorithm based on hierarchical clustering is proposed. It gathers similar candidate features into the same clustering and selects one feature that is most similar to the estimated feature as the representative feature. These representative features form the candidate feature subsets. Evaluation metrics are applied to these candidate feature subsets and the one that can produce best performance will be marked as the final result of feature selection. The experiment result shows that the proposed algorithm improves 12.6% and 3.75% in PRED (0.25) over other sequential search feature selection methods on ISBSG dataset and Desharnais dataset, respectively. Meanwhile, this thesis defines candidate feature weight using symmetric uncertainty which origins from information theory. The feature weight is capable of reflecting the impact of each feature with the estimated feature. The experiment result demonstrates that by applying feature weight, the performance of estimation model improves 8.9% than that without feature weight in PRED (0.25) value.

This thesis discusses and analyzes the drawback of proposed ideas as well as mentions some improvement directions.
Contents

Chapter 1. Introduction .............................................................................................................3
  1.1 Background ......................................................................................................................3
  1.2 Problem Isolation and Motivation ..................................................................................3
  1.3 Thesis Structure ..............................................................................................................4

Chapter 2. Software Cost Estimation Based on Mutual Information ...............................6
  2.1 Entropy and Mutual Information .....................................................................................6
    2.2.1 Entropy ....................................................................................................................6
    2.2.2 Mutual Information .................................................................................................7
  2.2 Case Based Reasoning .....................................................................................................7
  2.3 Evaluation Criteria .........................................................................................................8
    2.3.1 MMRE and MdMRE ...............................................................................................8
    2.3.2 PRED (0.25) .........................................................................................................9
  2.4 Feature Selection ...........................................................................................................9
  2.5 Case Selection ...............................................................................................................10
  2.6 Case Adaptation ............................................................................................................10

Chapter 3. Sequential Search Feature Selection ............................................................11
  3.1 Principle of Sequential Search Feature Selection .........................................................11
  3.2 Related Work ...............................................................................................................11
  3.3 INMIFS in Software Cost Estimation .........................................................................13

Chapter 4. Clustering Feature Selection ...........................................................................15
  4.1 Drawback of Sequential Search Feature Selection .......................................................15
  4.2 Supervised and Unsupervised Learning .....................................................................15
  4.3 Principle of Clustering Feature Selection .....................................................................16
  4.4 Related Work ...............................................................................................................16
  4.5 Hierarchical Clustering ................................................................................................17
  4.6 Feature Selection Based on Hierarchical Clustering ..................................................18
    4.6.1 Feature Similarity ...................................................................................................18
    4.6.2 Feature Clustering ................................................................................................18
    4.6.3 Number of Representative Feature ......................................................................18
    4.6.4 Choice of Best Number .........................................................................................19
    4.6.5 Schema of HFSFC ...............................................................................................20
4.6.6 Computational Complexity of HFSFC .................................................. 21
4.6.7 Limitation of HFSFC ........................................................................... 21

Chapter 5. Feature Weight in Case Selection ............................................ 22
  5.1 Principle of Feature Weight ................................................................. 22
  5.2 Symmetric Uncertainty ....................................................................... 22
  5.3 Feature Weight Based on Symmetric Uncertainty .............................. 22
  5.4 Global Distance and Local Distance ................................................... 23

Chapter 6. Experiment and Analysis ......................................................... 24
  6.1 Data Set in the Experiment ................................................................ 24
     6.1.1 Data Type ..................................................................................... 24
     6.1.2 ISBSG Data Set ............................................................................ 24
     6.1.3 Desharnais Data Set ...................................................................... 25
  6.2 Parameter Settings ............................................................................... 26
     6.2.1 Data Standardization .................................................................... 26
     6.2.2 K-Fold Cross Validation ............................................................... 26
     6.2.3 K Nearest Neighbor ..................................................................... 27
     6.2.4 Mean of Closet Analogy ............................................................... 27
  6.3 Experiment Platform and Tools ............................................................ 27
  6.4 Experiment Design ............................................................................... 27
  6.5 Experiment of Sequential Search Feature Selection .......................... 28
  6.6 Experiment of Hierarchical Clustering Feature Selection .................. 30
     6.6.1 Different Number of Representative Features .............................. 30
     6.6.2 Different Number of Nearest Neighbors ...................................... 31
  6.7 Comparison of Feature Selection Methods .......................................... 32
  6.8 Experiment of Feature Weight in Case Selection ............................... 33

Chapter 7. Conclusion and Feature Work .................................................. 35
  7.1 Conclusion .......................................................................................... 35
  7.2 Future Work ......................................................................................... 35

Acknowledgement ...................................................................................... 36
References .................................................................................................... 36
Appendix One: Developer Manual .............................................................. 39
Appendix Two: User Manual ...................................................................... 54
Chapter 1. Introduction

1.1 Background

Software systems are larger and more complex than ever before. Some typical software crisis like project delay, over budget and quality defect appear from late 1960s. The “CHAOS SUMMARY FOR 2010” published by The Standish Group indicates that only 32% of all the projects are successful, which means these projects are completed within deadline and budget. However, 24% of all the projects are not completed or canceled and the other 44% are questionable due to serious over budget. According to some professional analysis, the under-estimated the cost of projects and the unstable requirements are the two main factors that lead to the failure of software projects [1].

Software cost estimation is not only helpful to make decision for reasonable investment and commercial bidding, but also crucial for project managers to set up milestones and take charge of the progress. Therefore, it is necessary and important to do some researches into software cost estimation in order to improve the estimating accuracy.

1.2 Problem Isolation and Motivation

Software cost estimation mainly focuses on building different estimation models to improve the estimating accuracy in early stages of a project. The development of software cost estimation begins from process-oriented and experience-oriented modeling techniques, and then function-oriented, artificial intelligence-oriented and object-oriented modeling techniques are widely used for some years. To some extent, the modeling techniques mentioned above achieve good performance, but still have several common drawbacks [4][5]:

(1) The data sets are too small and contain missing fields;
(2) Some modeling techniques treat the numeric data and categorical data equally;
(3) Some modeling techniques do not employ feature selections.

Some experts try to divide all these modeling techniques into three categories: algorithm-based technique, non-algorithm-based technique and mixture technique [2].

The basic idea behind algorithm-based technique is to find out the factors that may influence the cost of software project and build a mathematical formula to calculate the cost the new project. The best known algorithm-based techniques in software cost estimation are represented by the Constructive Cost Model (COCOMO) suite [6][7a][7b] proposed by Professor Boehm who works in the software engineering research center at USC. COCOMO selects the most important factors that are relevant to the software cost and obtain the formula by training a large quantity of historical data sets.

The non-algorithm-based techniques include expert estimation, regression analysis, analogy etc. In the expert estimation, the experts are in charge of all the progress in cost estimation and some details of estimation is unclear and unrepeatable [6]. The drawback of expert estimation is that the personal preference and experience may bring risk in the estimation. Regression analysis needs to employ the historical project data to estimate new project cost. However, regression
analysis is sensitive to outliers and has to satisfy the precondition that all the variables are uncorrelated. Besides, regression analysis requires large data set for training regression model. These three limitations prevent regression analysis from being widely used in software cost estimation. Analogy estimation needs to select one or more software projects that are similar to the new project in the historical data set in order to estimate the cost of new project via the cost of historical projects. It mainly contains four stages:

1. Evaluate new project to decide the choice of similar historical data set;
2. Decide the choice of factors that may influence the cost of project and pick out the similar historical projects;
3. Select suitable formula to calculate the cost of new project via the cost of similar historical projects;
4. Adjust the calculated cost based on workload and current progress of the new project as the final estimating cost.

The advantages of analogy estimation include:

1. It is more accurate than expert estimation;
2. It is more reasonable to used historical data to estimate new data and it is repeatable;
3. It is more intuitive in constructing estimation model and making cost estimation.

The disadvantages mainly come from two aspects: it is dependent on the availability of historical data and it needs to find out similar historical projects.

Sheppard et al. [4] suggests applying analogy in software cost estimation. They conduct several experiments of software cost estimation through nine different data sets and demonstrate that analogy estimation performs better than expert estimation and regression analysis [5]. Based on the procedure of analogy estimation, Sheppard et al develop the aided estimation tool “ANGEL”. In addition, Keung et al. [9] propose the Analogy-X in order to improve the original analogy estimation.

There are three main issues in construct estimation model using analogy estimation:

1. How to extract the powerful feature subset from the original feature set to construct model;
2. How to define the similarity between different software projects in order to find out the most similar one or more historical projects;
3. How to apply the cost of similar historical projects to estimate the cost of new project.

The analogy estimation is the research focus of this thesis and the following chapters will discuss and explore these three issues.

### 1.3 Thesis Structure

This thesis research area is analogy estimation in software cost estimation, and the main focus is on feature selection and case selection. In the Chapter 2, concepts and applications of entropy and mutual information in information theory are introduced. Also, the procedure of case based reasoning, which is one branch of analogy, is discussed. The design principle of sequential search in feature selection and some related work is presented in Chapter 3 as well as comments about this kind of feature selection methods are given. The design principle of clustering feature selection can be found in Chapter 4 and a novel feature clustering feature selection method named HFSFC will be proposed in the same chapter. In chapter 5, case selection replaces feature
selection as the research interest. It describes the feature weight design principle and employ symmetric uncertainty as the feature weight. Chapter 6 presents all the details of experiments including data set, experiment platform and tools, parameter settings etc. Besides, the experiment results are illustrated and the analysis is presented. The last chapter will conclude the research of this thesis and summarize the feature work.
Chapter 2. Software Cost Estimation Based on Mutual Information

2.1 Entropy and Mutual Information

2.2.1 Entropy

Entropy, which originates from physics, is measure of disorder. But in this thesis, it is treated as a measure of uncertainty between random variables. The concept of entropy in information theory is proposed by C.E. Shannon in his article “A Mathematical Theory of Communication” in 1948 \[10\]. Shannon points out that redundancy exists in all information and the quantity of redundancy depends on the probability of the occurrence of each symbol. The average amount of information which eliminates the redundancy is called “information entropy”. In this thesis, the word “entropy” refers to “information entropy”.

The calculation formula of entropy is given by Shannon. Suppose that \( X \) represents the discrete random variable, and \( p(x) \) represents the probability density function of \( X \), then the entropy of \( X \) can be defined as follows:

\[
H(X) = -\sum_{x \in S_X} p(x) \log p(x)
\]  

(2.1)

Suppose that \( X \) and \( Y \) represent two discrete random variables, respectively. The uncertainty between \( X \) and \( Y \) can be defined as the “joint entropy”:

\[
H(X, Y) = -\sum_{x \in S_X} \sum_{y \in S_Y} p(x, y) \log p(x, y),
\]  

(2.2)

where \( p(x, y) \) is the joint probability density function of \( X \) and \( Y \).

Given random variable \( Y \), the uncertainty of random variable \( X \) can be described as “conditional entropy”:

\[
H(X|Y) = -\sum_{x \in S_X} \sum_{y \in S_Y} p(x, y) \log p(x|y).
\]  

(2.3)
2.2.2 Mutual Information

The mutual information of two random variables is a quantity that measures the mutual dependence of two random variables. Suppose that X and Y represent two discrete random variables, and then the mutual information can be defined as below:

\[
I(X, Y) = \sum_{x \in \Omega_x} \sum_{y \in \Omega_y} p(x, y) \log \frac{p(x,y)}{p(x)p(y)}.
\]  

(2.4)

If X and Y are continuous random variables, the formula for mutual information is written as:

\[
I(X, Y) = \int_{\Omega_y} \int_{\Omega_x} p(x, y) \log \frac{p(x,y)}{p(x)p(y)} \, dx \, dy.
\]  

(2.5)

Moreover, mutual information can be calculated through entropy and conditional entropy:

\[
I(X; Y) = H(X) - H(X|Y).
\]  

(2.6)

The mutual information \(I(X;Y)\) represents the dependency between two random variables. So the higher the mutual information value is, the more relevant the two random variables are. If the mutual information value between two random variables reaches 0, then it means these two random variables are totally independent of each other. Otherwise, if the value is 1, then random variable X completely depends on random variable Y.

The relationship between random variable X and Y is illustrated in the figure 1. The left circle and right circle represents the entropy of each random variable. The intersection part is the mutual information between X and Y. The pink part in left circle and the blue part in right circle represent the conditional entropy \(H(X|Y)\) and \(H(Y|X)\), respectively. The whole area that is painted by colors shows the joint entropy of X and Y.

