Feature Subset Selection Using Minimum Cost Spanning Trees

Mike Farah - 18548059

Supervisor: Sid Ray
## Contents

1 Pattern Recognition .............................................. 1
   1.1 Pattern Representation .................................... 1
   1.2 Feature Set Reduction ...................................... 1
       1.2.1 Feature Subset Selection .............................. 2
       1.2.2 Feature Extraction .................................. 2
       1.2.3 Feature Set Reduction Speed ......................... 3
   1.3 Classification ............................................. 3
       1.3.1 Bayesian Classifier .................................. 4
       1.3.2 Euclidean Distance Classifier ...................... 4
       1.3.3 Mahalanobis Classifier .............................. 4
       1.3.4 K-Nearest Neighbours Classifier ................... 5

2 Existing Feature Subset Selection Methods .................... 6
   2.1 Euclidean Distance ....................................... 6
   2.2 Mahalanobis Distance .................................... 6
   2.3 Feature Subset Searching ................................. 7
       2.3.1 Branch and Bound .................................. 7
       2.3.2 Sequential Forward Selection ...................... 8
       2.3.3 Sequential Backward Selection .................... 9
       2.3.4 Sequential Floating Selection ..................... 9
       2.3.5 Exhaustive Search .................................. 10

3 Graph Theoretic Approaches .................................. 10
   3.1 A Clustering Method Using Minimum Spanning Trees .... 10
   3.2 A Two-Sample Test Using Minimum Spanning Trees ....... 11
   3.3 A Graph Theoretic Approach in Pattern Recognition .... 11

4 Our Methodology .............................................. 12
   4.1 Proposed Criterion Function ............................. 12
   4.2 Calculating J(x) ......................................... 12
   4.3 Minimum Spanning Trees ................................ 13
       4.3.1 Delaunay triangulation ............................ 13
       4.3.2 Kruskal’s Algorithm ............................... 14
       4.3.3 The Fastest Algorithm for Constructing Minimum Spanning Trees 14
       4.3.4 Prim’s Algorithm .................................. 15

5 Summary ..................................................... 15
List of Figures

1. This diagram is from Devijver and Kittler (1982), page 307. It shows the original features \((y_1, y_2)\) in comparison to the features extracted \((u_1, u_2)\) by the Karhunen-Loeve transform.

2. This shows to classes, \(A\) on the left hand side and \(B\) on the right. A sample which belongs to \(B\) is highlighted as a large black point, on the edge of the \(B\) cluster.

3. This shows to classes, where one class makes a ring around another. The sample under consideration is highlighted as a large black point. Note that this is closer to the mean of the outer class, than to the class it belongs to.

4. This shows two feature subsets, subset \(A\) is graphed on the left, \(B\) on the right.

5. This shows two feature subsets, subset \(A\) is graphed on the left, \(B\) on the right.

6. This image is from Narendra and Fukunaga (1977), it represents the different subsets possible when selecting 2 features from 6 (discarding four of the features).

7. The graph on the left will produce the same value as the one on the right in the algorithm proposed by Friedman and Rafsky (1979), even though the one on the left is more desirable.
1 Pattern Recognition

Pattern recognition essentially enables the classification of objects into groups by learning from a small sample of objects. For example, to classify apples and strawberries, two classes result - apples and strawberries. A number of features on these fruits, such as size, weight, colour and texture exist. These features are then used to classify the fruits into one of the classes, apples or strawberries. Often, there are a large number of features and it is not feasible to use them all to classify the objects. This could be due to the cost of measuring and recording the features, or the computational cost, increasing the number of features used for classification, generally increases the classification process exponentially.

A subsequent problem of pattern recognition is, therefore, finding a reduced set of features to enable the classification of objects to be done efficiently and accurately. Continuing the aforementioned example, it would not be ideal to select colour as a key feature to identify apples and strawberries, since they can share the same colours (red strawberries and red apples). Better discriminants are size and weight since apples and strawberries differ in that respect.

Tou and Gonzalez (1974) believe that feature selection plays a central role in pattern recognition, and is one of the most difficult tasks. Our research consequently focuses on performing feature subset selection. We will be developing a criterion function which effectively rates feature subsets by building minimum cost spanning trees across the sample data-sets. To test out our new algorithm, we will be developing a piece of software called IFS (Interactive Feature Selection) which will enable users to run feature subset selection algorithms, on various sample data, to compare the accuracy and time of the algorithms.

According to Ray (2005c), pattern recognition can be divided into three stages: pattern representation, feature selection and classification, which are described below.

1.1 Pattern Representation

The pattern representation stage, Ray (2005c) explains, is concerned with measuring the features from the objects and recording the data so that a computer is able to process it. For example, if we were classifying apples and strawberries, then this stage may involve taking different weight and size measurements of the fruits. This stage essentially develops the data-sets that we will be using in our IFS program.

