Feature Selection

Introduction:

Features are the essential part of the dataset and are used to describe the dataset but too many features makes the dataset clumsy to predict as well as makes it difficult to visualise. The goal here is to select only the necessary features. Here we should aim to select the features which are good at discriminating the data points coming from different classes. The discriminating property generally means having large between-class variance and small with-in class variance.

Preprocessing

1) Outlier Removal:

An outlier removal is defined as the point that lies very far from the mean of the corresponding random variable. The distance is measured with respect to a given threshold. The given threshold is usually a number of times the standard deviation. For a normally distributed random variable, a distance of two times the standard deviation covers 95% of the data points and distance of three times a standard deviation covers 99% of the data points.

2) Data Normalization:

Usually in a large dataset, the different features contain different dynamic ranges. Thus features with a large values may have a larger influence in the cost function than the features with small values, although this does not necessarily reflect their respective significance in the design of the classifier. The problem is overcome by normalising the features so that their values lie within similar ranges. A straightforward technique is via the respective estimates of the mean and variance.

For N available data of the k^{th} feature we have,

$$Mean = \frac{1}{N} \Sigma x_{ik} \quad \forall k = 1, 2, ..., l$$

$$Variance = \frac{1}{N-1} \Sigma (x_{ik} - \bar{x}_k)^2$$

$$\hat{x}_{ik} = \frac{x - Mean}{Variance}$$

In other words the resulting normalized feature will now have zero mean and unit variance.

3) Missing Data:

In practice certain features might be missing from some feature vectors. The most traditional way of dealing with these missing data include schemes that complete the missing values by

(a) Zeros

(b) The unconditional mean, computed from the available values of the respective feature

(c) The conditional mean, if one has the estimate of the pdf of the

missing values given the observed data.

Completing the missing values in a set of data is known as imputation.

A popular alternative to the above methods, rather naively approaches is known as imputing from a conditional distribution. The idea here is to impute by respecting the statistical nature of the missing values. Under this rationale, missing values are not replaced by statistical means or zeros but by a random draw of from a distribution. Let x_{com} be the complete feature vector and x_{com} be the missing components and the rest are observed components represented by x_{obs}

$$x_{com} = [x_{obs} \quad x_{mis}]$$

Under the assumption that the probability of missing a value does not depend on the value itself, imputing from a conditional distribution means to simulate a draw from the following conditional pdf

$$p(x_{mis} | x_{obs} ; \theta) = p(x_{obs} , x_{mis} ; \theta) / p(x_{obs} ; \theta)$$

where

$$p(x_{obs} ; \theta) = \int p(x_{com} ; \theta) \, dx_{mis}$$

Where θ is unknown set of parameters. In practice, an estimate of $\widehat{\theta}$ of θ must be obtained from x_{obs} .

The Multiple Imputation(MI) is one step beyond the precious methodology, usually referred to as Single Imputation(SI). In MI, for each missing value, m > 1 samples are generated. The results are then combined appropriately so that certain statistical properties are fulfilled. MI can be justified as an attempt to overcome uncertainties associated with the estimation of parameter θ. Here one can use different parameters, $\widehat{\theta}_i$, $i = 1, 2, 3, \ldots , m$, and draw the m samples from

$$p(x_{mis} | x_{obs} ; \widehat{\theta}_i) , \; i = 1, 2, 3, \ldots , m$$

A way to approach this problem is via Bayesian inference arguments where the unknown parameter vector is treated as a random one described by a posterior probability.

Peaking Phenomenon

In order to design a classifier with good generalization performance, the number of training points, N, must be large enough with respect to the number of features , l, that is, the dimensionality of the feature space. The larger the N the better the estimate, since we can filter out the effects of noise and also minimize the effects of outliers.

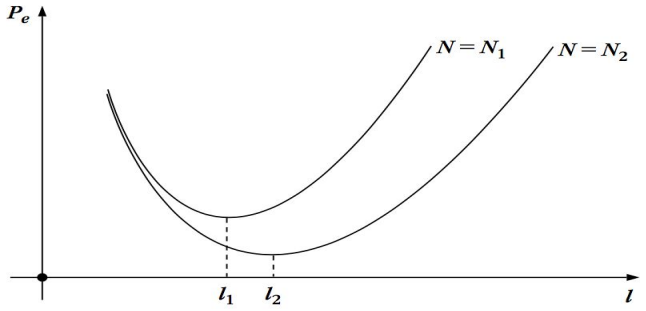
By this we can come to a conclusion that:

1) If for any l the corresponding pdf is known, then we can perfectly discriminate the two classes by arbitrarily increasing the number of features

2) If the pdf s are not known and the associated parameters must be estimated using a finite training set, then the arbitrary increase of the number of features leads to the maximum possible value of the error rate, that is, P_e = 0.5. This implies that under a limited number of training data we must try to keep the number of features to a relatively low number.