The motivation to consider the mutual information in software cost estimation is its capability of measuring arbitrary relations between features and it does not depend on transformations acting on different features\(^{[1]}\).

2.2 Case Based Reasoning

Recent years, analogy estimation has been applied to software historical data sets by many researchers\(^{[4]}\)\(^{[5]}\)\(^{[9]}\) in order to construct estimation model. Case based reasoning is one kind of analogy estimations\(^{[5]}\). It employs several historical software projects that are similar to the new project to predict the cost of the new one.

Generally speaking, there are four stages in the case based reasoning\(^{[8]}\):

(1) Find out one or more cases that are similar to the new case;

(2) Use those similar historical cases to solve the problem;
Adjust current solution to refine the current results;
Add the data and the solution into data set for next problem.
The first and second stages are the core parts in case based reasoning. When refers to software cost estimation, the core tasks remain in two parts:
(1) Find out the best feature subset that can help to construct the estimation model;
(2) Find out the most similar cases in historical data set to estimate the cost.

Feature selection, case selection and case adaptation consist of the three procedures in software cost estimation using case based reasoning.

![Flow chart of case based reasoning in software cost estimation](image)

In the feature selection, best feature subset that can help to predict the software project cost will be picked out. The candidate features in the set which is informative to predict the cost and independent with other features will be considered suitable to keep. All the kept features will compose the feature subset.

In the case selection, the historical software projects that are most similar to the new one will be picked out from all projects.

In the case adaptation, it provides a solution to estimate the cost of new project by using the similar historical projects which are picked out in the case selection.

The remaining section of this chapter will talk about the feature selection, case selection and case adaptation in detail. These three modules consist of the software cost estimation model. Before that, evaluation criteria of estimation model performance need to be mentioned.

### 2.3 Evaluation Criteria

In software cost estimation, evaluation criteria are used to assess the performance of the estimation model. Many criteria can be used as the evaluation criteria of software cost estimation, such as MMRE (Mean Magnitude of Relative Error)\(^{[14]}\), MdMRE (Median Magnitude of Relative Error)\(^{[14]}\), PRED (0.25), AMSE (Adjusted Mean Square Error)\(^{[15]}\), SD (Standard Deviation)\(^{[14]}\), LSD (Logarithmic Standard Deviation)\(^{[14]}\) etc. In this thesis, MMRE, MdMRE and PRED (0.25) are adopted as the evaluation criteria because they are widely accepted by many researchers in this field\(^{[16]}\)\(^{[17]}\). MMRE and MdMRE can be used to assess the accuracy of estimation while the PRED (0.25) is used to assess the confidence level.

#### 2.3.1 MMRE and MdMRE

MMRE value is the mean value of the relative error in software cost estimation. It can be defined
as below:

\[ MMRE = \frac{1}{n} \sum_{i=1}^{n} MRE_i, \quad (2.7) \]

\[ MRE_i = |\frac{AE_i - EE_i}{AE_i}|. \quad (2.8) \]

In the equations above, \( n \) represents the number of projects, \( AE_i \) is the actual (real) cost of the software project \( i \) while \( EE_i \) is the estimated cost of the software project \( i \).

In statistics and probability theory, the median is the numerical value separating the higher half of a data sample, a population, or a probability distribution, from the lower half. The median of a finite list of numbers can be found by arranging all the observations from lowest value to highest value and picking the middle one. MdMRE value is the median value of the relative error. So it can be calculated by the equation:

\[ MdMRE = \text{median}(MRE_i). \quad (2.9) \]

MMRE value and MdMRE value are the evaluation criteria based on statistical learning, so they have strong noise resistance \([14]\). The smaller MMRE and MdMRE value are, the better performance the estimation model owns.

### 2.3.2 PRED (0.25)

The PRED (0.25) is the percentage of estimated effort that falls within 25% of the actual effort:

\[ \text{PRED} (0.25) = \frac{1}{n} \sum_{i=1}^{n} (MRE_i \leq 0.25). \quad (2.10) \]

If the PRED (0.25) value is larger, it means that the proportion of estimation cases whose relative error are below 25% is higher. Therefore, larger PRED (0.25) value indicates better performance.

### 2.4 Feature Selection

Feature selection, also known as feature extraction, is to select a feature subset from the original feature set. The estimation model built by the selected feature subset needs to perform better than that built by the original feature set.

Feature selection needs to eliminate the irrelevant features and the redundant features in the original feature set. Irrelevant features are those features that cannot do help to predict the cost of software project. By eliminating irrelevant features, the remaining features are useful for estimation model to predict the cost thus the estimation error will be smaller. Redundant features are those features that dependent on other features. One representative feature among those redundant features is enough because more features are not only helpless to predict more accurate cost but also bring higher computational time and space.

By applying feature selection, number of features will decrease while the accuracy will increase. The meaningful feature subset reduces the cost of computation and makes the
estimation model effective and efficient. Feature selection methods will be explored in chapter 3 and chapter 4.

2.5 Case Selection

The task of case selection is to find out one or more software projects from the historical data set to match the new software project.

In case selection, similarity measurement is introduced to measure the degree of similarity between two projects. The similarity is made up of two parts: local similarity and global similarity. The local similarity refers to the difference in one feature in two software projects while the global similarity needs to be calculated with global distance formula which operates over the local similarity of all features. The most similar projects are decided by the global similarity instead of local similarity.

There are several global similarity that are accepted by researchers such as “Manhattan Distance”, “Euclidean Distance”, “Jaccard Coefficient”, “Cosine Similarity” etc. Case selection will be explored in chapter 5.

2.6 Case Adaptation

Case adaptation is to employ the most similar projects found out in case selection to construct specific model to give the cost of new project. There are several case adaptation models in software cost estimation. “Closest Analogy” \(^{[11]}\) model only needs the most similar historical project and use its cost as the estimated cost of new project. “Mean Analogy” \(^{[5]}\) model uses the mean cost of the most similar N historical projects as the estimated cost of new project. “Median Analogy” model is a bit like the “Mean Analogy” model but it uses the median cost as the estimated cost. “Inverse Weighted Mean of Closest Analogy” \(^{[19]}\) model has to predefined the weight of each similar project. If a historical project is more similar to the new one, the weight of this project should be higher. Then the estimated cost of new project will be calculated by weighted average cost of N similar historical projects.

In “Closest Analogy” model, only one similar historical project is used to predict the cost, so it may bring accidental error when the selected historical project is not that “similar” as the new one. “Mean Analogy” model is better than “Closest Analogy” model because it employs more similar projects and adopts the mean value of each historical project to reduce the risk that “Closest Analogy” model exists. In this thesis, “Mean Analogy” model will be used as the case adaptation model.
Chapter 3. Sequential Search Feature Selection

3.1 Principle of Sequential Search Feature Selection

There are two kinds of features in the data set. The feature whose value is unknown and needed to be predicted is called estimated feature. The other features whose values are given and used to predict the value of estimated feature are called independent features. In software cost estimation, the cost of project is estimated feature while the other features used to predict the cost are the independent features.

In the feature selection, one candidate independent feature in the original feature set needs to satisfy two conditions in order to become a member of final feature subset \(^{[22]}^{[23]}^{[24]}\):

1. The independent feature is strongly relevant to estimated feature;
2. The independent feature is relevant with any other independent feature.

Suppose that RL (short for “relevance”) represents the relevance between one independent feature and the estimated feature and RD (short for “redundancy”) represents the redundancy between one independent feature and other independent features, then candidate independent feature \(F_i\) which satisfies the following expression will be selected as a member of feature subset:

\[
\text{MAX}\{\text{RL}(F_i) - \text{RD}(F_i)\}.
\]

(3.1)

The computational formulas for RL\((F_i)\) and RD\((F_i)\) will be given in the rest of this chapter.

3.2 Related Work

Mutual information can be used to measure the independence among random variables in information theory, thus it is practicable to employ it to measure the degree of relevance and redundancy between features in the sequential search feature selection.

Battiti et al \(^{[20]}\) proposes MIFS (Mutual Information Feature Selection) method. The expression is given as below:

\[
\text{MIFS} = I(C; f_i) - \beta \sum_{f_s \in S} I(f_i; f_s).
\]

(3.2)

\(I(C; f_i)\) is the mutual information between candidate independent feature \(f_i\) and the estimated feature \(C\), so it can represent the relevance between the independent feature and the estimated feature. \(\beta \sum_{f_s \in S} I(f_i; f_s)\) is the redundancy between independent feature \(f_i\) and the selected feature in the feature subset \(S\). \(\beta\) is a parameter used to adjust the impact of relevance and redundancy and its value range is \([0,1]\). If \(\beta\) is 0, then the expression value above is decided by the relevance part \(I(C; f_i)\). If \(\beta\) is 1, then the expression value above depends on the redundancy part \(\beta \sum_{f_s \in S} I(f_i; f_s)\). For each unselected independent feature, if the independent feature \(f_i\) makes the value of expression above larger than any other independent feature, then \(f_i\) are to be selected into the feature subset.
On the basis of MIFS method, Kwak and Choi et al.\(^{21}\) proposes the MIFS-U method using entropy to improve the redundancy part:

\[
\text{MIFS - U} = I(C; f_i) - \beta \sum_{f_s \in S} \frac{I(C; f_s)}{H(f_s)} I(f_i; f_s). \tag{3.3}
\]

\(H(f_s)\) is the entropy of selected feature \(f_s\). Both MIFS and MIFS-U methods have the same problem: the value of \(\beta\) has to be trained when dealing with different data set; the value of redundancy part keep increasing when more and more independent features are selected into the feature subset but the value of relevance part changes not much. Therefore, the impact of relevance part is much larger than the relevance part which results in selecting irredundant features but also irrelevant features.

In order to overcome these two disadvantages, Peng et al.\(^{22}\) proposes their mRMR (Max Relevance and Min Redundancy) method:

\[
mRMR = I(C, f_i) - \frac{1}{|S|} \sum_{f_s \in S} I(f_i, f_s). \tag{3.4}
\]

mRMR method replaces \(\beta\) with the \(|S|\) which is the number of selected features in the feature subset. In this way, the value of redundancy part can be regarded as the average redundancy value between candidate independent feature and each selected feature. Besides, the value of redundancy part will not keep increasing though the number of selected feature become more and more. mRMR method keeps good balance between relevance part and redundancy part.

Estevez et al.\(^{23}\) suggest to normalize the value of mutual information for restrict the value into the range \([0,1]\). Data normalization can eliminate the dimension. Thereby it makes data more comparable. They propose their NMIFS (Normalized MIFS) method:

\[
\text{NMIFS} = I(C, f_i) - \frac{1}{|S|} \sum_{f_s \in S} NI(f_i, f_s). \tag{3.5}
\]

\(NI(f_i, f_s)\) represents the normalized mutual information between features \(i\) and \(s\), and the equation for normalized mutual information is given as below:

\[
NI(f_i, f_s) = \frac{I(f_i, f_s)}{\min(H(f_i), H(f_s))}. \tag{3.6}
\]

Thang et al.\(^{24}\) propose INMIFS (Improved NMIFS) method based on the NMIFS:

\[
\text{INMIFS} = NI(C, f_i) - \frac{1}{|S|} \sum_{f_s \in S} NI(f_i, f_s). \tag{3.7}
\]

In NMIFS method, the value of redundancy part is restricted into the range \([0,1]\). However, the value of relevance part is not restricted into the same range. Sometimes, the value of relevance part is much larger than 1. In that case, the impact of relevance part is larger than that of
redundancy part. So INMIFS method also restricts the value of relevance part into \([0,1]\) in order to balance the relevance part and the redundancy part.

### 3.3 INMIFS in Software Cost Estimation

There are two methods in sequential search feature selection: filter method and wrapper method. Filter method evaluates the data set according to the property of data set without using specific model. Therefore filter method is independent of any prediction model. On the contrary, wrapper method evaluates the data set using a specific model \[^{25}\]. The performance of data set depends on the chosen prediction model. Filter methods are often more computationally efficient than wrapper methods while wrapper methods can yield more accurate prediction result than filter methods.

The analogy based sequential search feature selection scheme is shown in figure 3 \[^{26}\].