We will be using both real and synthetic data-sets for testing in IFS. For example, we are using the well known Iris dataset from Fisher (1936) (which represents 3 different Iris flowers, with 50 samples for each flower and 4 features recorded for each sample) and we are using the random generator by Wallace (1996) to produce our synthetic data sets.

1.2 Feature Set Reduction

A problem with pattern recognition, explains Friedman and Kandel (1999), is that often it is not possible to use optimal features. An example by Tou and Gonzalez (1974) is character recognition. This is the problem of interpreting human written characters. The best way may be to record the order of the strokes for each character and where each stroke begins and ends. For various reasons this may not be feasible, so instead we may record other features. A solution by Ray and Turner (1992) plots the characters onto a grid and counts the number of times pixels occur in each column and each row. This will not be as effective, but it can be easily measured and recorded. One can imagine that some rows and columns will be less useful than others, for example the borders, since
most characters would have the same value of zero. We may then want to discard the less useful features, since they do not provide extra information about the object. The importance of removing redundant features is further highlighted by a comment from Jain et al. (2000), stating that using a large number redundant features can make two different pattern appear similar.

The process of discarding features is called feature subset selection. Alternatively, we can combine features mathematically and produce a new reduced feature set. This is called feature extraction. According to Jain and Zongker (1997), there has been an increase of interest in this area due to increases in interest in areas, such as data mining.

The aim of feature set reduction, according to Jain and Dubes (1988) and Jain et al. (2000), is to decrease the cost of recognition and increase the classification accuracy (particularly when the number of samples is small in relation to the number of features and classes of the dataset). An important aspect of feature set reduction is speed. Even though this step is typically only ever run once, speed is still an issue, as it will be explained in section 1.2.3.

As explained, there are two methods of reducing the number of features, Jain et al. (2000) states that often both are used, feature extraction first to extract the key data from the feature set, and then feature subset selection to discard features which provide little information about the classes.

1.2.1 Feature Subset Selection

In feature subset selection, subset of the original features is selected to use for the classification process. Jain and Zongker (1997) explain feature subset selection as essentially trying to find a feature set X, for which \( X \subset Y \), where Y is the entire feature set. The subset X is chosen such that it optimises some criterion J. Feature subset selection can, therefore, be seen as two separate processes:

1. Obtaining a criterion function, “J”, which effectively rates feature subsets. This is the main issue in feature set reduction, according to Jain et al. (2000), and it is the focus of our research.

2. Feature subset searching. This step is necessary, because it is not feasible to run the proposed algorithm exhaustively on every feature subset possible (Jain and Zongker, 1997; Narendra and Fukunaga, 1977; Clausen, 1999). If, for example, we have 10 features to select from, then the number of different possible subsets are: \( 10 \choose 1 + 10 \choose 2 + 10 \choose 3 \ldots + 10 \choose 10 = 1023 \). This increases combinatorially. If we have 20 features, then there are over 1 million different subsets. This step involves searching possible subsets, with the goal of optimising the criterion function “J”.

In Section 2, feature subset selection is examined further.

1.2.2 Feature Extraction

Feature extraction, on the other hand, transforms the original set of features into a smaller set. Ray (2005b) explains that feature extraction takes an original set of features, \( Y = (y_1, \ldots, y_n) \) and transforms them into a smaller set \( X = (x_1, \ldots, x_m) \), where \( m < n \). Note that this is different from feature subset selection, in that the transformed features are not a subset of the original, but rather a product or function of them. An example of a commonly used feature extraction method is outlined below.
Karhunen-Loeve Transform

The key idea in the Karhunen-Loeve transform is, according to Devijver and Kittler (1982), to compress the information contained in the original set $Y$ to set $X$. The Karhunen-Loeve expansion works by decorrelating the features of $Y$, and then removing the features which do not contain significant information, which are those which have a smaller variance. Figure 1 from Devijver and Kittler (1982) shows the effect of the Karhunen-Loeve transformation. The original features $(y_1, y_2)$ are shown, and can be seen to be highly correlated. The transformed features $(u_1, u_2)$ are not correlated, and it is clear that $u_1$ provides significantly more discriminatory information than $u_2$, therefore $u_2$ can be discarded without much loss in information.

Ray (2005b) explains that the set $X$ can be calculated by finding the matrix $T$ and solving $\tilde{X} = T'^T \tilde{Y}$ where $\tilde{X}$ is the reduced feature vector $[x_1...x_m]^T$ and $\tilde{Y}$ is the original feature vector $[y_1...y_n]^T$. $T$ is obtained from the $m$ eigen vectors, which have the $m$ maximum eigen values of the resulting matrix of $S_w^{-1}.S_b$ (where $S_w$ is the within-class covariance matrix and $S_b$ is the between-class covariance matrix of the original feature set $Y$).