In practice, for a finite N, by increasing the number of features one obtains an initial improvement in performance, but after a critical value further increase of the number of features results in an increase of the probability of error. This phenomenon is also known as *peaking phenomenon.*

Let l denote the number of features and let N denote the size of the training set. For $N2 >> N1$, the error values corresponding to $N2$ are lower than those resulting for $N1$, and the peaking phenomenon occurs for a value $l2 > l1$. For each value of N, the probability of error starts decreasing with increasing *l* till a critical value where the error starts increasing. The minimum in the curves occurs at some number $l = \frac{N}{\alpha}$, where , usually, takes values in the range of 2 to 10. Consequently, in practice, for a small number of training data, a small number of features must be used. If a large number of training data is available, a larger number of features can be selected that yield better performance.



x-axis => number of features
y-axis => Error rate

Although the above scenario covers a large set of “traditional” classifiers, it is not always valid. We have already seen that adopting an appropriate kernel function to design a nonlinear SVM classifier implies a mapping to a high-dimensional space, which can even be of infinite dimension. In spite of the fact that one now works in almost empty spaces (*N* is much less than the dimensionality of the space), the generalization performance of the SVM classifiers can be very good. The secret to that was disclosed to us fairly recently. It is not the number of parameters that really controls the generalization performance, under finite *N* , but another quantity. For some types of classifiers,this quantity is directly related to the number of parameters to be estimated and the dimensionality of the feature space. However, for some classifiers, such as the SVM, this quantity can be controlled independent of the dimensionality of the feature space.

Feature Selection Approach:

According to the type of optimality rule that one chooses to work with, the feature selection task is classified into two categories:

1) Filter approach
2) Wrapper approach

1) Filter Approach:

In this approach, the optimality rule for feature selection is independent of the classifier,which will be used in the classifier design stage. For each combination we should use one of the separability criteria and select the best feature vector combination. We obtain the total number number of vectors as

$$^{m}C_{l} = m!/b!(m-b)!$$

This is a very large number even for a small number of $m,$ the total number of features and $b,$ total number of optimal features. Furthermore, in many practical cases the number b is not even known *a priori*. Thus,one has to try feature combinations for different values of b and select the“best”value for it (

beyond which no gain in performance is obtained) and the corresponding"best" b-dimensional feature vector.

2) **Wrapper Approach:**

Sometimes it is desirable to base our feature selection decision not on the values of an adopted class separability criterion but on the performance of the classifier itself. That is, for each feature vector combination the classification error probability of the classifier has to be estimated and the combination resulting in the minimum error probability is selected. This approach may increase the complexity requirements even more, depending, of course, on the classifier type.

For both approaches, in order to reduce complexity, a number of efficient searching techniques have been suggested. Some of them are suboptimal and some optimal (under certain assumptions or constraints).

Feature Subset Selection Algorithms:

1) Sequential Backward Selection (SBS):

This is one of the Sub-optimal searching techniques. Starting from the full set, sequentially remove the feature x^- that least reduces the value of objective function J($Y - x^-$)

Removing a feature may actually increase the objective function $J(Y_k - x^-) > J(Y_k)$; such functions are said to be non-monotonic.

Algorithm:

1) Start with the full set $Y_0 = X$
2) Remove the worst feature x^- = arg max $J(Y_k - x)$, where $x \in Y_k$
3) $Y_{k+1} := Y_k - x^-$; k := k+1
4) Go to 2. Repeat this process N - b times, where N is the total number of features and b is the required number of features.

--Sequential Backward Selection works best when the optimal feature subset is large, since SBS spends most of its time visiting large subsets.

--The main limitation of SBS is its inability to reevaluate the usefulness of a feature after it has been discarded.

Thus starting from total number of features, at each step we drop out one feature from the best combination until we obtain a vector of b features. Obviously, this is a *suboptimal* searching procedure, since nobody can guarantee that the optimal two- dimensional vector has to originate from the optimal three-dimensional one. The number of combinations searched via this method is,

$$1 + 1/2((m+1)m - b(b+1))$$

which is substantially less than that of the full search procedure.

2) Sequential Forward Selection (SFS):

Here, the reverse to the preceding procedure is followed. Starting from the empty set, sequentially add the features x^+ that maximizes the $J(Y_k + x^+)$ when combined with the features Y_k that have already been selected

Algorithm:

1) Start with empty set $Y_0 = \{\Phi\}$
2) Select the next best feature x^+ = arg max $J(Y_k + x)$
3) $Y_{k+1} := Y_K + x^+ ; \; k := k+1$
4) Go to 2. Repeat this b times, where b is the required number of features.