---

**Figure 3. Sequential search feature selection scheme.**

This scheme combines filter method and wrapper method. \(m\) feature subsets are selected in the filter method and the best one in the those \(m\) feature subsets is determined in the wrapper method. The best feature subset should yield smallest value of MMRE or highest value of PRED (0.25). The whole framework is shown as below:
In the filter method:

1. Initialization: set $F \leftarrow$ initial set of $n$ features; set $S \leftarrow$ empty set.
2. Computation of the MI value between each candidate feature and the response feature:
   For each $f_i \in F$, compute $I(C; f_i)$.
3. Selection of the first feature: Find the feature $f_i$ that maximize the $I(C; f_i)$, set $F \leftarrow F \setminus \{f_i\}$ and set $S \leftarrow \{f_i\}$.
4. Greedy selection: Repeat until $|S| = k$.
   a. Computation of MI between variables: For each pair of features $(f_i, f_s)$ with $f_i \in F$ and $f_s \in S$, compute $I(f_i, f_s)$ if it is not yet available.
   b. Selection of the next feature: Choose the feature $f_i$ that maximizes the value of equation as follows:

   $\max \{NI(C; f_i) - \frac{1}{|S|} \sum_{f_s \in S} NI(f_i; f_s)\}$

   then set $F \leftarrow F \setminus \{f_i\}$ and set $S \leftarrow \{f_i\}$.
5. Output the set $S$ containing the selected features.

In the wrapper method:
The task is to determine the optimal $m$ number. Suppose there are $n$ candidate features to be selected in the data set, the INMIFS method using incremental selection produces $n$ sequential feature sets $S_1 \subset S_2 \subset \cdots \subset S_m \subset \cdots \subset S_{n-1} \subset S_n$. Then compare all these $n$ sequential feature sets $S_1, \cdots, S_m, \cdots, S_n$ to find the set $S_m$ that can minimize the MMRE value of the training set. Finally, $m$ is the optimal number of features and the set $S_m$ is the optimal feature set.
Chapter 4. Clustering Feature Selection

4.1 Drawback of Sequential Search Feature Selection

Suppose that original feature set is $F$, the feature subset which is the result of feature selection is $S$ and the estimated feature is $C$. Then according to the sequence that the independent feature is selected (or eliminated), there are three kinds of sequential search feature selections, namely forward sequential search feature selection, backward sequential search feature selection and floating sequential search feature selection \[44\].

In forward sequential search feature selection, initialize the set $S=\emptyset$. In each iteration, independent feature $f \in F/S$ when $S$ is given, the feature $f_{\text{max}}$ which can maximize the value of relevance will be selected and change the value $S$ to $S = S \cup f_{\text{max}}$. In backward sequential search feature selection, initialize the set $S=F$. Then select the independent feature $f \in F$ when the value of expression $S/f$ is given. The feature $f_{\text{min}}$ that minimize the value of relevance will be eliminated from set $S$, namely change the value $S$ to $S = S/f_{\text{min}}$. Both forward and backward sequential search feature selection methods will result in “Nesting Effect” \[45\]. It refers that if an independent feature is selected (or eliminated) into $S$ (from $S$), then the following selected (or eliminated) independent feature will be affected by the already selected (or eliminated) features \[44\] \[45\]. There are two kinds of floating sequential search feature selection methods. As similar with the forward and backward sequential search feature selection, floating sequential search is made up of forward floating and backward floating. In forward floating, it firstly selects an independent feature that can maximize the value of relevance and then it will evaluate the performance of current feature subset to determine whether to eliminate a selected feature in the feature subset. Backward floating is more or less same as forward floating, but it firstly eliminates an independent feature and then determines whether to add a feature into the subset.

The feature selection methods in Chapter 3 belong to the forward sequential search. Due to the “Nesting Effect”, these feature selection methods may not yield accurate estimating results. Floating sequential search feature selection can overcome the “Nesting Effect” to some extent but with the cost of very high computational spend. In some cases, it is not practical to employ floating search to solve the problems. Therefore, it is necessary to propose a new kind of feature selection to solve the problem of “Nesting Effect” meanwhile reduce the computational cost.

4.2 Supervised and Unsupervised Learning

There are many kinds of learning methods in data mining and machine learning such as “Analytic Learning”, “Analogy Learning” and “Sample Learning”. Generally Speaking, the most valuable one is the “Sample Learning”. Supervised learning and unsupervised learning are two popular sample learning methods \[29\].

In the supervised learning, it trains the data set to obtain an optimal model for prediction and uses this model to output the result when the input data is given. Supervised learning is often used for classification. KNN and SVM are the typical application of supervised learning \[38\].
On the contrary, the unsupervised learning does not have to train data to construct model but employ the data directly to discover the structural knowledge behind the data\textsuperscript{[38]}. Clustering is one of the most typical applications of unsupervised learning.

### 4.3 Principle of Clustering Feature Selection

Compared to the sequential search feature selection methods, clustering feature selection methods can avoid “Nesting Effect” much better. In addition, clustering feature selection methods reduce much computational cost when compared to the floating sequential search feature selection methods.

The basic idea behind clustering feature selection methods is similar to the data clustering. It groups the similar features into several clusters and then selects representative feature from each cluster. It is a totally new schema of feature selection and is able to lower the estimated variance value. Besides, clustering feature selection is more stable and scalable.

Estimated feature is not employed in the feature clustering and only the independent features are used to process the feature clustering. Based on the idea of clustering, there are three steps in the clustering feature selection in software cost estimation:

1. Define the feature similarity and group the independent features into several clusters;
2. Pick up one independent feature from each clusters as representative feature and add it into feature subset;
3. Evaluate each feature subset using the estimating model and select the feature subset that can estimate most accurate cost as the final feature subset of feature selection.

### 4.4 Related Work

Zhang et al\textsuperscript{[27]} propose the FSA method and define the RD (relevance degree) as the feature similarity measurement:

$$\text{RD}(f_i, f_j) = \frac{2I(f_i,f_j)}{H(f_i)+H(f_j)}.$$ \hspace{1cm} (4.1)

Meanwhile, FSA method predefines two threshold values $\delta$ and $K$ which represent the cluster relevance degree and the number of clusters. The clustering process stops when the relevance value is larger than $\delta$ or the current cluster number is larger than $K$. FSA also defines the RA (representative ability) of each independent feature in one cluster. However, FSA still has two major disadvantages. First, the predefined values of $\delta$ and $K$ cannot guarantee to obtain accurate results in different data set. Second, it never considers the relevance between independent feature and the estimated feature when defining the RA. The second one may lead to large estimating error because it may keep the irrelevant features to build the estimating model.

Li et al\textsuperscript{[28]} propose the FSFC method. FSFC defines the feature similarity based on MICI:

$$C(S_i,S_j) = \min(D(S_i,S_j),D(S_j,S_i)).$$ \hspace{1cm} (4.2)

$$D(S_i,S_j) = \frac{1}{m} \sum_{i=1}^{m} \sum_{x_i \in S_i} \min\{MICI(x_i,x_j),x_j \in S_j\}. \hspace{1cm} (4.3)$$
FSFC method also predefines K as the number of cluster. When feature clustering is completed, it calculates the sum of distance between one independent feature and the other independent features in the same cluster. If the independent feature \( f_i \) can minimize the sum value, then \( f_i \) is selected as the representative feature. However, FSFC method has the same problems as the FSA, namely that the predefined K may not be suitable for all data set and the representative feature has nothing to do with the estimated feature.

In summary, both FSA and FSFC methods have two major drawbacks here:

1. They use only unsupervised learning approaches in feature clustering without considering the relevance between independent feature and the estimated feature, which will result in picking up irrelevant features to build estimating model;
2. They predefine some threshold value but cannot guarantee these values are suitable and effective in different data set.

In the following pages in this chapter, a clustering feature selection method will be proposed to overcome the problem mentioned above. It combines the supervised learning and unsupervised learning so that the feature subset kept by the proposed method is relevant with the estimated feature. In addition, the new method employs wrapper method in order to select the optimal feature subset without predefining \( \delta \) and K value.

4.5 Hierarchical Clustering

There are two types of clustering in data mining, namely partition clustering and hierarchical clustering [29]. Partition clustering simply group the data objects into several non-overlapping clusters and make sure that the data object stays in only one cluster. Hierarchical clustering is nested and organized as a tree. All the nodes in the tree are merged by the children nodes except the leaf nodes and the root node contains all the data objects.

![Figure 4. Tree diagram of hierarchical clustering](image)
4.6 Feature Selection Based on Hierarchical Clustering

4.6.1 Feature Similarity

Feature similarity is one of the core parts in feature selection. The proposed hierarchical clustering feature selection method employs normalized mutual information as the feature similarity measurement:

\[
NI(f_i, f_j) = \frac{I(f_i, f_j)}{\min \{H(f_i), H(f_j)\}},
\]

(4.4)

Normalized mutual information is able to eliminate the bias in calculation of mutual information [23].

4.6.2 Feature Clustering

Feature dissimilarity is crucial in feature clustering. The feature dissimilarity is always related to the feature similarity. Suppose the feature similarity is \( S \) then the dissimilarity can be defined as \( D = 1 - S \).

In the hierarchical clustering, all the independent features will be grouped into several clusters except the estimated feature. According to the definition of feature dissimilarity, the feature similarity of proposed method can be given as below:

\[
FDis(f_i, f_j) = 1 - NI(f_i, f_j).
\]

(4.5)

Two nearest neighboring clusters have to be merged into one larger cluster until all the clusters are combined into one cluster. When measuring the distance of nearest neighboring clusters, there are three kinds of distances as solutions, namely the single link, complete link and the group average. In single link mode, the distance of two clusters can be regarded as the shortest distance between two data objects in each cluster. In complete link mode, the distance of two clusters can be regarded as the longest distance between two data objects in each cluster. In group average mode, the distance is defined as the average distance between all the data objects in each cluster. Because of the excellent resistance of noise data, complete link is more suitable in software cost estimation data sets. The distance of complete link mode is given as below:

\[
CDis(C_x, C_y) = \max \{FDis(f_i, f_j), f_i \in C_x \text{ and } f_j \in C_y\}.
\]

(4.6)

Here the \( CDis(C_x, C_y) \) represents the distance between cluster \( X \) and cluster \( Y \).

4.6.3 Number of Representative Feature

Hierarchical clustering feature selection combines filter method and wrapper method as the
sequential search feature selection. In filter method, independent features are clustered and those representative features form the candidate feature subsets. In wrapper method, the candidate feature subsets are evaluated in the estimating model using evaluation criteria like MMRE and PRED (0.25). The feature subset that can yield best performance will be chosen as the final result of clustering feature selection, and the number of features in the feature subset is determined.

4.6.4 Choice of Best Number

The proposed hierarchical clustering feature selection method needs to select the representative feature from the original feature set. The order of selecting representative feature from clusters is opposite to the order of clustering. It picks up the representative feature from the bottom to the top. The first pick is from the root cluster which is the largest cluster containing all the features. The second pick is from the cluster which is formed before the root cluster but after the other clusters. The rest can be done in the same manner. The condition for selecting representative feature is that the independent feature can maximize the relevance value with estimated feature:

$$\text{MAX}\{I(f_i, e)\}. \quad (4.7)$$

In the above expression, $f_i$ is the independent feature and the $e$ is the estimated feature.

The process of hierarchical clustering can be described in the following figure. Initialize each feature as a cluster. Merge two nearest neighboring clusters into one larger cluster. For example, cluster C and D will be merged into a larger cluster marked with number 1. After four times merging, the root cluster marked with number 4 contains all the independent features. The first round pick will start from cluster number 4. The representative feature of cluster number 4 is selected from independent feature A, B, C, D and E. Suppose A is most relevant with the estimated feature $e$ and then A is selected as the representative feature of cluster number 4. The next round pick will select features in the cluster number 3. Though A is more relevant with the estimated feature $e$ but A is selected in the first round, so B is selected as the representative feature of cluster number 3. After two round picks, there are two feature subsets, namely subset $S_1 = \{A\}$ and $S_2 = \{A, B\}$.

The selection of representative feature takes the relevance between independent feature and estimated feature into consideration and enable the feature subset is useful for building estimating model, so it will improve the accuracy of prediction.
4.6.5 Schema of HFSFC

The proposed hierarchical clustering feature selection employs both supervised and unsupervised learning, so the name of it is HFSFC (Hybrid Feature Selection using Feature Clustering). The schema is given as below:

---

**Hybrid Feature Selection using Feature Clustering (HFSFC)**

**Input:** Original feature set with $n$ features $F = \{f_1, f_2, \ldots, f_n\}$, predicted variable $f_c$.

**Output:** the optimal feature subset $S$;

**Step 1:** $S = \emptyset$
- Calculate pair-wise feature distance

**Step 2:** $C_i = f_i$, each feature in $F$ represents a cluster.

**Step 3:** Repeat
- Merge $C_i$ and $C_j$ if the their cluster distance is minimal
- Until all clusters are merged into one cluster.

