Chapter 6

Feature Selection

Almost, all the databases contain some irrelevant/redundant features. That is why, removing the redundant features from the databases has been an interesting research problem for some decades. In the real-world situations, relevant features are unknown apriori. Therefore, many candidate features are introduced to better represent the domain. It has been found from the experiments that many of the features are either irrelevant or redundant to the target concept. An irrelevant feature does not affect the target concept in any way, and a redundant feature does not add anything new to the target concept [JKP94]. In many applications, the size of the dataset is so large that learning algorithms might not work as well before removing the unwanted features. Reducing the number of irrelevant or redundant features drastically reduces the running time of a learning algorithm and yields more general concept of a real-world classification problem [KS95], [KS96].

There are basically two categories of feature selection methods - *supervised*, where each instance is associated with a class label and *unsupervised*, where instances are not related with any class label. In case of supervised feature selection, the relevant features are selected to increase the accuracy of prediction of the class label for a given instance. Unsupervised feature selection is used as a preprocessing of other machine learning techniques to reduce the dimensionality of the domain space without much loss of information content.

Feature selection is defined by many authors in different ways. Some of them are
CHAPTER 6. FEATURE SELECTION

as follows.

Idealized: It finds the minimally sized feature subset, necessary and sufficient for a target concept [KR92].

Classical: It selects a subset \( M \) features from a set of \( N \) features, \( M < N \), such that value of a criterion function is optimized over all subsets of size \( M \).

Improving Prediction Accuracy: Here, aim of feature selection is to choose a subset of features for improving prediction accuracy or decreasing the size of the structure without significantly decreasing prediction accuracy of the classifier built using only selected features [KS96].

Approximating original class distribution: Here, goal of feature selection is to select a small subset such that the resulting class distribution given only the values for the selected features is as close as possible to the original class distribution given all feature values.

Generally, feature selection attempts to select the minimally sized subset of features according to two basic criteria: (i) the classification accuracy does not significantly decrease and (ii) the resulting class distribution, given only the values for the selected features, is as close as possible to the original class distribution given all the features. Some feature selection methods find the best feature subset in terms of some evaluating function among the possible \( 2^N \) subsets. Basically, there are four steps in a typical feature selection algorithm.

1. A generation procedure to generate the next candidate subset.

2. An evaluation function to evaluate the subset under examination.

3. A stopping criterion to decide when to stop.

4. A validation procedure to check if the selected subset is valid or not.

A good number of algorithms have been developed for feature selection over the years [Doa92]. Some of the prominent feature selection algorithms are Branch & Bound [NF77], Focus [AD91], Relief [KR92], LVF [LS96], etc. Next section presents a brief discussion on some of these algorithms.
CHAPTER 6. FEATURE SELECTION

This chapter also introduces a new supervised feature selection algorithm for binary classification based on frequent (or large) features of association rule mining technique [AMS+94, AMS+96].

6.1 Some Existing Feature Selection Algorithms

In this section, some of the popular and prominent feature selection algorithms are reproduced.

6.1.1 Branch and Bound

This is an exponential search algorithm and was proposed by Narendra and Fukonaga in 1977. It follows top-down approach with backtracking and based on the assumption that the criterion function is monotonous. Suppose, it is required to select 2 features out of four features \((f_1, f_2, f_3, f_4)\) by using \textit{Branch and Bound}. \textit{Branch and Bound} first constructs the search tree as given in Figure 6.1 on the following page, where root denotes the set of all features and leaves denotes the set of two features. Nodes of level \(k\) is constructed by removing \(k\) features from the root. Nodes in the \(k\)th level represents the subset of \(N - k\) features, where \(N\) is the total number of features. The algorithm starts with searching from the root and every time it reaches a leaf, it updates the \textit{bound} (current maximum) with the corresponding criterion value of the leaf. The advantage of the algorithm over exhaustive search is that it is not required to construct a branch of a node if the criterion value of the node is less than the current \textit{bound} because of monotonous property of the criterion function.
The algorithm is presented in Algorithm 6.1 on the next page. The algorithm needs inputs of required number of features \((M)\) and it attempts to find out the best subset. The algorithm uses two functions. The function \(\text{isbetter}(X,Y)\) checks if the set \(X\) is better than the set \(Y\) and the function \(\text{Card}(X)\) finds cardinality of the set \(X\). However, the algorithm suffers from the following drawbacks.