1.2.3 Feature Set Reduction Speed

An obvious question may be that as feature set reduction is typically done off line, the execution time of the process is not particularly important. As Jain and Zongker (1997) explain, this is valid to a point, however the process will need to be run at least once. The execution time becomes a problem when processing large data sets with hundreds of features. The execution time of the algorithm is then a crucial issue, as it may not be feasible to run some algorithms even once on large data sets. For this reason, our research will not only focus on the subset our algorithm chooses, but also time it takes choosing it. IFS will record both the subset chosen by and algorithm, and the total time the algorithm took to complete.

1.3 Classification

This is the final stage, where the reduced feature subset and, in general, knowledge learnt from the sample data is used to classify objects according to their classes. The Bayesian
classifier is introduced, which is the optimal classification method, and several other commonly used methods are discussed. We will need to implement classifiers into IFS, since they provide a natural means of testing the results of different feature set reduction approaches.

1.3.1 Bayesian Classifier

The Bayesian classifier is the optimal classification method. It uses the a’priori probabilities of the classes (which is the initial probabilities of the class, for example we may know that 70% of the fruits are apples and 30% are strawberries) and the probability of an input pattern occurring given a class to obtain the probability of a class occurring given an input pattern. This is expressed in Ray (2005a) as:

$$P(W_i|X) = \frac{P(X) \cdot P(X|W_i)}{P(X)}$$

Where $W_i$ is class $i$, and $X$ is the input pattern. Note that $P(X)$ does not need to be calculated, as it is the same value for all of the classes. The equation can then be calculated as:

$$P(W_i|X) = P(X) \cdot P(X|W_i)$$

Unfortunately, according to Ray and Turner (1992), this method is at best difficult to compute, if not possible at all. To solve this, other classification methods have been introduced to attempt to estimate this classification process. There are many classifiers available, we are going to be implementing the following three, due to being closely related to our research and/or commonly used in pattern recognition.

1.3.2 Euclidean Distance Classifier

For an input pattern $x$, Friedman and Kandel (1999) explains, this classifier will classify $x$ according the class mean it is closest to. The mean of each class is found by taking its average feature vector, and the the euclidean distance from $x$ to each class mean is calculated as:

$$d_i^2 = (x - \mu_i)'(x - \mu_i)$$

Where $\mu_i$ is the mean feature vector of class $i$ and $x$ is the feature vector of input pattern $x$. The input pattern $x$ is then assigned to the class $i$ if $d_i < d_j$ for all classes $j = 1...n$. Since we are not interested in the actual values of $d_x$, just whether the distance from one class is smaller than another, we can use $d_i^2$ and obtain the same result, without the expensive square root operation.

This is a quite a simple method, however it does have its drawbacks. For example, consider figure 2, the case of two classes, where one class has a much larger standard deviation than the other. This classifier will assign the input pattern to the wrong class, even though the input pattern is closer to class mean of $A$, it is clearly in the cluster of class $B$.

1.3.3 Mahalanobis Classifier

This is similar to the Euclidean classifier, however instead of using the Euclidean distance, we use the Mahalanobis distance which takes into account the statistical properties of the
Figure 2: This shows two classes, A on the left hand side and B on the right. A sample which belongs to B is highlighted as a large black point, on the edge of the B cluster.

Figure 3: This shows two classes, where one class makes a ring around another. The sample under consideration is highlighted as a large black point. Note that this is closer to the mean of the outer class, than to the class it belongs to.

classes. Ray (2005d) defines it as:

$$\Delta_i^2 = (x - \mu_i)'\Sigma_i^{-1}(x - \mu_i)$$

Where $\mu_i$ is the mean feature vector of class $i$, $x$ is the feature vector of input pattern $x$, and $\Sigma_i$ is the variance-covariance matrix of class $i$ for the given feature vector. The input pattern $x$ is then assigned to the class $i$ if $\Delta_i^2 < \Delta_j^2$ for all classes $j = 1...n$.

This method is expected to perform better than Euclidean since it takes into account the statistical properties of the classes. Unlike the Euclidean distance, this classifier will correctly assign the input pattern to class A in figure 2. However, there is another problem. This method will generally work quite well, as long as the data is from a Gaussian distribution. Consider figure 3, where there is class which makes a ring around another class. The variance-covariance matrix will not be effective in this case, because the point is clearly within the standard deviation of the outer-class. This classifier, and the Euclidean distance classifier will not be able to efficiently classify input patterns with this graph, even though both classes are clearly separated. This brings us to the next classifier.