**Step 4:** For $K = 1$ to $n$
- Recognize the top K clusters $S_k$ from the hierarchical clustering result.
  - $FS_k = \emptyset$
  - For each cluster $C_x$ in $S_k$
    - The unselected feature $f_x$ that can maximize the feature similarity with the predicted variable $f_c$ is selected as a representative feature
    - $FS_k = FS_k \cup f_x$
  - EndFor
- Evaluate the performance of subset $FS_k$
- EndFor

**Step 5:** The feature subset $FS_o$ that can achieve best performance is kept as the final result of the hybrid feature selection method

$$S = FS_o$$
4.6.6 Computational Complexity of HFSFC

Assume that the original data set contains $n$ features. It has computational complexity $O(n^2)$ in the filter approach for feature clustering and $O(n^2)$ in the wrapper approach for determining the optimal number of representative features. So the total complexity of HFSFC is $O(n^2) = O(n^2) + O(n^2)$.

4.6.7 Limitation of HFSFC

There is still one limitation in this algorithm. If the data set contains $n$ features, then we need to yield $n$ feature subset and have to evaluate these subsets one by one to determine the best one as the final result.
Chapter 5. Feature Weight in Case Selection

5.1 Principle of Feature Weight

In feature selection, it selects the irredundant feature subset that is relevant to the estimated feature. However, the selected features in the feature subset contribute differently to the estimation of cost. Some features are more important than the others in the contribution of estimation. Therefore, they should have more power when constructing the global distance using local distance. So, it is necessary to introduce the feature weight in case selection to reflect the impact of each selected feature.

The principle of feature weight is rather simple. If one selected feature is more relevant to the estimated feature, its feature weight is larger.

5.2 Symmetric Uncertainty

Symmetric uncertainty is a concept based on mutual information. The formula for symmetric uncertainty is given as below:

\[
SU(X, Y) = \frac{2 \cdot \text{Gain}(X|Y)}{H(X) + H(Y)}.
\]

Gain(X,Y) = \( H(X) - H(X|Y) = H(Y) - H(Y|X) \).

H(X) and H(Y) represent the entropy of random variables X and Y while H(X|Y) and H(Y|X) are the conditional entropy. The information gain in the formula above is the mutual information between random variable X and Y, and the symmetric uncertainty is the normalization of the mutual information.

Mutual information can be the measurement of relevance of random variable X and Y. But sometimes the value of mutual information is large and the entropy values of random variables are also large, then the mutual information value cannot reflect the relationship of random variable X and Y.

5.3 Feature Weight Based on Symmetric Uncertainty

Based on the introduction of symmetric uncertainty above, the definition of feature weight is given as below:

\[
w_k = SU(k, e).
\]

In the equation above, k represents the kth feature, e represents the estimated feature and SU(k,e) is the symmetric uncertainty value of kth feature and the estimated feature while \( w_k \) is the feature weight of kth feature.
5.4 Global Distance and Local Distance

There are many famous global distance formulas such as “Manhattan Distance”, “Euclidean Distance”, “Jaccard Coefficient” and “Cosine Similarity”.

Researches \cite{5,16} indicate that Euclidean distance outperform other solutions in software cost estimation. The Euclidean distance between two projects $i$ and $j$ can be written as follows:

$$D_{ij} = \sqrt{\sum_{k=1}^{n} w_k LDis(f_{ik}, f_{jk})}. \quad (5.4)$$

$$LDis(f_{ik}, f_{jk}) = \begin{cases} (f_{ik} - f_{jk})^2, & \text{if } f_{ik} \text{ and } f_{jk} \text{ are numeric} \\ 1, & \text{if } f_{ik}, f_{jk} \text{ are nominal and } f_{ik} \neq f_{jk} \\ 0, & \text{if } f_{ik}, f_{jk} \text{ are nominal and } f_{ik} = f_{jk} \end{cases}. \quad (5.5)$$

In the equation above, $f_{ik}, f_{jk}$ represent the value of $k$th feature in software project $i$ and $j$, respectively. Here if the $k$th feature is numeric data, then the local distance is the square of difference value. If the $k$th feature is nominal data, we only need to consider whether $f_{ik}, f_{jk}$ are equal. If equal, then the local distance is 0. Otherwise, the local distance is 1.

In global distance, each selected independent feature has different impact on the estimated feature. The independent feature which is more important to the estimated feature should have larger feature weight. Therefore, the feature weight defined above can be used to improve the equation:

$$GDis(i,j) = \sqrt{\sum_{k=1}^{n} SU(k,e) \times LDis(f_{ik}, f_{jk})}. \quad (5.6)$$
Chapter 6. Experiment and Analysis

6.1 Data Set in the Experiment

In software cost estimation, ISBSG (International Software Benchmarking Standard Group) \[30\] data set and Desharnais \[31\] data set are two typical data sets. ISBSG data set is paid by my supervisor to use in the experiment while the Desharnais data set is free.

6.1.1 Data Type

There are two kinds of data types in ISBSG and Desharnais data sets, namely nominal data and numeric data. Nominal data is mainly used to represent qualitative value and is not suitable for calculation. For example, the post code for different cities and blue, red, green for different colors are nominal data. Numeric data is mainly used to represent quantitative value and is calculable. For example, the weight of fruits and temperature of a day are numeric data.

6.1.2 ISBSG Data Set

ISBSG Release 8 data set contains 2008 real records of software projects which come from several industry fields. All the records in data set are rated in 4 classes. Class A records are the most reliable and useful data for doing software cost estimation. The whole data set contains 608 A-rated records with 50 independent features and 2 estimated features. By conducting data preprocessing \[32\], there are only 345 records with 11 features (10 independent features and 1 estimated feature).

<table>
<thead>
<tr>
<th>J</th>
<th>ISBSG-Project ID</th>
<th>Date</th>
<th>Derived Code Function Point</th>
<th>Value of FP Standard</th>
<th>Counting Tactic</th>
<th>Reference Tables</th>
<th>Appraisal Summary View</th>
<th>Recording Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10001</td>
<td>B</td>
<td>NESMA</td>
<td>254</td>
<td>NESMA</td>
<td>Value-manual</td>
<td>Combined all tables into</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>10012</td>
<td>B</td>
<td>IPUG</td>
<td>74</td>
<td>0.98</td>
<td>IPUG 4</td>
<td>Supported by a Counted set of tables</td>
<td>1100</td>
</tr>
<tr>
<td>3</td>
<td>10015</td>
<td>B</td>
<td>IPUG</td>
<td>479</td>
<td>1.25</td>
<td>IPUG 4</td>
<td>Supported by a Counted set of tables</td>
<td>22000</td>
</tr>
<tr>
<td>4</td>
<td>10036</td>
<td>B</td>
<td>CFP</td>
<td>183</td>
<td>COSMIC-PPF</td>
<td>460</td>
<td>Staff hours (recorded)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10039</td>
<td>B</td>
<td>IPUG</td>
<td>179</td>
<td>IPUG 4</td>
<td>271</td>
<td>Staff hours (recorded)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10047</td>
<td>B</td>
<td>IPUG</td>
<td>832</td>
<td>1.14</td>
<td>IPUG 4</td>
<td>20975</td>
<td>Staff hours (recorded)</td>
</tr>
<tr>
<td>7</td>
<td>10052</td>
<td>B</td>
<td>IPUG</td>
<td>163</td>
<td>1.02</td>
<td>IPUG 4</td>
<td>709</td>
<td>Staff hours (recorded)</td>
</tr>
<tr>
<td>8</td>
<td>10054</td>
<td>C</td>
<td>IPUG</td>
<td>196</td>
<td>2560</td>
<td>Staff hours (recorded)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10056</td>
<td>B</td>
<td>IPUG</td>
<td>135</td>
<td>1.18</td>
<td>IPUG 4</td>
<td>7290</td>
<td>Staff hours (recorded)</td>
</tr>
<tr>
<td>10</td>
<td>10074</td>
<td>B</td>
<td>IPUG</td>
<td>460</td>
<td>1.00</td>
<td>FPMA-3</td>
<td>1567</td>
<td>Staff hours (recorded)</td>
</tr>
<tr>
<td>11</td>
<td>10075</td>
<td>A</td>
<td>IPUG</td>
<td>1587</td>
<td>IPUG 4</td>
<td>Supported by a Counted set of tables</td>
<td>7490</td>
<td>Productive time only (s)</td>
</tr>
<tr>
<td>12</td>
<td>10116</td>
<td>B</td>
<td>IPUG</td>
<td>779</td>
<td>IPUG 4</td>
<td>Supported by a tool</td>
<td>25040</td>
<td>Staff hours (recorded)</td>
</tr>
<tr>
<td>13</td>
<td>10130</td>
<td>B</td>
<td>IPUG</td>
<td>294</td>
<td>IPUG 4</td>
<td>7791</td>
<td>Staff hours (recorded)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>10131</td>
<td>B</td>
<td>IPUG</td>
<td>90</td>
<td>1.12</td>
<td>IPUG 4</td>
<td>2879</td>
<td>Staff hours (recorded)</td>
</tr>
<tr>
<td>15</td>
<td>10145</td>
<td>A</td>
<td>IPUG</td>
<td>250</td>
<td>1.01</td>
<td>IPUG 4</td>
<td>2924</td>
<td>Productive time only (s)</td>
</tr>
<tr>
<td>16</td>
<td>10151</td>
<td>B</td>
<td>IPUG</td>
<td>174</td>
<td>1.01</td>
<td>IPUG 4</td>
<td>Supported by a Counted set of tables</td>
<td>1976</td>
</tr>
<tr>
<td>17</td>
<td>10153</td>
<td>A</td>
<td>IPUG</td>
<td>48</td>
<td>0.97</td>
<td>IPUG 4</td>
<td>Supported by a Counted set of tables</td>
<td>646</td>
</tr>
<tr>
<td>18</td>
<td>10174</td>
<td>B</td>
<td>IPUG</td>
<td>562</td>
<td>1.00</td>
<td>FPMA-3</td>
<td>2623</td>
<td>Staff hours (recorded)</td>
</tr>
<tr>
<td>19</td>
<td>10181</td>
<td>B</td>
<td>NESMA</td>
<td>338</td>
<td>NESMA</td>
<td>Value-manual</td>
<td>Combined all tables into</td>
<td>646</td>
</tr>
<tr>
<td>20</td>
<td>10189</td>
<td>A</td>
<td>IPUG</td>
<td>280</td>
<td>1.07</td>
<td>IPUG 4</td>
<td>4150</td>
<td>Productive time only (s)</td>
</tr>
<tr>
<td>21</td>
<td>10203</td>
<td>C</td>
<td>IPUG</td>
<td>90</td>
<td>1.12</td>
<td>IPUG 4</td>
<td>4150</td>
<td>Productive time only (s)</td>
</tr>
<tr>
<td>22</td>
<td>10212</td>
<td>B</td>
<td>IPUG</td>
<td>152</td>
<td>0.93</td>
<td>IPUG 4</td>
<td>Supported by a Counted set of tables</td>
<td>2019</td>
</tr>
<tr>
<td>23</td>
<td>10216</td>
<td>A</td>
<td>IPUG</td>
<td>203</td>
<td>1.19</td>
<td>IPUG 4</td>
<td>420</td>
<td>Staff hours (recorded)</td>
</tr>
<tr>
<td>24</td>
<td>10323</td>
<td>B</td>
<td>IPUG</td>
<td>252</td>
<td>1.12</td>
<td>Addendum to Supported by a Counted set of tables</td>
<td>3238</td>
<td>Staff hours (recorded)</td>
</tr>
<tr>
<td>25</td>
<td>10357</td>
<td>A</td>
<td>IPUG</td>
<td>292</td>
<td>0.95</td>
<td>Addendum to Supported by a Counted set of tables</td>
<td>294</td>
<td>Staff hours (recorded)</td>
</tr>
<tr>
<td>26</td>
<td>10365</td>
<td>A</td>
<td>IPUG</td>
<td>83</td>
<td>0.95</td>
<td>Addendum to Supported by a Counted set of tables</td>
<td>1333</td>
<td>Staff hours (recorded)</td>
</tr>
<tr>
<td>27</td>
<td>10370</td>
<td>B</td>
<td>IPUG</td>
<td>217</td>
<td>1.25</td>
<td>IPUG 4</td>
<td>Supported by a Counted set of tables</td>
<td>20800</td>
</tr>
<tr>
<td>28</td>
<td>10316</td>
<td>A</td>
<td>IPUG</td>
<td>79</td>
<td>1.02</td>
<td>Addendum to Supported by a Counted set of tables</td>
<td>1007</td>
<td>Staff hours (recorded)</td>
</tr>
</tbody>
</table>

Figure 6.1 ISBSG Release 8 data set
Feature Name | Data Type | Meaning in Software Project
--- | --- | ---
CouTech | Nominal | Technique for calculating function points
DevType | Nominal | Development type: new, improved or redevelopment
FP | Numeric | Function point
RLevel | Numeric | Available resource level
PrgLang | Nominal | Program language
DevPlatform | Nominal | Development platform
Time | Numeric | Estimated time for development
MethAcquired | Nominal | Purchase or research independently
OrgType | Nominal | Use database or not to organize data
Method | Nominal | Methods for recording workload
SWE | Numeric | Software cost

Table 6.1 Features in ISBSG data set and its meaning in software project

There are seven nominal independent features and three independent numeric features while 1 numeric estimated feature "SWE" in the ISBSG data set.