--SFS performs best when the optimal subset is small

- When the search is near the empty set, a large number of states can be potentially evaluated
- Towards the full set, the region examined by SFS is narrower since most features have already been selected

--The search space is drawn like an ellipse to emphasize the fact that there are fewer states towards the full or empty sets

- The main disadvantage of SFS is that it is unable to remove features that become obsolete after the addition of other features

Let m be the total number of features, and let b be the optimal number of features then we can show that the number of combinations searched with this procedure is

$$bm - b(b-1)/2$$

Thus, from a computational point of view, the backward search technique is more efficient than the forward one for b closer to m than to 1.

3) Plus 'L' minus 'R' method (LRS):

Plus 'L' minus 'R' is a generalization of Sequential Forward Selection algorithm and Sequential Backward Selection algorithm. If L>R, LRS starts from the empty set and repeatedly adds 'L' features and removes 'R' features. If L<R, LRS

starts from the full set and repeatedly removes 'R' features followed by 'L' feature additions.

Algorithm:

1) If L>R, then start with the empty set $Y = \{\Phi\}$ else start with full set $Y = X$ Goto step 3.
2) Repeat L times X^+ = arg max $[J(Y_k + X)]$; $x \in Y_k$ and $Y_{k+1} := Y_k + X^+$; $k := k + 1$
3) Repeat R times X^- = arg max $[J(Y_k - X^-)]$; $x \in Y_k$ and $Y_{k+1} := Y_k + X^-$; $k := k + 1$
4) Goto step 2

--LRS attempts to compensate for the weaknesses of SFS and SBS with some backtracking capabilities.

--Its main limitation is the lack of a theory to help predict the optimal values of L and R.

Criterion Functions:

1) Correlation based criterion function:

There are many criterion functions in this field of research. One among them is Correlation Based Criterion Function which is given by,

$$J(Y_m) = (\Sigma \rho_{ic}) / (\Sigma\Sigma \rho_{ij})$$

where ρ_{ic} is the is the correlation coefficient between feature i and the class label and ρ_{ij} is the correlation coefficient between features i and j. Correlation between any two vectors a and b is given by,

$$\rho_{ab} = (\Sigma a_i b_i) / \sqrt{(\Sigma a_i^2 \ \Sigma b_i^2)}$$

2) Fisher's Discriminant Ratio(FDR):

Fisher's Discriminant Ratio is also one of the most widely used criterion function. This is given by

$$FDR_1 = \sum_{i}^{M} \sum_{j \neq i}^{M} \frac{(\mu_i - \mu_j)^2}{\sigma_i^2 + \sigma_j^2}$$

where the subscripts *i*, *j* refer to the mean and variance corresponding to the feature under investigation for the classes *i* , *j* , respectively.

FDR is sometimes used to quantify the separability capabilities of individual features. It reminds us of the test statistic *q* appearing in the hypothesis statistical tests dealt with before.

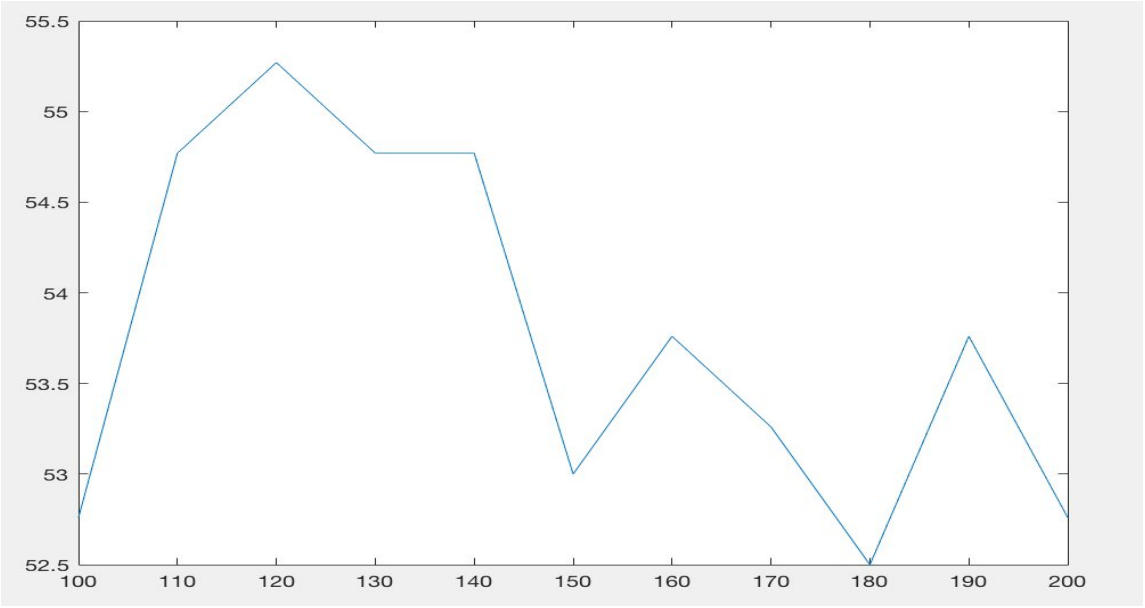
The Below outputs are for Arrhythmia data set