1.3.4 K-Nearest Neighbours Classifier

The K-Nearest neighbors classifier is quite different from the above two. Duda and Hart (1973) explain that this algorithm works by finding the $k$ nearest neighbours of the input pattern (the neighbours consist of the sample data, whose classes we know) and assigning the input pattern to the class which occurs the most in those neighbours. For example,
if the classes of the \( k = 3 \) nearest neighbours are (1,2,2) then the input pattern will be assigned class 2, since this occurs the most.

This method is based on the assumption that patterns of the same class will be close together. There is no specific \( k \) to chose to optimise the classifier, the optimal value will vary with the data-sets. However, as a general rule, increasing \( k \) too much will decrease the accuracy, since the classifier will begin relying more on the a’priori probabilities of the classes rather than the neighbours of the input pattern. In our experimentation we found \( k = 3 \) to be a good value. Unlike the previous two, this method will be able to correctly classify input patterns, given a distribution like figure 3.

2 Existing Feature Subset Selection Methods

Many subset selection methods exist, however we will be focusing on the following since they are closely related to the proposed method, or are commonly used in pattern recognition. Generally the aim is to find a feature subset which keeps samples of the same class close together, and samples of different classes far apart. Two well known criterion functions are Euclidean distance and Mahalanobis distance, which are discussed below. In Section 3, two addition graph-theoretic and minimum-spanning tree methods are discussed due to being closely related to our research. Other approaches for feature selection include entropy minimisation, probabilistic methods and genetic algorithms.

2.1 Euclidean Distance

In this method the distance is measured from each class mean to the each other class mean. The aim is to maximise this distance overall, to ensure different classes are kept separate from each other. In our implementation we maximise the minimum distance between the classes. Alternatively we could maximise the average or median distance between the classes, but this holds a risk of accepting a subset which has some classes very close together, undetected because some are very far apart. \( J(x) \) can be easily calculated as:

\[
J(x) = \min \{ (\mu_i - \mu_j)' \Sigma_{ij}^{-1} (\mu_i - \mu_j) \}
\]

For all classes \( i, j \) where \( i \neq j \) and \( \mu_k \) is the mean feature vector of class \( k \).

Similarly to the Euclidean classifier, this method does not take into account the statistical properties of the classes, for example in figure 4 this classifier would choose feature subset \( B \), since the distance between the two classes is greater than subset \( A \), even though subset \( A \) separates the classes better.

2.2 Mahalanobis Distance

This method is similar to the euclidean distance, but we use the Mahalanobis distance instead. Ray (2005b) explains that \( J(x) \) is calculated as:

\[
J(x) = \min \{ (\mu_i - \mu_j)' \Sigma_{ij}^{-1} (\mu_i - \mu_j) \}
\]

\[
\Sigma_{ij} = \pi_i \Sigma_i + \pi_j \Sigma_j
\]

For all classes \( i, j \) where \( i \neq j \), \( \mu_k \) is the mean feature vector of class \( k \), \( \pi_k \) is the a’priori probability of class \( k \) and \( \Sigma_k \) is the variance-covariance matrix of class \( k \) for the given feature vector.
Figure 4: This shows two feature subsets, subset A is graphed on the left, B on the right.

Figure 5: This shows two feature subsets, subset A is graphed on the left, B on the right.

This is significantly more difficult to compute than the euclidean distance, however it should also be significantly better, since it takes into account the statistical properties of the classes. This method will choose feature subset A over B in figure 4. However, this method has similar drawbacks to the Mahalanobis classifier, consider figure 5. This will prefer feature subset B over A, even though subset A is better, since in A the distance between the class means is close to zero, which means $\Delta^2$ is close to zero.

2.3 Feature Subset Searching

As explained earlier, it is not feasible to run the aforementioned criterion functions on ever possible feature subset. Even though Cover and Campenhout (1977) explain that no non-exhaustive search can be guaranteed to select an optimal subset, we must search the possible subsets, with the possibility of discarding good, or optimal subsets, to ensure that the algorithm completes in a time efficient manner. We will be looking at the following feature subset searching methods:

2.3.1 Branch and Bound

The branch and bound algorithm is explained by Narendra and Fukunaga (1977) as an efficient feature subset search method which finds the optimal subset. Chen (2003); Narendra and Fukunaga (1977); Clausen (1999) explain that the efficiency of this algorithm is due to the algorithm discarding many subsets which are guaranteed to be sub-optimal, without having to actually run the criterion function on the sets. The algorithm assumes monotonicity. For example, if we have two feature subsets $A = \{1, 2, 3\}$ and $B=\{2, 3\}$ where $B \subset A$, then $J(A) \geq J(B)$, that is $A$ is no worse than $B$, with respect to the criterion
function being used.