6.1.3 Desharnais Data Set

Desharnais data set contains much less records than ISBSG R8 data set. The records of Desharnais data set come from one software company. In Desharnais data set, there are 81 records of historical software projects but 4 of them contain missing fields. Therefore, only 77 records with complete field data are kept for experiment. The original Desharnais data set includes 11 features (10 independent features and 1 estimated feature).

```
10, 3, 4, 83, 4, 2422, 78, 58, 116, 24, 103, 1
11, 4, 1, 34, 21, 4067, 167, 99, 265, 24, 237, 1
12, 2, 1, 84, 17, 9051, 146, 118, 258, 49, 271, 1
13, 1, 1, 84, 3, 2282, 38, 72, 165, 19, 88, 1
14, 2, 4, 85, 8, 4173, 162, 61, 233, 32, 316, 1
15, 4, 4, 85, 9, 4877, 223, 121, 344, 28, 320, 1
16, 3, 2, 85, 5, 1017, 119, 46, 157, 26, 152, 1
17, 4, 3, 85, 8, 3132, 57, 63, 108, 43, 103, 1
18, 4, 4, 86, 14, 3437, 68, 316, 304, 20, 326, 2
19, 3, 5, 87, 14, 4699, 3, 396, 395, 21, 340, 2
20, 4, 2, 86, 5, 840, 58, 34, 92, 29, 88, 1
21, 4, 4, 85, 12, 14973, 318, 269, 567, 34, 581, 2
22, 2, 4, 85, 18, 5180, 88, 170, 258, 34, 255, 1
23, 2, 4, 86, 5, 5775, 306, 132, 438, 37, 447, 1
24, 4, 1, 87, 20, 10577, 364, 78, 382, 39, 397, 1
25, 1, 4, 86, 8, 3983, 89, 200, 289, 53, 285, 1
26, 4, 1, 85, 14, 3164, 85, 230, 316, 33, 310, 1
27, 2, 0, 86, 5, 3542, 71, 235, 306, 37, 312, 1
28, 3, 1, 85, 14, 4277, 148, 324, 472, 39, 491, 1
```

Figure 6.2 Desharnais data set
<table>
<thead>
<tr>
<th>Feature Name</th>
<th>Data Type</th>
<th>Meaning in Software Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>TeamExp</td>
<td>Numeric</td>
<td>Project experience of team</td>
</tr>
<tr>
<td>ManagerExp</td>
<td>Numeric</td>
<td>Project experience of manager</td>
</tr>
<tr>
<td>YearEnd</td>
<td>Nominal</td>
<td>End year of project</td>
</tr>
<tr>
<td>Length</td>
<td>Numeric</td>
<td>Required time for the project</td>
</tr>
<tr>
<td>Language</td>
<td>Nominal</td>
<td>Program language</td>
</tr>
<tr>
<td>Transactions</td>
<td>Nominal</td>
<td>Transaction number in the project</td>
</tr>
<tr>
<td>Entities</td>
<td>Numeric</td>
<td>Entity number in the project</td>
</tr>
<tr>
<td>PointNonAdjust</td>
<td>Numeric</td>
<td>Function point number before adjustment</td>
</tr>
<tr>
<td>Adjustment</td>
<td>Numeric</td>
<td>Factor for function adjustment</td>
</tr>
<tr>
<td>PointAdjust</td>
<td>Numeric</td>
<td>Function point number after adjustment</td>
</tr>
<tr>
<td>Effort</td>
<td>Numeric</td>
<td>Software cost</td>
</tr>
</tbody>
</table>

Table 6.2 Features in Desharnais data set and its meaning in software project

Data type in Desharnais data set is quite different from ISBSG R8 data set. It contains 8 independent numeric features, 2 independent nominal features, and 1 numeric feature “Effort”.

6.2 Parameter Settings

6.2.1 Data Standardization

In the experiment of software cost estimation, it is quite necessary to carry out data standardization and normalize the value range to \([0, 1]\). The formula for standardization is given as below:

\[
\text{NewValue} = \frac{\text{OldValue} - \text{MinValue}}{\text{MaxValue} - \text{MinValue}}
\]  

(6.1)

Oldvalue and newvalue in the above equation represent the feature value before and after the standardization, respectively. Maxvalue and minvalue are the maximal value and minimal value of one specific feature in the data set.

6.2.2 K-Fold Cross Validation

Cross validation is a statistical analysis for evaluating the performance of classifier. The basic idea behind it is to divide the data set into two parts, one for training and the other for testing. The training set is used to train the classifier model and the testing set is used to evaluate the training model. In this thesis, 3-fold cross validation is employed. It splits the data into 3 equal parts. Two of three parts are used as the training set and the remaining one is used as the testing set. Training set will be used to construct the estimating model while the testing set will be used to
evaluate the performance of the model.

6.2.3 K Nearest Neighbor

In the case selection, one or more historical software projects are needed to estimate the cost of new project. Auer [39], Chiu et al [16] and Walkerden et al [40] employ K=1 in the closet analogy. Others like Jorgenson [41], Menders [42] and Shepperd et al [5] agree that K=2, 3, 4 can yield better result in the closet analogy. So, in this thesis, K value will be 1,2,3,4 and 5 in order to cover the recommended K value in others’ research work. The experiment will evaluate the performance of estimating model when using different K value.

6.2.4 Mean of Closet Analogy

In the case adaptation, it employs cost of historical software projects to estimate the cost of new project. In this thesis, mean of closet analogy is used to estimate the cost, and the formula is given as below:

\[
EE = \frac{1}{K} \sum_{i=1}^{K} HE_i. \quad (6.2)
\]

EE represents the estimated cost of new project while HE$_i$ represents the ith cost of historical project.

6.3 Experiment Platform and Tools

The platform for experiments in this thesis is R [33]. It can be used to conduct statistical experiment and visualize the results. The program language here is R and it is a script language. R language is very efficient in vector and matrix calculation after some optimization, so it is suitable for large scale data processing. R language contains built-in packages for calculation as well as visualization. In addition, programmers are able to install open source extensible packages to customize specific calculation.

The hardware for experiments includes one x86 PC which contains a 2.6GHZ CPU and a 2G memory.

6.4 Experiment Design

The experiment design consists of following four parts:
(1) Compare the performance of sequential search feature selection methods INMIFS, NMIFS and mRMRFS in software cost estimation data sets.
(2) Evaluate the performance of proposed HFSFC method with different parameter settings.
(3) Compare the HFSFC method with sequential search feature selection method INMIFS and clustering feature selection method FSFC.
(4) Evaluate the performance of HFSFC method with feature weight.
6.5 Experiment of Sequential Search Feature Selection

<table>
<thead>
<tr>
<th>Methods</th>
<th>K value</th>
<th>MMRE</th>
<th>PRED (0.25)</th>
<th>MdMRE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INMIFS</td>
<td>K=1</td>
<td>1.3230</td>
<td>0.2303</td>
<td>0.5927</td>
</tr>
<tr>
<td></td>
<td>K=2</td>
<td>1.4641</td>
<td>0.2394</td>
<td>0.5641</td>
</tr>
<tr>
<td></td>
<td>K=3</td>
<td>1.4786</td>
<td>0.2333</td>
<td>0.5854</td>
</tr>
<tr>
<td></td>
<td>K=4</td>
<td>1.4199</td>
<td>0.2515</td>
<td>0.5539</td>
</tr>
<tr>
<td></td>
<td>K=5</td>
<td>1.4963</td>
<td>0.2789</td>
<td>0.4930</td>
</tr>
<tr>
<td>NMIFS</td>
<td>K=1</td>
<td>1.5038</td>
<td>0.2000</td>
<td>0.6177</td>
</tr>
<tr>
<td></td>
<td>K=2</td>
<td>1.1990</td>
<td>0.2152</td>
<td>0.6043</td>
</tr>
<tr>
<td></td>
<td>K=3</td>
<td>1.5197</td>
<td>0.2303</td>
<td>0.5779</td>
</tr>
<tr>
<td></td>
<td>K=4</td>
<td>1.3951</td>
<td>0.2456</td>
<td>0.5843</td>
</tr>
<tr>
<td></td>
<td>K=5</td>
<td>1.7999</td>
<td>0.2545</td>
<td>0.5669</td>
</tr>
<tr>
<td>mRMRFS</td>
<td>K=1</td>
<td>1.3396</td>
<td>0.1969</td>
<td>0.5926</td>
</tr>
<tr>
<td></td>
<td>K=2</td>
<td>1.2929</td>
<td>0.2333</td>
<td>0.5490</td>
</tr>
<tr>
<td></td>
<td>K=3</td>
<td>1.6263</td>
<td>0.2303</td>
<td>0.5823</td>
</tr>
<tr>
<td></td>
<td>K=4</td>
<td>1.4002</td>
<td>0.2454</td>
<td>0.5331</td>
</tr>
<tr>
<td></td>
<td>K=5</td>
<td>1.6670</td>
<td>0.2515</td>
<td>0.5242</td>
</tr>
</tbody>
</table>

Table 6.3 Experiment results of ISBSG data set

The experiment result of ISBSG data set is shown in the Table 6.3. It can be seen from the result that the K value of nearest neighbors has impact on the performance. When K is 5, the INMIFS method can achieve highest PRED (0.25) value 0.2789. The INMIFS method performs 10.89% and 9.58% better than mRMRFS method and NMIFS method, respectively when K value is 5. When considering the MMRE value, INMIFS method obtains 1.4963 when K value is 5, which is 10.24% and 16.87% lower than mRMRFS method and NMIFS method.
<table>
<thead>
<tr>
<th>Methods</th>
<th>K value</th>
<th>MMRE</th>
<th>PRED (0.25)</th>
<th>MdMRE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INMIFS</strong></td>
<td>K=1</td>
<td>0.7335</td>
<td>0.3718</td>
<td>0.3452</td>
</tr>
<tr>
<td></td>
<td>K=2</td>
<td>0.6303</td>
<td>0.3846</td>
<td>0.3885</td>
</tr>
<tr>
<td></td>
<td>K=3</td>
<td>0.3951</td>
<td>0.4893</td>
<td>0.3317</td>
</tr>
<tr>
<td></td>
<td>K=4</td>
<td>0.5567</td>
<td>0.4487</td>
<td>0.2786</td>
</tr>
<tr>
<td></td>
<td>K=5</td>
<td>0.4508</td>
<td>0.3974</td>
<td>0.3354</td>
</tr>
<tr>
<td><strong>NMIFS</strong></td>
<td>K=1</td>
<td>0.7200</td>
<td>0.3205</td>
<td>0.3934</td>
</tr>
<tr>
<td></td>
<td>K=2</td>
<td>0.4435</td>
<td>0.3717</td>
<td>0.3419</td>
</tr>
<tr>
<td></td>
<td>K=3</td>
<td>0.5494</td>
<td>0.4615</td>
<td>0.2846</td>
</tr>
<tr>
<td></td>
<td>K=4</td>
<td>0.5499</td>
<td>0.3718</td>
<td>0.9400</td>
</tr>
<tr>
<td></td>
<td>K=5</td>
<td>0.7960</td>
<td>0.3333</td>
<td>0.3762</td>
</tr>
<tr>
<td><strong>mRMRFS</strong></td>
<td>K=1</td>
<td>0.6779</td>
<td>0.3589</td>
<td>0.3445</td>
</tr>
<tr>
<td></td>
<td>K=2</td>
<td>0.4803</td>
<td>0.3718</td>
<td>0.3267</td>
</tr>
<tr>
<td></td>
<td>K=3</td>
<td>0.5202</td>
<td>0.4359</td>
<td>0.3070</td>
</tr>
<tr>
<td></td>
<td>K=4</td>
<td>0.6226</td>
<td>0.3974</td>
<td>0.3640</td>
</tr>
<tr>
<td></td>
<td>K=5</td>
<td>0.5500</td>
<td>0.3077</td>
<td>0.3827</td>
</tr>
</tbody>
</table>