1. It does not perform well if the criterion function is of high computational complexity.
2. It does not guarantee to remove enough sub-trees.
3. Criterion value computation is slower, nearer to the root.
4. Removal of sub-trees is less, nearer to the root.

### 6.1.2 Relief

Relief uses heuristic technique to generate candidate feature subset and distance to evaluate a candidate subset. It is a feature weight-based algorithm and uses statistical method to choose the relevant features. It also uses the concept of \(\text{NearHit}\) and \(\text{NearMiss}\). \(\text{NearHit}\) of an instance is defined as the instance having minimum Euclidean distance among all instances of the same class as that of the instance. \(\text{NearMiss}\) of an instance is defined as the instance having minimum
CHAPTER 6. FEATURE SELECTION

Input: \( D, F, M \).
Output: \( S_f \).

B&B(\( D, F, M \))

1. If \( \text{Card}(F) \neq M \) then /*subset generation*/
2. \( j=0; \)
3. \( S_f = F; \)
4. For all features \( f \in F \) begin
5. \( S_j = F - f; \) /*remove one feature at a time */
6. If (\( S_j \) is legitimate) then
7. If isbetter(\( S_j, S_f \)) then
8. \( S_f = S_j; \)
    /*recursion*/
9. B&B(\( S_j, M \));
10. Endfor
11. \( j++; \)
12. Endif
13. Return \( S_f; \)

Algorithm 6.1: Branch & Bound
Euclidean distance among all instances of different class. The algorithm finds the weights of the features from a sample of instances and chooses the features with weight greater than a threshold. The algorithm uses one function \textit{diff()} to find difference of same features in two different records. The algorithm is given in \textit{Algorithm} 6.2 on the following page. The advantage of the algorithm is that it can work for noisy and correlated features. However, it suffers from following drawbacks.

1. It cannot work with redundant features and hence generates non-optimal features, if the database contains redundant features.

2. It works only with binary classes.

3. Another problem is the selection of optimum values of \textit{NoSample} and \textit{Threshold} is not clear.

\subsection*{6.1.3 Focus}

This is an inductive learning algorithm and it is based on the concept of \textit{Min-feature} bias. According to \textit{Min-feature} bias, if two functions are consistent with the training examples, the functions with minimum features will be preferred. The algorithm first identifies \( p \) features that are required to define a binary function over \( n \) boolean input features. Then, it applies some learning procedures that focus on those \( p \) features. In other words, the algorithm generates all possible feature subsets and uses consistency measure to evaluate the subsets. The algorithm has been found to work well with noise-free data. However, the main disadvantage is how to select correct inconsistency measure. The algorithm is given in \textit{Algorithm} 6.3 on the next page.
CHAPTER 6. FEATURE SELECTION

Input: $D$, $F$, NoSample, Threshold.
Output: $S_f$;

1. $S_f = \phi$;
2. Initialize all weights, $W_j$ to zero;
3. For $i = 1$ to NoSample
4. Randomly choose an instance $t$ in $D$;
5. Find its NearHit and NearMiss;
6. For $j = 1$ to $N$
7. $W_j = W_j - \text{diff}(f^i, \text{NearHit}(j))^2 + \text{diff}(f^i, \text{NearMiss}(j))^2$;
8. For $j = 1$ to $N$
9. If $W_j \geq \text{Threshold}$
10. Append feature $f^j$ to $S_f$;
11. Return $S_f$;

Algorithm 6.2: Relief

Input: $D$, $F$.
Output: $S_f$.

1. $S_f = F$;
2. For $i=0$ to $N$ /*$N$ is number of features*/
3. For each subset $X$ of size $i$
4. If no inconsistency in the training set $D$ then
5. $S_f = X$;
6. Return $S_f$;

Algorithm 6.3: Focus
6.1.4 **LVF**

LVF generates the candidate subsets randomly and uses consistency measure to evaluate a subset. It randomly searches the subset space and calculates an inconsistency count for the subset. To search the optimal subset, the algorithm uses *Las Vegas* algorithm [BB96]. The algorithm calculates the inconsistency count based on the intuition that most frequent class label among those instances matching this subset of features is the most probable class label. An inconsistency threshold is assumed and any subset with inconsistency measure greater than that value is rejected. The algorithm can find optimal subset even for datasets with noise and user does not have to wait too long because it outputs any subsets that is better than the previous best. The algorithm is given in *Algorithm 6.4* on the following page. The algorithm has used two functions: Card(X) and InConCal(X, Y). Card(X) finds cardinality of the set X and inConCal(X, Y) finds inconsistency between sets X and Y. This algorithm is efficient, as only the subset having the number of features smaller than that of the current best subset are checked for inconsistency. In addition to that, the algorithm is easy to implement and is guaranteed to find the optimal subset. However, the algorithm suffers from two major drawbacks.