The algorithm is explained by Narendra and Fukunaga (1977) as follows, from figure 6, the algorithm takes the right most branch, and descends down the single path until the bottom level (z4). At this point the “bound” is set to J(1,2) (since we have discarded \{3,4,5,6\}) and the subset \{1,2\} is saved as the best subset seen so far. The algorithm then backtracks up the tree to the top (z0) and chooses the next branch, left of the one just processed, which is ‘2’. J(\{1,3,4,5,6\}) is calculated, if it is smaller than the “bound” then, by monotonicity, we can discard that node and all the subsets below it, since they are guaranteed to be worse. We then backtrack up the tree and choose the next branch, which will be ‘1’.

On the other hand, if J(\{1,3,4,5,6\}) (node ‘2’) it is greater than the bound, we can descend down the tree. If we reach the bottom, we update the “bound”, save the subset as the best seen so far, and backtrack up the tree. The algorithm is finished when we have discarded and/or evaluated all the possible subsets, which is when we backtrack to the top from the left most branch.

Monotonicity is not a trivial issue, and it is clear that it plays an important role in this algorithm. Although the monotonicity criterion should hold for most data sets, it would be wrong to assume it holds for them all. Jain and Zongker (1997) point out that from the ‘curse of dimensionality’, in small samples size situations the monotonicity may not hold.

However, in the experiments by Hamamoto et al. (1990), the method is shown to work well even when the monotonicity criterion does not hold. The other problem with this method is that even though it is substantially more efficient than exhaustive search, Jain and Zongker (1997) point out that its worst case complexity is exponential, which becomes an issue when dealing with large feature sets.

Branch and Bound is a commonly used and well known method to solve combinatorial problems, such as feature subset selection, and therefore we will be implementing it in IFS.

2.3.2 Sequential Forward Selection

Forward selection works by selecting the best single feature and then progressively adding features which enhance a particular criterion the most. The algorithm is explained by Pudil et al. (1994) as:

1. We begin with the current subset \textit{Current} = \emptyset and \textit{Available} = \{x_1, x_2, ... x_m\}, where \(x_i\) is feature \(i\) and \(m\) is the total number of features.
2. Find \(x_i\) where \(J(\textit{Current} \cup x_i)\) is the largest for all \(x_j\) in \textit{Available}
3. \textit{Current} = \textit{Current} \cup x_i
4. \textit{Available} = \textit{Available} – \(x_i\)
5. If the subset size is smaller than our requirement, Goto step 2, else terminate the algorithm

While this method is fast, it discards many subsets at an early stage. Thus, it is unlikely to provide an optimal subset. As Guyon and Elisseeff (2003) state, a feature might seem useless by itself but can be quite useful with another particular feature.
2.3.3 Sequential Backward Selection

This algorithm effectively works in reverse of Forward Selection. We start with the complete set of features, and progressively discard the least useful feature until we have a subset of the required size.

This is more difficult to implement computationally, and also takes longer to run than forward selection. For these reasons it is not a commonly used method, and we will not be implementing it in IFS.

2.3.4 Sequential Floating Selection

This method essentially puts together forward selection and backward selection into the same algorithm. The problem with both methods, explains Pudil et al. (1994), is that once a decision has been made to add (or remove) a feature, there is no chance to undo the step later down the track. The sequential floating selection solves this by running a number of backward steps after each forward step, as long as each backward step results in a better subset. The algorithm given by Pudil et al. (1994) essentially works as follows:

Let $X_k = \{x_1, x_2, \ldots, x_k\}$, the current subset of size $k$. We begin with $X_0 = \emptyset$.

1. (a) Find $x$ where $\max(J(X_k \cup x))$
   (b) $X_{k+1} = X_k \cup x$
   (c) $k = k + 1$

   Apply one forward step, that is select the feature which when added to the current subset, the resulting subset enhances the criterion the most.

2. (a) Find $x$ where $\max(J(X_k - x))$
(b) If \( J(X_k - x) > J(X_{k-1}) \) then
   i. \( X_{k-1} = X_k - x \)
   ii. \( k = k - 1 \)
   iii. Goto step 2
(c) else Goto step 1

Apply one backward step, that is select the feature which when removed from the current subset, the resulting subset enhances the criterion the most. If that subset of size \( k - 1 \) performs better than the previous subset of size \( k - 1 \), remove the feature from the subset and re-apply step 2. Otherwise, goto step 1.

The algorithm terminates once we have a subset of the required size. Although this algorithm is not guaranteed to produce an optimal subset, unlike Branch and Bound this does not assume monotonicity. Pudil et al. (1994) states that this algorithm produces results very close to the Branch and Bound algorithm, but with less computational effort.

It will be useful to compare the results of this method and Branch and Bound using our proposed criterion function, mainly due to the monotonicity assumption in Branch and Bound. Therefore, this algorithm will also be implemented in IFS.