Table 6.4 Experiment results of Desharnais data set

The experiment result of Desharnais is shown in the Table 6.4. K value also influences the result. When K value is 3, PRED (0.25) of INMIFS method achieves the peak value 0.4893, which is 12.05% an 6.02% higher than mRMRFS method and NMIFS method, respectively. Meanwhile, the MMRE value of INMIFS method is 0.3951, which is also lower than that of mRMRFS method and NMIFS method.
6.6 Experiment of Hierarchical Clustering Feature Selection

6.6.1 Different Number of Representative Features

After applying hierarchical clustering on \( n \) features, it will obtain \( n-1 \) clusters. In each cluster, one representative feature will be picked up to form the feature subset. Different number of representative feature will lead to different performance of estimating model.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Representative Feature</th>
<th>KCluster</th>
<th>MMRE</th>
<th>PRED(0.25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISBSG</td>
<td>3</td>
<td>1</td>
<td>1.5697</td>
<td>0.1914</td>
</tr>
<tr>
<td></td>
<td>9 3</td>
<td>2</td>
<td>1.6033</td>
<td>0.2340</td>
</tr>
<tr>
<td></td>
<td>9 3 2</td>
<td>3</td>
<td>1.6018</td>
<td>0.2462</td>
</tr>
<tr>
<td></td>
<td>9 3 2 6</td>
<td>4</td>
<td>1.4886</td>
<td>0.2462</td>
</tr>
<tr>
<td></td>
<td>9 3 2 6 5</td>
<td>5</td>
<td>1.2824</td>
<td>0.2492</td>
</tr>
<tr>
<td></td>
<td>9 3 2 6 5 7</td>
<td>6</td>
<td>1.3937</td>
<td>0.2627</td>
</tr>
<tr>
<td></td>
<td>9 3 2 6 5 7 10</td>
<td>7</td>
<td>2.2082</td>
<td>0.2522</td>
</tr>
<tr>
<td></td>
<td>9 3 2 6 5 7 10 8</td>
<td>8</td>
<td>2.2174</td>
<td>0.2462</td>
</tr>
<tr>
<td></td>
<td>9 3 2 6 5 7 10 8 1</td>
<td>9</td>
<td>2.3197</td>
<td>0.2317</td>
</tr>
</tbody>
</table>

Table 6.5 Results of different number of representative features in ISBSG

It can be seen from Table 6.5 that KCluster value is important to the ISBSG data set. When KCluster value is 6, the corresponding representative features are \{2,3,5,6,7,9\} (feature names are “DevType”, “FP”, “PrgLang”, “DevPlatform”, “Time”, “OrgType”), the PRED (0.25) value achieve highest value 0.2627, and the MMRE value is 1.3938. Table 6.6 shows the results of different number of representative features on the Desharnais data set. When KCluster number is 2, the corresponding representative features are \{5,10\} (feature names are “Language”, “PointAdjust”), the PRED (0.25) value achieve highest value 0.5063, and the MMRE value is 0.3406.
### Table 6.6 Results of different number of representative features in Desharnais

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Representative Feature</th>
<th>KCluster</th>
<th>MMRE</th>
<th>PRED(0.25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desharnais</td>
<td>10</td>
<td>1</td>
<td>0.7028</td>
<td>0.3636</td>
</tr>
<tr>
<td></td>
<td>10 5</td>
<td>2</td>
<td>0.3406</td>
<td>0.5062</td>
</tr>
<tr>
<td></td>
<td>10 5 7</td>
<td>3</td>
<td>0.4561</td>
<td>0.4155</td>
</tr>
<tr>
<td></td>
<td>10 5 7 1</td>
<td>4</td>
<td>0.4818</td>
<td>0.3766</td>
</tr>
<tr>
<td></td>
<td>10 5 7 1 3</td>
<td>5</td>
<td>0.7657</td>
<td>0.2727</td>
</tr>
<tr>
<td></td>
<td>10 5 7 1 3 2</td>
<td>6</td>
<td>0.8294</td>
<td>0.2467</td>
</tr>
<tr>
<td></td>
<td>10 5 7 1 3 2 9</td>
<td>7</td>
<td>0.8171</td>
<td>0.2337</td>
</tr>
<tr>
<td></td>
<td>10 5 7 1 3 2 9 4</td>
<td>8</td>
<td>0.7651</td>
<td>0.2597</td>
</tr>
<tr>
<td></td>
<td>10 5 7 1 3 2 9 4 8</td>
<td>9</td>
<td>0.7938</td>
<td>0.2331</td>
</tr>
</tbody>
</table>

Table 6.6 Results of different number of representative features in Desharnais

### 6.6.2 Different Number of Nearest Neighbors

<table>
<thead>
<tr>
<th>Data Set</th>
<th>K Nearest Neighbors</th>
<th>MMRE</th>
<th>PRED (0.25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISBSG</td>
<td>1</td>
<td>1.2607</td>
<td>0.2613</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.2169</td>
<td>0.2462</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1.3937</td>
<td>0.2627</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1.9238</td>
<td>0.2644</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1.8723</td>
<td>0.2583</td>
</tr>
</tbody>
</table>

Table 6.7 Results of different nearest neighbors in ISBSG

<table>
<thead>
<tr>
<th>Data Set</th>
<th>K Nearest Neighbors</th>
<th>MMRE</th>
<th>PRED (0.25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desharnais</td>
<td>1</td>
<td>0.3862</td>
<td>0.4415</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.3669</td>
<td>0.3766</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.3406</td>
<td>0.5062</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.3662</td>
<td>0.4285</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.3803</td>
<td>0.4675</td>
</tr>
</tbody>
</table>

Table 6.8 Results of different nearest neighbors in Desharnais

In order to find out the impact of different nearest neighbors in the performance, the number of representative feature should be fixed. Therefore, the number of representative feature is set 6 (\{2,3,5,6,7,9\}) and 2 (\{2,5\}) in ISBSG data set and Desharnais data set, respectively. The experiment results are shown in Table 6.7 and 6.8. The different number of nearest neighbors has little impact on PRED (0.25) value but much impact on MMRE value in ISBSG data set. However, the number of nearest neighbors is rather important on Desharnais data set. When k value is 3, HFSFC method can achieve highest PRED (0.25) value and lowest MMRE value.
6.7 Comparison of Feature Selection Methods

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Method Type</th>
<th>Method Name</th>
<th>MMRE</th>
<th>PRED(0.25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISBSG</td>
<td>Hybrid Learning</td>
<td>HFSFC</td>
<td>1.3938</td>
<td>0.2627</td>
</tr>
<tr>
<td></td>
<td>Supervised Learning</td>
<td>INMIFS</td>
<td>1.4786</td>
<td>0.2333</td>
</tr>
<tr>
<td></td>
<td>Unsupervised Learning</td>
<td>FSFC</td>
<td>1.4660</td>
<td>0.2318</td>
</tr>
<tr>
<td>Desharnais</td>
<td>Hybrid Learning</td>
<td>HFSFC</td>
<td>0.3406</td>
<td>0.5063</td>
</tr>
<tr>
<td></td>
<td>Supervised Learning</td>
<td>INMIFS</td>
<td>0.3951</td>
<td>0.4893</td>
</tr>
<tr>
<td></td>
<td>Unsupervised Learning</td>
<td>FSFC</td>
<td>0.7425</td>
<td>0.3625</td>
</tr>
</tbody>
</table>

Table 6.9 Comparison of HFSFC, INMIFS and FSFC on two data set

From the experiment results in Table 6.9, it can be seen easily that HFSFC method outperforms INMIFS method and FSFC method.

In ISBSG data set, the number of representative features here is 6 and the number of nearest neighbors is 3. The MMRE value of HFSFC method is 5.74% and 4.92% lower than that of INMIFS method and FSFC methods while the PRED (0.25) value of HFSFC method is 12.60% and 13.33% higher than that of INMIFS method and FSFC methods.

In the Desharnais data set, the number of representative features is 2 and the number of nearest neighbors is 3. The MMRE value of HFSFC method is 13.79% and 54.13% lower than that of INMIFS method and FSFC methods while the PRED (0.25) value of HFSFC method is 3.75% and 39.67% higher than that of INMIFS method and FSFC methods.

Figure 6.4 Comparison of HFSFC method, INMIFS method and FSFC method in ISBSG

Figure 6.5 Comparison of HFSFC method, INMIFS method and FSFC method in Desharnais
### 6.8 Experiment of Feature Weight in Case Selection

<table>
<thead>
<tr>
<th>Data set</th>
<th>Feature set</th>
<th>K Nearest Neighbors</th>
<th>MMRE</th>
<th>PRED (0.25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desharnais</td>
<td>SU</td>
<td>1</td>
<td>0.3473</td>
<td>0.4750</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0.3459</td>
<td>0.5125</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>0.3094</td>
<td>0.5513</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>0.3590</td>
<td>0.4750</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>0.3787</td>
<td>0.4375</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>1</td>
<td>0.3862</td>
<td>0.4415</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0.3669</td>
<td>0.3766</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>0.3406</td>
<td>0.5062</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>0.3662</td>
<td>0.4285</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>0.3803</td>
<td>0.4675</td>
</tr>
<tr>
<td>ISBSG</td>
<td>SU</td>
<td>1</td>
<td>1.4770</td>
<td>0.2515</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1.4887</td>
<td>0.2424</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1.3761</td>
<td>0.2696</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>1.3804</td>
<td>0.2393</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>1.3788</td>
<td>0.2212</td>
</tr>
<tr>
<td></td>
<td>None</td>
<td>1</td>
<td>1.2607</td>
<td>0.2613</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>1.2169</td>
<td>0.2462</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>1.3937</td>
<td>0.2627</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>1.9238</td>
<td>0.2644</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>1.8723</td>
<td>0.2583</td>
</tr>
</tbody>
</table>

Table 6.10 Experiment results of feature weight on ISBSG and Desharnais data set

In this section’s experiment, the feature subset in ISBSG data set and Desharnais data set are \{2,3,5,6,7,9\} and \{5,10\}. The “SU” in the table 6.10 means that it employs “Symmetric Uncertainty” as feature weight while “None” means it does not employs any feature weight formula.

As shown in the Table 6.10, it is obvious that using feature weight can outperform non-using feature weight in the Desharnais data set. In terms of PRED (0.25) value, when K nearest neighbors is 1-4, the performance of using feature weight is better than that of non-using feature weight. When K nearest neighbors is 3, it obtains the highest PRED (0.25) value 0.5513, which is 8.9% higher than non-using feature weight. The MMRE value of using feature weight is lower no matter what K nearest neighbors is when comparing to non-using feature weight. When K is 3, using feature weight can outperform 9.16% over the non-using feature weight.

Though the feature weight can improve the performance in the Desharnais data set, it cannot bring obvious improvement in the ISBSG data set. In Desharnais data set, numeric data is much more than the nominal data, so the feature weight is easy to reflect the quantitative
relation between each independent feature and the estimated feature. But in the ISBSG data set, it has more nominal data and the quantitative relation is not as important as in the numeric data. Therefore, the performance of feature weight in ISBSG data set is improved not much.

Figure 6.6 Experiment results of feature weight in ISBSG data set.

Figure 6.7 Experiment results of feature weight in Desharnais data set.
Chapter 7. Conclusion and Feature Work

7.1 Conclusion

This thesis mainly focuses on feature selection and case selection in software cost estimation based on analogy.