1. Selection of optimum inconsistency threshold (ucon) is difficult.

2. Selection of number of samples (Maxtries) is also a difficult decision.

6.1.5 **Discussion**

In this section, some of the popular feature selection algorithms have been discussed. It has been observed that, different algorithms have used different concepts of relevant features to select the features. In addition to that, different algorithms used different assumptions. As for example, *Branch and Bound* has assumed that criterion function is monotonous; *Relief* is based on the concept of *NearHit* and *NearMiss* and so on. So, different algorithms give optimum results in different environments and with different data sets. In other words, no algorithm is suitable for all environments or can select relevant features in all types of data sets. As a matter of fact, it will be virtually impossible to design
Input: \( D,F,Maxtries,ucon \).
Output: \( S_f \), Set of relevant features.

1. \( S_f = F \);
2. For \( i=1 \) to \( Maxtries \);
3. Randomly choose a subset of features, \( X \);
4. If \( \text{Card}(X) \leq \text{Card}(S_f) \)
5. If \( \text{InConCal}(X,D) \leq ucon \)
6. \( S_f = X \);
7. Output \( X \);
8. Else
9. Append \( X \) to \( S_f \);
10. Output \( X \) as 'another solution';
11. Endfor
12. Return \( S_f \);

Algorithm 6.4: LVF

an algorithm, which will find most relevant features in all environments and for all types of data sets.

Researchers are trying to use different concepts to design algorithms to select relevant features. One such useful concept is the frequency count(support count), as can be found in association rule mining technique. To the best of our knowledge, there has not been much attempt to find relevant features based on frequent features/items of association rule mining technique [AMS+94, AMS+96]. J Moore used association rules to select features in a web page for web page clustering. V Jovanoski and N Lavrac used association rules in inductive concept learning i.e. to device a classifier. They also used association rules to measure the accuracy.
of other feature selection algorithms.

This chapter (next section) presents a supervised feature selection algorithm for binary classification based on frequent items/features of association rule mining technique [AMS+94, AMS+96]. In binary classification, instances of a database are associated with only one class label. The class label is either 1, which represents the instance belongs to the class, or 0, which represents the instance does not belong to the class. So, the instances in the database may belong to the class or may not belong to the class. The chapter also presents comparative results of the proposed algorithm and some existing algorithms.

6.2 The FFC Algorithm

The proposed algorithm, FFC (Feature selection using Frequency/support Count), is meant for relevant feature selection from binary classified data and is based on frequent items/features of association rule mining technique [AMS+94, AMS+96]. Here, frequency count refers to support count of association rule mining. Let us consider a binary classified database of instances, where each instance either belongs to a class or does not belong to the class. It is also assumed that each instance is of the form $<TID, f^1, f^2, ... , f^n, C_i>$, where $TID$ is the unique identification number of the instance, $f^i$ is the $i^{th}$ feature and $C_i$ is the class of the instance. $f^i$ can be either 1, if the feature has occurred in the instance, or 0, if the feature has not occurred in the instance. Similarly, $C_i$ can be either 1, if the instance belong to the class, or 0, if the instance does not belong to the class. As for example, one instance may be $(11, 1, 0, 1, ... , 1)$. The first number 11 is the instance number. Among the rest, 1 represents that corresponding feature has occurred and 0 represents that corresponding feature has not occurred. The last value represents the occurrence of the class, where 1 represents that the instance belongs to the class and 0 represents that the instance does not belong to the class.

It has been observed that in binary classified data, where instances are associated with only one class label, if a feature $f$ is relevant or has some influence on the occurrence of the class $C_i$ then there may be two possible cases: One is that the class $C_i$ occurs when $f$ occurs and the other is that class $C_i$ occurs when $f$ does
not occur in most of the instances. In association rule mining terminology, a feature $f$ will be relevant with respect to the class $C_i$, if either $fC_i$ or $f'\neg C_i$ is frequent, where $f'$ represents the non-occurrence of $f$. The algorithm explores these observations to find the relevant features.