2.3.5 Exhaustive Search

Despite its efficiency limitations, will will be implementing the exhaustive search, which can be used on small-data sets. A benefit of exhaustive search is that it allows us to find out which subset the criterion function deems to be optimal, which can be a useful thing to compare. It can also be used to compare the results of different feature subset searching methods.

3 Graph Theoretic Approaches

The following three subsections describe three different methods which are closely related to our proposed criterion function. The first is a cluster validity measure, which uses minimum spanning trees. The next is a method which also uses minimum spanning trees and can be easily applied in feature subset selection and finally a graph-theoretic approach to feature subset selection is discussed.

3.1 A Clustering Method Using Minimum Spanning Trees

An algorithm for clustering by Smith and Jain (1984) plots random points ‘y’ within the bounds of the sample data ‘x’, then proceeds to build a minimum spanning tree across this data. The algorithm then counts the number of edges that joined a random point to a sample point, the smaller the number, the better. The idea behind it is, according to Jain and Dubes (1988), well clustered data should have a high number of x-x and y-y joins compared to random data. Although this can be used to measure how clustered the data is, it cannot be used to determine whether the clusters are from the same class or not, which is critical to selecting feature subsets.
3.2 A Two-Sample Test Using Minimum Spanning Trees

A method proposed by Friedman and Rafsky (1979) uses minimum spanning trees to evaluate whether two sets of n-dimensional data are from the same distribution. A minimum spanning tree is built across the data points, and edges which connect data from one distribution to the other are removed. If many edges are removed, then the data from the distributions are mixed up together, and so they must come from the same distribution.

This approach can easily be applied in feature subset selection. Instead of attempting to determine whether the sets of data come from different distributions, we try to find a feature subset which best shows that the sets of data come from different classes. Given a feature subset to evaluate, a minimum spanning tree is built across the sample data. The edges leading from one class to another are removed, and tallied. The more edges that are removed, the worse the feature subset is. This approach will be able to handle the distributions shown in figures 4 and 5.

However, does not take into account the weight of the edges, so that if one feature subset has two clusters connected by an edge length 2\(^x\) and another subset has 2 clusters connected by an edge length of \(x\), both will yield the same value when the subset with the edge of \(2^x\) should be selected. In addition, the algorithm does not take into account the distances between samples of the same class, which should be minimized. Dense clusters of the same class are preferred to sparse clusters. Figure 7 shows an example of this situation. Although it can be argued that either subset can be selected, and a classifier should provide excellent results in both cases, it is important to remember that the data being used is sample data, and the live data could have a larger spread, and noisier input patterns.

3.3 A Graph Theoretic Approach in Pattern Recognition

A new method of feature subset selection was proposed by Don and Kothari (2003). They introduced the concept that an n-dimensional classification problem can be visualised in \((n+1)\) dimensions with the class label as the extra dimension. An example provided was a 2D graph of a 2 feature problem, made into a 3D graph, with the class label as the third dimension. They explain that the smoothness of the class label surface can be quantified, and therefore the classifiability can be measured (the smoother the surface the better). This method is good for rating subsets when samples from different classes are close together and it will be able the distributions shown in figures 4 and 5.
However, similar to the method proposed by Friedman and Rafsky (1979), if two of subsets exist where clusters of samples from different classes are not touching, the class label surface will be smooth for both subsets. This is not desirable if one of the subsets keeps the classes separated further away than the other.

4 Our Methodology

4.1 Proposed Criterion Function

As explained previously, our research focuses on the criterion function of feature subset selection. Our proposed criterion function will rate a feature subset $x$ by building a minimum spanning tree, and calculating $J(x)$ from the edges of the graph. $J(x)$ will be calculated in such a way that graphs which keep samples from different classes separate and samples from the same class close, will be chosen.

We would like to stress that only recently we became aware that the method by Friedman and Rafsky (1979) (described in Section 3.2) is quite similar to ours and can be easily applied in feature subset selection. We have independently came up with our solution and it has some key differences. Primarily, the weights of the edges in the minimum spanning tree are used in addition to the number of edges which connect samples of different classes. This means that our criterion function will take into the account how close samples of the same class are and how far apart they are from samples of different classes. Since the bulk of the processing for both methods is calculating the distances between the nodes and building the minimum spanning tree, our method should not take much longer to complete to obtain better results than the method of Friedman and Rafsky (1979).

4.2 Calculating $J(x)$

The proposed function works as follows:

1. The euclidean distance between all the points is calculated, with respect to the feature subset $x$.

2. A minimum spanning tree is built across the data points, Section 4.3 explains how this is achieved.

3. From the tree, we obtain a number of variables which we will use to rate the subset:

   - $withinEdges$ The set of edges which connect samples of the same class.
   - $withinNum$ The cardinality of the $withinEdges$ set.
   - $betweenEdges$ The set of edges which connect samples of different classes.
   - $betweenNum$ The cardinality of the $betweenEdges$ set.