First, it compares several popular sequential search feature selection methods and demonstrates that the INMIFS method can achieve better performance than other methods.

Second, it proposes a novel clustering feature selection method HFSFC. HFSFC method uses normalized mutual information as the feature similarity measurement and group independent features into several clusters, then it selects the representative features to form the optimal feature subset as the final result of feature selection method. The experiment result shows that HFSFC method outperforms the INMIFS method and FSFC method.

Third, it employs symmetric uncertainty in the feature weight to reflect the impact of different independent features in the global distance calculation, and it achieves rather good results in numeric data set.

The work in this thesis will be a supplement in the research of software cost estimation and help to improve the predictive accuracy.

7.2 Future Work

The symmetric uncertainty feature weight is useful for the numeric data but not the nominal data. So it is necessary to improve the feature weight formula to make it more applicative. Besides, the data sets used in this thesis is not so up-to-date, so it is necessary to update the data set in order to make it more close to the real situation. In addition, there are some new techniques for feature clustering such as MST (minimum spanning tree) which is rather computationally efficient. So maybe MST is a good choice for clustering feature selection method.
Acknowledgement

First of all, I would like to give my deepest gratitude to my supervisor Juliana Stoica and Qin Liu for their patience, great support and constructive suggestions. Their expertise in software engineering and positive characters in life always motivate me excel in my profession for continued excellence.
I would also thank all my dear classmates in the SDE SIG lab, especially Jiakai Xiao and Xiaoyuan Chu. You two give me many impressive thought in designing algorithms and models and help me solve some difficult problems in coding the estimating model.
Finally I have to thank for my parents who give me life and have made me grown healthily and happily. Every achievement I made is due to their support and love.

References

[1] CHAOS SUMMARY FOR 2010, the Standish Group, 2010


[32] Liu Q, Mintram R C. Preliminary data analysis methods in software estimation [J]. Software


Appendix One: Developer Manual

The program can be divided into five parts:
1. Prepare necessary data: Calculate matrix data such as entropy of each feature and mutual information of two features for feature selection.
2. Define Unsupervised feature selection: define algorithm for unsupervised feature selection and pick out the best feature subset for case selection
3. Define Feature weight: define feature weight using symmetric uncertainty
4. Define Evaluate metric: define evaluate metric for estimation model
5. Define case adaptation and evaluate performance

Source Code:
#Log#
#Add cross validation into unsupervised feature selection method#

#Load library#
library(infotheo);
library(graphics);

#Configuration list#
config<-list(  
  wd="E:/Data/SourceData",
  fileName=c("deshstd.txt","r8std.txt"),
  similarityMatrix=c("mi","su","nmi","mici"),
  featureWeight=c("none","su"),
  featureSelection=c("supervised","unsupervised"),
  evaluationApproach=c("kfold","leave_one_out"),
  kCluster=c(1,2,3,4,5,6,7,8,9),
  kfold=c(3,5,10),
  kNearestNeighbour=c(1,2,3,4,5)
)

#Settings of this experiment#
wd<-config[["wd"]]; #working directory
fileName<-config[["fileName"]][1]; #file name
vecColType<-vector(length=11); #feature column type: "1" for categorical data data and "0" for
numeric data

if(fileName==config["fileName"]){
    vecColType=c(0,0,1,0,1,0,0,0,0,0,0);  ##deshstd.txt
} else if(fileName==config["fileName"]){
    vecColType=c(1,1,0,1,1,1,0,1,1,0,0);  ##r8std.txt
}

k=config["kNearestNeighbour"][3]; #number of nearest neighbours in KNN
kCluster=config["kCluster"][2]; #number of clusters in hierarchical clustering
similarityMatrix=config["similarityMatrix"][3]; #the approach for similarity measurement
kFoldNbr=config["kfold"][3];

#Data used in this experiment
setwd(wd);
dData=as.matrix(read.table(fileName,header=TRUE));
colNumber=ncol(dData);
sData=dData[, colNumber]; #eliminate the "effort" column for unsupervised learning

#Main entrance of this program#
mainFunc=function(wd, fileName){
    ###############Unsupervised Feature Selection##################
    #Entropy of each feature#
    entp=getEntropy(sData);
    smltMatrix=0; #similarity matrix
dissmltMatrix=0; #dissimilarity matrix
    if("mi"==similarityMatrix){
        smltMatrix=getMiMatrix(sData); #mutual information matrix#
dissmltMatrix=getDissimilarityMatrix_MI(smltMatrix);
    } else if("su"==similarityMatrix){
        miMat=getMiMatrix(sData); #symmetric uncertainty matrix#
smltMatrix=getSymmetricUncertainty(miMat, entp);
dissmltMatrix=getDissimilarityMatrix_SU(smltMatrix);
    } else if("nmi"==similarityMatrix){
        miMat=getMiMatrix(sData); #normalized mutual information matrix#
smltMatrix=getNormalizedMiMat(miMat,entp);dissmltMatrix=getDissimilarityMatrix_NMI(smltMatrix);
    } else if("mici"==similarityMatrix){
        dissmltMatrix=getDissimilarityMatrix_MICI(sData);
    }

    #get triangle distance matrix
tDSM=getTriangleDSM(dissmltMatrix);
# Hierarchical clustering
hc = hclust(tDSM, "complete");

# plot(hc); plot(hc, hang = -1);

# cluster matrix: in each row i, clusterMatrix[i,j]==1 means that feature j is selected into one cluster in clustering step i.
clusterMatrix = getClusterFeatures(hc$merge, colNumber - 1);

# get representative feature in each cluster: the feature with smallest distance sum to all the other features in the cluster will be selected
vecRepresentativeFeatures = getRepresentativeFeature_MI(parseClusterResults, clusterMatrix, kCluster);

# get the needed features for evaluation
vecBestSubset = vecRepresentativeFeatures;

# Evaluate estimation model: leave-one-out
vecMMRE = 0;
vecPRED = 0;
vecMdMRE = 0;

# kFold = nrow(dData);
weight <- getFeatureWeights_None(vecBestSubset);

# Each case will act as testing set once and the other cases act as training set#
for (z in 1:kFoldNbr) {

## Separate input data into two parts: TrainingSet and TestingSet
rtnList <- separateSets(dData, kFoldNbr);
vecTestingIds = rtnList$var1;
vecTrainingIds = rtnList$var2;

### Evaluate model performance###

# Evaluate estimation model: leave-one-out
vecMMRE = 0;
vecPRED = 0;
vecMdMRE = 0;

# kFold = nrow(dData);
weight <- getFeatureWeights_None(vecBestSubset);

# Each case will act as testing set once and the other cases act as training set#
for (z in 1:kFoldNbr) {

## Separate input data into two parts: TrainingSet and TestingSet
rtnList <- separateSets(dData, kFoldNbr);
vecTestingIds = rtnList$var1;
vecTrainingIds = rtnList$var2;
testingData <- matrix(nrow=length(vecTestingIds), ncol=ncol(dData));
trainingData <- matrix(nrow=length(vecTrainingIds), ncol=ncol(dData));

for(i in 1:length(vecTestingIds)){
    testingData[i,] <- dData[vecTestingIds[i],];
}
for(i in 1:length(vecTrainingIds)){
    trainingData[i,] <- dData[vecTrainingIds[i],];
}

# evaluate testing set
evaluation <- EvalTesting(testingData, trainingData, weight, vecBestSubset);
# collect the experiment results
result <- vector(length=3);
result[1] <- MMREFunc(evaluation, nrow(testingData));
result[2] <- PREDFunc(evaluation, nrow(testingData), 0.25);
result[3] <- MdMREFunc(evaluation, nrow(testingData));
# print("Testing Set Result: ")
# print(result);

vecMMRE[z] <- result[1];
vecPRED[z] <- result[2];
vecMdMRE[z] <- result[3];

print(vecMMRE);
print("PREDs: ")
print(vecPRED);
print("MdMREs: ")
print(vecMdMRE);
print("Average in MMRE: ")
print(mean(vecMMRE))
print("Average in PRED: ")
print(mean(vecPRED))
print("Average in MdMRE: ")
print(mean(vecMdMRE))


########################################################################
### Part A: Get value of entropy, mutual information, dissimilarity ######
########################################################################
A.1 Calculate entropy value of each column
getEntropy <- function(dData){

entp<-vector(length=ncol(dData));
for(i in 1:ncol(dData)){
##discretize continuous data and calculate entropy
  entp[i]=entropy(discretize(dData[,i]));
}
#print("Entropy vector.");
#print(entp);
return(entp);
}

##A.2 Calculate mutual information matrix between two columns
getMiMatrix<-function(dData){
  ##Allocate a new matrix to store MI results
  miMat<-matrix(nrow=ncol(dData),ncol=ncol(dData))
  ##Get MI of every two cols (Independent-Independent & Independent-Response)
  for(i in 1:ncol(dData)){
    for(j in 1:ncol(dData)){
      ##discretize continuous data
      miMat[i,j]=mutinformation(discretize(dData[,i]),discretize(dData[,j]));
    }
  }
  #print("Mutual inforatination matrix.");
  #print(miMat)
  return(miMat)
}

##A.3 Calculate normalized mutual information matrix between two columns
getNormalizedMiMat<-function(miMat, entp){
  NMImat=matrix(nrow=entp, ncol=entp);
  for(i in 1:length(entp)){
    for(j in 1:length(entp)){
      NMImat[i,j]=miMat[i,j]/(min(entp[i],entp[j]));
    }
  }
  return(NMImat);
}