The algorithm works as follows. The inputs to the algorithm are $D$, $\text{minsup}$, $\text{incr}$ and $\text{minf}$. The output is $S_f$. Initially, the algorithm assumes that all the features are relevant. So, $S_f$ contains all the features. Then it generates $L_1$ and $L'_1$ followed by generation of $C$. Each element of $C$ consists of two elements and is of the form $fC_i$, where $f$ is a feature and $C_i$ is the class such that either $f$ or $f'$ is frequent. Then the algorithm finds the support count for all the elements of $C$. The algorithm uses bitmaps [HLL03] of the features to find the support count because it reduces the execution time to a great extent. At the end, the relevant features are extracted. The relevant features are those which are included in at least one frequent element of $C$. Then $\text{minsup}$ is incremented by $\text{incr}$. The purpose of increasing the minimum support is to find the features with maximum possible support. This process is repeated as long as number of features in $S_f$ is greater than the minimum number of required features. If the number of selected features become less than the minimum number of required features, the immediate previous set of selected features is returned. Otherwise, $S_f$ is returned. The algorithm is presented in Algorithm 6.5 on the next page.

The algorithm uses the concept of frequent itemset. As a result, it may not be able to remove the redundant/correlated features in small database. The value of $\text{minsup}$, $\text{incr}$ and $\text{minf}$ plays a major role to determine the execution time and how quickly the algorithm will converge. The larger value of $\text{minsup}$ and $\text{incr}$, the more quickly the algorithm will converge. However, larger value of $\text{incr}$ may miss some relevant features. As far as $\text{minf}$ is concerned, the execution time decreases as the value of $\text{minf}$ increases for less number of iterations over the database. In the experiments, it was found that the value of $\text{minsup}$ between 0.01 and 0.05, and the value of $\text{incr}$ between 0.005 and 0.01 give good results. For dense database, $\text{minsup}$ can be set to a higher value. So, there should be some trade-off in choosing the values of the above mentioned parameters.
CHAPTER 6. FEATURE SELECTION

Input: \( D, \text{minsup}, \text{incr}, \text{minf} \).
Output: \( S_f \) (set of relevant features).

1. \( S_f = \) All features;
2. Scan the database and find the bitmaps of all the features and the class label.
3. Do while\(|S_f| > \text{minf}\)
4. \( C = \emptyset \);
5. \( L_1 = \{ f | \text{Sup}(f) \geq \text{minsup} \} \); /*Features whose occurrence is large*/
6. \( L'_1 = \{ f | \text{Sup}(f') \geq \text{minsup} \} \); /*Features whose non-occurrence is large*/
7. \( C = C \cup \{ xC | x \in L_1 \cup L'_1 \} \); // support count of \( C \)
8. For all \( c \in C \)
9. Find support count (\( c.\text{count} \)) of \( c \) using bitmaps;
10. \( F1 = S_f \);
11. \( S_f = \{ f | c \in C, \ c = fC_i/f'C_i, \ c.\text{count} \geq \text{minsup} \} \);
12. \( \text{minsup} = \text{minsup} + \text{incr} \);
13. Enddo
14. If \(|S_f| < \text{minf}\) then
15. \( S_f = F1 \);
16. Endif
17. Return \( S_f \);

Algorithm 6.5: FFC
CHAPTER 6.  FEATURE SELECTION

6.3 Experimental Results

Feature selection methods can be validated either by using artificial datasets or by real-world datasets. Artificial datasets are constructed with some known relevant features and some noise features. Feature selection methods are run over these datasets to check if they can find the known relevant features or not. In case of real-world datasets, relevant features are unknown. Accuracy of a feature selection method is determined with the help of a suitable classifier. However, selecting suitable classifier is difficult because different classifiers support different datasets. So, four artificial datasets were chosen to evaluate the performance of the algorithms.

6.3.1 Datasets Used

The datasets, which are described below, have combinations of relevant, correlated, irrelevant and redundant features. These datasets are available in UCI Machine Learning Repository (http://www.ics.uci.edu). Here, the attempt is to evaluate the strengths and weaknesses of the proposed method along with the other methods.