Note:

- $withinNum + betweenNum$ equals the total number of edges in the minimum spanning tree
- $withinEdges \cup betweenEdges$ equals the set of all the edges in the minimum spanning tree
- $withinEdges \cap betweenEdges = \emptyset$. 

12
4. A function $Avg(e)$ is a function on the set of edges $e$, this returns the median weight of the edges in the set $x$.

5. From these variables and $Avg(e)$, we obtain $J(x)$ which rates the effectiveness of the feature subset.

Turi and Ray (2000) states that a common goal in clustering is to find “compact clusters which are well separated”. They used a cluster validity measure of $validity = \frac{intra}{inter}$, where $intra$ is the intra-cluster distance, which we want to minimise, and $inter$ is the inter-cluster distance, which we want to maximise. The objective is, therefore, to minimize the validity measure.

Similarly in feature subset selection, we want to find a feature subset which has compact clusters of samples of the same class, which are well separated from clusters of samples from different classes. Keeping this in mind, this essentially means we want to have a small number of long $betweenEdges$, and a large number of short $betweenEdges$. Therefore, the aim is to maximise $withinNum$ and $Avg(betweenEdges)$ while minimising $betweenNum$ and $Avg(withinEdges)$. Currently, this is being done in the following equation:

$$J(x) = (1 - \frac{betweenNum}{betweenNum + withinNum}) + \frac{Avg(betweenEdges) - Avg(sameEdges)}{Avg(betweenEdges) + Avg(sameEdges)}$$

They key problem with this expression is that while $\frac{betweenNum}{betweenNum + withinNum}$ is bounded from 0 to 1, $\frac{Avg(betweenEdges) - Avg(sameEdges)}{Avg(betweenEdges) + Avg(sameEdges)}$ is unbounded. This is not only bad computationally, but it means the first half becomes more and more insignificant as the second half increases in size beyond 1.

4.3 Minimum Spanning Trees

Minimum spanning trees, explains Friedman and Rafsky (1979), are well known for providing superior descriptions of sets of points in pattern recognition.

A minimum-spanning tree is a sub-graph of a weighted, connected and undirected graph. It is acyclic, connects all the nodes in the graph, and the sum of all of the weight of all of its edges is minimum. That is, there is no other spanning tree, or sub-graph which connects all the nodes and has a smaller sum. According to Wikipedia (2005b), if the weights of all the edge’s are unique, then the minimum spanning tree is unique. The nodes in the tree will represent the samples, and the axis of the n-dimensional graph represent the n features.

The Euclidean minimum spanning tree is a minimum spanning tree where the weight of an edge connecting two nodes is the Euclidean distance between the nodes. The distance between sample pairs will therefore be the euclidean distance between their feature vectors. Coetzee (2005) explains that the a minimum spanning tree can be simply constructed by firstly building a complete graph such that all every node is connected to every other node (that is, calculate the distances of each pair of nodes), which takes $O(N^2)$ time and then building a minimum spanning tree from this graph using one of the techniques discussed below.

4.3.1 Delaunay triangulation

Coetzee (2005) proved that the minimum spanning tree of a graph is a subset of the Delaunay Triangulation on the graph. Applying Delaunay triangulation to a complete
graph will significantly reduce the number of edges in the graph, this is useful since many algorithms which build minimum spanning trees have complexities which depend on the number of edges in the graph.

However, further research into this method showed that this cannot be used in our solution. The problem is that Delaunay triangulation will not exist for a two dimensional graph if there are 3 points on the same line or four on the same circle. This cannot be guaranteed not to happen with our data-sets, and in fact, some of the sample data we are using have nodes not only on the same line, but on the same point (when they have identical feature subset vectors). For an n-dimensional graph the Delaunay triangulation will not exist if there are n+1 points on the same hyper-plane or n+2 points on the same hyper-sphere.

This presents the problem of either checking whether Delaunay triangulation can be performed on the data before running the algorithm, or restrict our method to not being able to handle data for which Delaunay Triangulation cannot be performed. For this reason, we have decided not to use this method, since it may end up complicating and increasing the time required to build a minimum spanning tree, or having some data-sets not work with our solution.

4.3.2 Kruskal’s Algorithm

This algorithm is explained by Morris (1998) as follows:

1. We start with a forest of trees, where each vertex is a minimum spanning tree, M is the set of all the edges in the minimum spanning tree and \( i = 0 \)

2. while \( (i < M) \)
   (a) Find the smallest edge that joins two trees and does not create a cycle (since a minimum spanning tree is acyclic)
   (b) Join the two trees that the smallest edge connects, into one tree
   (c) Increment \( i \)

According to Wikipedia (2005a), this algorithm can be done in \( O(M \log N) \) time (\( N \) is the number of nodes in the graph, which is equal to the number of samples), which makes it a good algorithm to use to find a minimum spanning tree on a sparse graph. A sparse graph is a graph that does not have many edges. However, in this case the opposite is true, we have a complete graph, in which every node in the graph is connected to every other node in the graph. This means that the number of edges in the graph is \( O(N^2) \), which makes the complexity of this algorithm \( O(N^2 \log N) \).