##A.4 Calculate symmetric uncertainty between two features
getSymmetricUncertainty<-function(miMat, entp){
  miWeight=matrix(nrow=entp, ncol=entp);
  for(i in 1:length(entp)){
    for(j in 1:length(entp)){
      miWeight[i,j]=(2*miMat[i,j])/(entp[i]+entp[j]);
    }
  }

```r
### A.5 Get dissimilarity matrix symmetric uncertainty
getDissimilarityMatrix_SU<-function(miSU){
  mat=1-miSU[,1:10,1:10];
  #print("Dissimilarity matrix(SU): ");
  #print(mat);
  return(mat);
}

### A.6 Get dissimilarity matrix of standard mutual information
getDissimilarityMatrix_MI<-function(miMat){
  mat=1-miMat[,1:10,1:10];
  #print("Dissimilarity matrix(MI): ");
  #print(mat);
  return(mat);
}

### A.7 Get dissimilarity matrix of normalized mutual information
getDissimilarityMatrix_NMI<-function(NMiMat){
  mat=1-NMiMat[,1:10,1:10];
  #print("Dissimilarity matrix(NMI): ");
  #print(mat);
  return(mat);
}

### A.8 Get dissimilarity matrix of maximal information compression index
getDissimilarityMatrix_MICI<-function(sData){
  colNbr=ncol(sData);
  MICIMat=matrix(nrow=colNbr,ncol=colNbr);
  varVector=vector(length=colNbr);
  for(i in 1:length(varVector)){
    varVector[i]=var(sData[,i]);
  }
  ccMat=matrix(nrow=colNbr, ncol=colNbr);
  for(i in 1:colNbr){
    for(j in 1:colNbr){
      temp1=cov(sData[,i],sData[,j]);
      temp2=sqrt(varVector[i]*varVector[j]);
    }
  }
}
ccMat[i,j]=temp1/temp2;
}

# print("Correlation Coefficient: ");
# print(ccMat);
for(i in 1:colNbr){
    for(j in 1:colNbr){
        temp1=varVector[i]+varVector[j];
        temp2=sqrt((varVector[i]+varVector[j])^2 - 4*varVector[i]*varVector[j]*(1-ccMat[i,j]^2));
        MICIMat[i,j]=(temp1-temp2)/2;
    }
}

# print("MICI Matrix: ");
# print(MICIMat);
return(MICIMat);

### A.9 Get dissimilarity matrix in triangle format using "as.dist" function

getTriangleDSM<-function(dsm){
    tDSM=dsm;
    colNum=ncol(dsm);
    for(i in 1:colNum){
        for(j in i:colNum){
            tDSM[i,j]=0;
        }
    }
    # print(tDSM);
    return(as.dist(tDSM));
}

# Part B: Unsupervised feature selection to get representative feature of each cluster #

# B1. Get representative feature from each cluster
getRepresentativeFeature_TopKDis<-function(clusterMatrix, dsm, k){
    clusterMatrixRow=nrow(clusterMatrix);
    clusterMatrixCol=ncol(clusterMatrix);
    vecIsFeatureSelected=c(rep(0,10));
    vecRepresentativeFeatures=vector(length=k);
    for(i in 1:k){

vecRepresentativeFeatures[i]=getRepresentativeFeature_Core_MinDisSum(clusterMatrix[clusterMatrixRow+1-i], dsm, vecIsFeatureSelected);
vecIsFeatureSelected[vecRepresentativeFeatures[i]]=1;

print(vecRepresentativeFeatures);
return(vecRepresentativeFeatures);

#B2.1. Get representative feature from each cluster (Core:Minimum distance sum)
getRepresentativeFeature_Core_MinDisSum<-function(clusterMatrixRow, dsm, vecIsFeatureSelected){
  representativeFeature=0;
  disMin=100;
  for(i in 1:length(clusterMatrixRow)){
    disSum=0;
    if(1==clusterMatrixRow[i]&&0==vecIsFeatureSelected[i]){ 
      for(j in 1:length(clusterMatrixRow)){
        if(1==clusterMatrixRow[j]&&j!=i){
          disSum=disSum+dsm[i,j];
        }
      }
      if(disSum<disMin){
        disMin=disSum;
        representativeFeature=i;
      }
    }
  }
  if(0==representativeFeature){
    for(i in 1:length(vecIsFeatureSelected)){
      if(0==vecIsFeatureSelected[i]){ 
        representativeFeature=i;
      }
    }
  }
  return(representativeFeature);
}

#B2.2. Get representative feature from each cluster (Core:Maximal distance sum)
getRepresentativeFeature_Core_MaxDisSum<-function(clusterMatrixRow, dsm, vecIsFeatureSelected){
  representativeFeature=0;
  disMax=0;
  for(i in 1:length(clusterMatrixRow)){
    disSum=0;
    if(1==clusterMatrixRow[i]&&&0==vecIsFeatureSelected[i]){ 
      for(j in 1:length(clusterMatrixRow)){
        if(1==clusterMatrixRow[j]&&&j!=i){
          disSum=disSum+dsm[i,j];
        }
      }
      if(disSum>disMax){
        disMax=disSum;
        representativeFeature=i;
      }
    }
  }
  return(representativeFeature);
}
disSum=0;
if(1==clusterMatrixRow[i]&&0==vecIsFeatureSelected[i]){
    for(j in 1:length(clusterMatrixRow)){
        if(1==clusterMatrixRow[j]&&j!=i){
            disSum=disSum+dsm[i,j];
        }
    }
    if(disSum>disMax){
        disMax=disSum;
        representativeFeature=i;
    }
}
}

if(0==representativeFeature){
    for(i in 1:length(vecIsFeatureSelected)){
        if(0==vecIsFeatureSelected[i]){
            representativeFeature=i;
        }
    }
}
return(representativeFeature);
}

#B2.3. Get representative feature from each cluster (Core: mutual information with target feature)
getRepresentativeFeature_MI<-function(parseClusterResult, clusterMatrix, kValue){
    representativeFeature=vector(length=kValue);
    miMat=getMiMatrix(dData);
    targetFeatureColNbr=ncol(dData);
    if(1==length(parseClusterResult)){
        maxMiValue=0;
        for(i in 1:ncol(clusterMatrix)){
            if(miMat[i,targetFeatureColNbr]>maxMiValue){
                maxMiValue=miMat[i,targetFeatureColNbr];
                representativeFeature[1]=i;
            }
        }
        return (representativeFeature);
    }
    for(i in 1:kValue){
        tempValue=parseClusterResult[i];
        if(tempValue<0){
            representativeFeature[i]=0-tempValue;
        }
    }
    return(representativeFeature);
}
} else {
    colNbr = ncol(clusterMatrix);
    maxMiValue = 0;
    for (j in 1:colNbr) {
        if (1 == clusterMatrix[tempValue, j]) {
            if (miMat[j, targetFeatureColNbr] > maxMiValue) {
                maxMiValue = miMat[j, targetFeatureColNbr];
                representativeFeature[i] = j;
            }
        }
    }
}

# print("Representative feature: ");
# print(representativeFeature);
return(representativeFeature);

# B3. Get hierachical clustering matrix: each row represents one iteration in the clustering
getClusterFeatures <- function(clusterResult, featureNumber) {
    clusterMatrix = matrix(0:0, nrow=(featureNumber - 1), ncol=featureNumber);
    iteration = featureNumber - 1;
    for (i in 1:iteration) {
        temp1 = clusterResult[i, 1];
        if (temp1 < 0) {
            clusterMatrix[i, abs(temp1)] = 1;
        } else {
            for (x in 1:featureNumber) {
                if (1 == clusterMatrix[temp1, x]) {
                    clusterMatrix[i, x] = 1;
                }
            }
        }
        temp2 = clusterResult[i, 2];
        if (temp2 < 0) {
            clusterMatrix[i, abs(temp2)] = 1;
        } else {
            for (y in 1:featureNumber) {
                if (1 == clusterMatrix[temp2, y]) {
                    clusterMatrix[i, y] = 1;
                }
            }
        }
    }
}
#B4. Parse all the clusters in each step

parseClusters<-function(mergeMat, kValue){
  result=c();
  mergeMatRowNbr=nrow(mergeMat);
  if(1==kValue){
    result=0;
  }else{
    for(i in 1:(kValue-1)){
      pos=mergeMatRowNbr+1-i;
      if(0!=length(result)){
        for(j in 1:length(result)){
          if(pos==result[j]){result=result[-j];break;}
        }
      }
      leftValue=mergeMat[mergeMatRowNbr+1-i,1];
      result[length(result)+1]=leftValue;
      rightValue=mergeMat[mergeMatRowNbr+1-i,2];
      result[length(result)+1]=rightValue;
    }
  }
  #print("Parse Cluster:");
  #print(result);
  return(result);
}

#C.1 none weight: all feature weight is "1"

getFeatureWeights_None<-function(vecS){
  weightVector<-vector(length=length(vecS));
  for(i in 1:10){weightVector[i]=1;}
  return(weightVector);
#C.2 Use symmetric uncertainty as feature weight

def getFeatureWeights_SU(vecS):
    miMat = getMiMatrix(dData);
    entp = getEntropy(dData);
    suMatrix = getSymmetricUncertainty(miMat, entp);
    weightVector = vector(length=length(vecS));
    targetFeature = ncol(dData);
    for i in 1:length(vecS):
        weightVector[i] = suMatrix[vecS[i], targetFeature];
    return(weightVector);

#D.1 Devide raw data into two parts: training set and testing set

def separateSets(dData, kFold):
    dataRange = 0:nrow(dData);
    vecTestingIds = 0:
    vecTestingIds = sample(dataRange, round(nrow(dData)/kFold), replace=FALSE);
    vecTrainingIds = 0:
    rowcount = 1;
    for i in 1:nrow(dData):
        if any(vecTestingIds == i):
            vecTrainingIds[rowcount] = i;
            rowcount = rowcount + 1;
    return(list(var1=vecTestingIds, var2=vecTrainingIds);

#D.2 CBR algorithm in case selection

def CBR(target, ds, w, vecS):
    tempData = cbind(ds[, ncol(ds)], rep(NA, nrow(ds)));
    # distance from all rows
    for i in 1:nrow(ds):
        total = 0.0
        # your CBR algorithm here
#distance from the ith row
for(j in 1:length(vecS)){
  if(vecColType[vecS[j]]==1){
    if(target[vecS[j]]!=ds[i,vecS[j]]){
      total=total+w[j]*1;
      #total=total+1;
    }
  }else{
    total = total + w[j]*(target[vecS[j]]-ds[i,vecS[j]])^2;
    #total = total + (target[vecS[j]]-ds[i,vecS[j]])^2;
  }
}
tempData[i,2] <- sqrt(total);
}
#print(target);
#print(tempData);

#The number of rows with minimum distances
minimum = which.min(tempData[,2])
nMin = length(minimum)
estimate = 0.0
#print(target);

for(i in 1:k){
  minimum = which.min(tempData[,2]);
  ##print(tempData[minimum[1],1]);
  estimate = estimate + tempData[minimum[1],1];
  #Set the distance to a much greater value
  tempData[minimum[1],2] = 100;
}
estimate = estimate/k;
#print(estimate);
return(estimate)

#D.3 Evaluation function
#ds is the whole data set
#weight is the weighting vector for each feature
#cbr is the CBR algorithm
#returns a n*2 matrix, where the first column is actual effort and the second the estimated
Eval<-function(ds,weight,vecS){
  #Keep the result
  evaluation = matrix(data=NA,nrow=nrow(ds),2)
  #Evaluate
  for(i in 1:nrow(ds)){

## D.4 Evaluate the method: Use TrainingSet to evaluate the TestingSet

```
EvalTesting<-function(TestingDataSet, TrainingDataSet, weight, vecS){
    # Keep the result
    evaluation = matrix(data=NA, nrow=nrow(TestingDataSet), 2)
    # Evaluate
    for(i in 1:nrow(TestingDataSet)){
        evaluation[i,1] = TestingDataSet[i, ncol(TestingDataSet)]
        evaluation[i,2] <- CBR( TestingDataSet[i,], TrainingDataSet, weight, vecS)
    }
    return(evaluation)
}
```

---

### E.1 MMRE function: Mean Magnitude Relative Error

```
MMREFunc<-function(evaluation,n){
    re = abs(evaluation[,2] - evaluation[,1])/evaluation[,1]
    reFinite = is.finite(re)
    mmre = sum(re[reFinite])/fre(reFinite)
    return(mmre);
}
```

```
fre<-function(x){
    count = 0;
    for(i in x){
        if(i==T){
            count = count +1
        }
    }
    return(count)
}
```

### E.2 MdMRE function: Median Magnitude Relative Error

```
MdMREFunc<-function(evaluation,n){
    MREVector<-vector(length=n);
    for(i in 1:n){
        ```
```
MREVector[i]=abs(evaluation[i,2]-evaluation[i,1])/evaluation[i,1];
}
return(median(MREVector));
}

#E.3 PRED function: Pred (l) is used as a complementary criterion to count the percentage of estimates that fall within less than l of the actual values.
PREDFunc<-function(evaluation,n,l){
  counter<0
  for(i in 1:n){
    temp<-abs(evaluation[i,2]-evaluation[i,1])/evaluation[i,1];
    if(temp<l){
      counter=counter+1;
    }
  }
  pred=counter/n;
  return(pred);
}

#Invoke the main function
mainFunc(wd,fileName);
Appendix Two: User Manual

1. Modify the configuration part in the source code. The parameters includes data source, similarity matrix, feature selection method, feature weight, k neighbors etc.
2. Copy the source code into the R console, and press ENTER key to run. Wait to obtain the results.
3. Source code running in the R console is shown in the sample picture as follow:

   ```
   > #Configuration list:
   > config<-list(
   +   wd="E:/Data/SourceData",
   +   fileName="Genstd.txt","GBstd.txt"),
   +   similarityMatrix="ml","eu","dml","micl"),
   +   featureWeight="(Cont", "Num"),
   +   featureSelection="(Supervised", "unsupervised"),
   +   evaluationApproach="(Kfold","leave_one_out")
   +   kCluster=1,2,3,4,5,6,7,8,9,
   +   kfoldMP=3,5,10,
   +   kNeighbors=0(1,2,3,4,5)
   >
   > #Settings of this experiment:
   > fileName=/config("fileName")[[1]]; #fileName
   > vecColTypes=vector(length=15); #feature column type: "1" for categorical data data and "0" for numeric data
   > if(fileName=/config("fileName")[[1]]){
   +   vecColTypes=c(0,0,1,0,0,0,0,0,0,0,0,0,0,0,0); #Genstd.txt
   +   else if(fileName=/config("fileName")[[2]]){
   +     vecColTypes=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1); #GBstd.txt
   +   }
   >}
   > k=config("KNeighbors")[[3]]; #number of nearest neighbour in KNN
   > kCluster=config("kCluster")[[2]]; #number of clusters in hierarchical clustering
   > similarityMatrix=config("SimilarityMatrix")[[3]]; #the approach for similarity measurement
   > kFold= config("kFold")[[3]]
   >
   > #Data used in this experiment
   > dataView();
   ```

4. Specified scenario:
   Sample input: 20130811_UnsupervisedFeatureSelection_CrossValidation.r

   Sample output:
   Line1: feature selected [2,3,5,8,9];
   Line2: MMRE [0.5814], PRED[0.4269]