Note: The original dataset with 279 features yielded with 48% for k=5

1) Number of selected features and their respective accuracy

Number of Features	Accuracy
100	52.76%
110	54.77%
120	55.27%
130	56.77%
140	54.77%
150	53.00%
160	53.76%
170	53.26%
180	52.50%
190	53.76%
200	52.76%

Below is a plot of number of features v/s varying accuracy



x-axis=> number of features; y-axis=> accuracy

The Below outputs are for Internet Advertisement data set

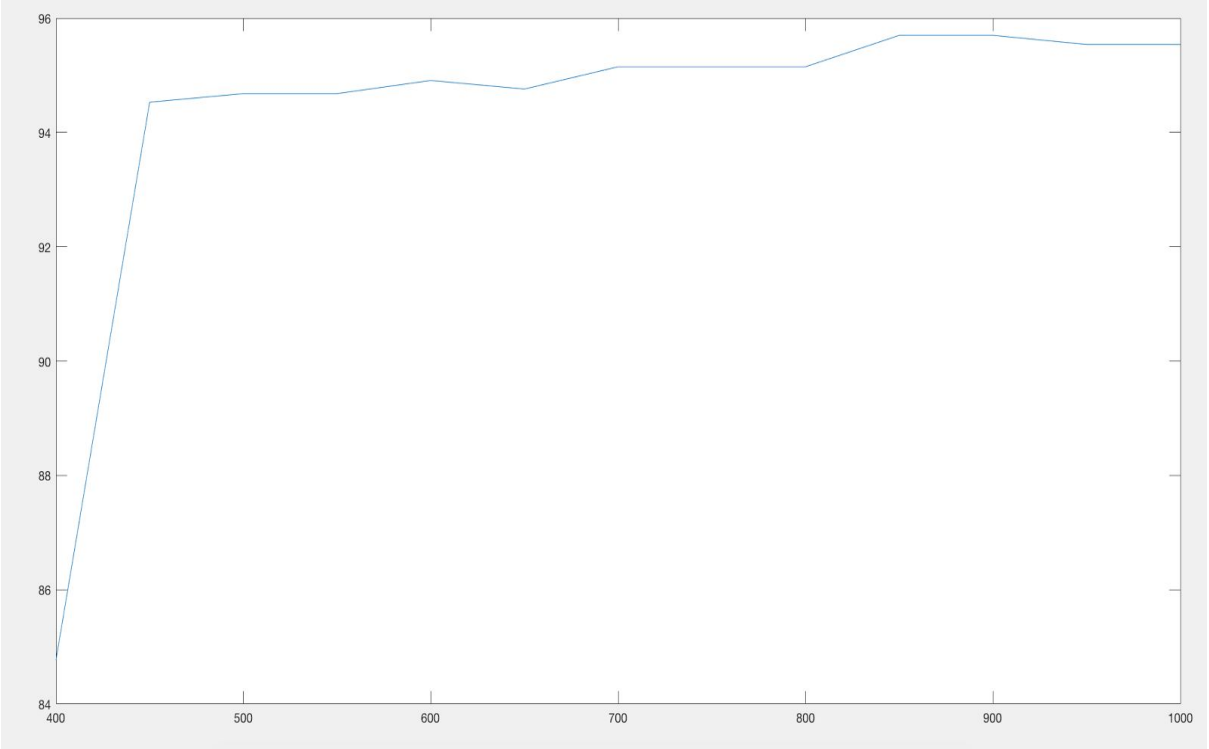
Note: The original dataset with 1558 features yielded with 95.93% for k=5

1) Number of selected features and their respective accuracy

Number of Features	Accuracy
400	84.77%
450	94.53%
500	94.68%
550	94.68%
600	94.91%
650	94.76%
700	95.15%
750	95.15%
800	95.15%
850	95.70%
900	95.70%
950	95.54%

1000	95.54%

Below is a plot of number of features v/s varying accuracy



x-axis=> number of features; y-axis=> accuracy