*CorrAL Dataset  [KAH96]*

The dataset consists of 32 instances, binary classes and six boolean features($A_0, A_1, B_0, B_1, I, C$), where $I$ is irrelevant and $C$ is the class level. Here, relevant features are $A_0, A_1, B_0$ and $B_1$.

*Modified Par3+3 Dataset*

The dataset contains 64 instances. It consists of binary classes and twelve boolean features. Among them, $A_1, A_2, A_3$ are relevant and $A_7, A_8, A_9$ are redundant.
**Monk1 and Monk3 Dataset [TBB+91]**

*Monk1* consists of five discrete features ($A_1, A_2, A_3, A_4$ and $A_5$) and the binary class, out of which $A_1, A_2$ and $A_5$ are relevant to the target concept.

*Monk3* consists of six discrete features ($A_1, A_2, A_3, A_4, A_5$ and $A_6$) and the binary class, out of which $A_2, A_4$ and $A_5$ are relevant to the target concept.

### 6.3.2 Experimental Setup

The proposed algorithm was implemented using a Intel PIV machine. The value of $\minsup$ and $\text{incr}$ were taken as 0.05 and 0.005 respectively for all the datasets. The value of $\minf$ was taken as 4 for *CorrAL* and 3 for other datasets.

### 6.3.3 Results

Average experimental results are presented in Table 6.1. All the algorithms took very less amount of time and there was a little variation in the execution times.

<table>
<thead>
<tr>
<th>Method (RA)</th>
<th><em>CorrAL</em> $(A_0,A_1,B_0,B_1)$</th>
<th><em>Monk3</em> $(A_2,A_4,A_5)$</th>
<th><em>Monk1</em> $(A_1,A_2,A_5)$</th>
<th>Modified Par 3+3 (${A_1,A_7}, {A_2,A_8}, {A_3,A_9}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relief</td>
<td>$A_0,B_0,B_1,C$</td>
<td>$A_2,A_5$ always &amp; one or both of $A_3,A_4$</td>
<td>$A_1,A_2,A_5$</td>
<td>$A_1,A_2,A_3,A_7,A_8,A_9$</td>
</tr>
<tr>
<td>B&amp;B</td>
<td>$A_0,A_1,B_0,I$</td>
<td>$A_1,A_3,A_4$</td>
<td>$A_1,A_2,A_4$</td>
<td>$A_1,A_2,A_3$</td>
</tr>
<tr>
<td>Focus</td>
<td>$A_0,A_1,B_0,B_1$</td>
<td>$A_1,A_2,A_5$</td>
<td>$A_3,A_4,A_5$</td>
<td>$A_1,A_2,A_3$</td>
</tr>
<tr>
<td>LFV</td>
<td>$A_0,A_1,B_0,B_1$</td>
<td>$A_2,A_4,A_5$</td>
<td>$A_1,A_2,A_5$</td>
<td>$A_2,A_3,A_7$</td>
</tr>
<tr>
<td>FFC</td>
<td>$A_0,A_1,B_0,B_1,I,C$</td>
<td>$A_1,A_2,A_4,A_5$</td>
<td>$A_1,A_2,A_5$</td>
<td>$A_1,A_2,A_3$</td>
</tr>
</tbody>
</table>

Table 6.1: Experimental Results of *Relief, B&B, Focus, LFV and FFC*
6.3.4 Observations

Followings are some observations from the experimental results.

1. The proposed algorithm could not remove the Correlated/Redundant features from CorrAL data which is evident in the result. However, it found all the relevant features in Monk1 and Monk3 datasets.

2. In case of Monk1, it could select all relevant features. The discrepancy in the results could be attributed to the inherent nature of frequent itemset, type and size of the database.

6.4 Discussion

This chapter has presented an algorithm for relevant feature selection in binary classified data using the concept of frequent itemset. Based on experimentation, it has been found that the proposed algorithm is equally good when compared with the other counterparts. One disadvantage of the algorithm is that it could not find all the relevant features in all the datasets. However, this cannot be considered as a major disadvantage, because no algorithm can find all the relevant features in all kinds of data. The main advantage of the algorithm is the simplicity and easy implementation compared to its counterparts. So, the algorithm can be very useful to find relevant features in binary classified data.