4.3.3 The Fastest Algorithm for Constructing Minimum Spanning Trees

Another algorithm for constructing minimum spanning trees was introduced by Chazelle (2000). Wikipedia (2005b) states that this is the fastest algorithm to date. The algorithm essentially uses a divide-and-conquer technique, recursively computing minimum spanning trees of subgraphs, and then merging them until will have the complete minimum spanning tree of the graph.

The complexity for this algorithm is \( O(M f(M,N)) \), where \( M \) is the number of edges, \( N \) is the number of nodes and \( f \), explains Chazelle (2000), is the “classical functional inverse of Ackermann’s function”. Wikipedia (2005b) states that for all practical purposes, \( f(M, N) \)
can be considered to be a constant less than or equal to 4. Therefore, for all practical purposes, Chazelle’s algorithm takes $O(M)$ time. Keeping in mind that we are building a minimum spanning tree on a complete graph, in this particular case, the algorithm is close to $O(N^2)$. This is a good improvement over Kruskal’s algorithm.

### 4.3.4 Prim’s Algorithm

According to Jain and Dubes (1988), this algorithm is well known to suit dense, or complete graphs, which is our situation. The algorithm works as follows: Let $v_i$ be vertex $i$, $N$ is the total number of nodes, $e_{ij}$ is the edge from $v_i$ to $v_j$ $T$ is the set of vertices in the minimum spanning tree $G$ is the set of $(v_i, e_{ij})$ where $v_i$ is a vertex not in the minimum spanning tree, and $e_{ij}$ is the closest edge to the spanning tree we have found so far.

1. Set $T = \{v_0\}$, $G = \{(v_1, e_{10})..(v_N, e_{N0})\}$
2. While ($G \neq \emptyset$)
   - (a) Find the pair $(v_i, e_{ij})$ in $G$ with the smallest $e_{ij}$, add $v_i$ to $T$, and remove the pair from $G$
   - (b) Update the the pairs in $G$ with respect to the newly added vertex $v_i$, that is update pair $(v_x, e_{xj})$ to $(v_x, e_{xi})$ if $e_{xi}$ is smaller than $e_{xj}$.

Finding the smallest edge in $G$ can be done at the same time as updating $G$ (we update a pair, and check if the edge is smaller than the smallest we have seen so far). Every update iterates through $G$, which has an average size of $n/2$, and the outer loop iterates over all of the $N$ nodes. The complexity is therefore $O(N * N/2) = O(N^2)$. Wikipedia (2005c) states that Prim’s algorithm can be computed in $O(M + N\log N)$ time by using “a more sophisticated Fibonacci heap”, however in this case this would be slower since the graph is complete, and $M$ is in the order of $N^2$.

When building a minimum spanning tree on a complete graph, explains (Kershenbaum and Slyke, 1972), an algorithm which has a complexity based on the number of edges must have a complexity better than $O(M)$ to beat Prim’s algorithm. The algorithm by Chazelle (2000) is approximately $O(M)$, but it is far more complicated than this algorithm. As a result, we are using Prim’s algorithm to construct minimum spanning trees in our criterion function.

### 5 Summary

We have introduced the key concepts in pattern recognition: pattern representation, feature set reduction and classification. The focus of this paper was largely on feature subset selection, with a some focus on classifiers. Classifiers are important in our research, because the provide a means of evaluating the feature subsets chosen by the different approaches.

Feature subset selection was split up into two parts, subset searching and criterion functions. For both parts, the common algorithms were introduced and analysed. Finally, our criterion function was introduced and explained.

IFS is being developed to experiment with our criterion function. It allows users to test out various combinations of criterion functions, feature subset searchers and classifiers. This is the core of the application, as it enables us to discover which feature subset searcher and classifier best work with our criterion function, and how it performs against existing
methods. Additionally, IFS has graphing capabilities. Currently it is able to graph up to 3 features of a data set, in 3 dimensions. This assists users in evaluating and understanding the feature subsets chosen by different approaches.

**References**


Clausen, J. (1999). Branch and bound algorithms - principals and examples. Department of Computer Science, University of Copenhagen, Denmark.


URL: http://en.wikipedia.org/wiki/Kruskal%27s_algorithm

URL: http://en.wikipedia.org/wiki/Minimum_spanning_tree

URL: http://en.wikipedia.org/wiki/Prim%27s_algorithm