Machine Learning Module

Week 5

Lecture Notes 9 & 10

Non-Probabilistic Classification Methods

Mark Girolami
girolami@dcs.gla.ac.uk
Department of Computing Science
University of Glasgow

May 14, 2006

1 Non-Probabilistic Classification

Last week we studied two approaches to probabilistic classification the first explicitly defined a discriminant function using a linear model and approximate Bayesian inference was employed in obtaining the model parameters. The second approach, generative, makes estimates of the class-conditional probability distributions or densities of the feature vectors. This week we will look at two very popular methods of classification which are not motivated from a probabilistic model of the data. The first is the classical K-Nearest neighbour method and the second is the rather exotic sounding Support Vector Machine.

2 K-Nearest Neighbours

Consider again the example of classifying Male & Female based on measured height. Given the height h of N individuals as our *training* data set i.e. $\mathcal{D} = \{(h_1, t_1), (h_2, t_2), \cdots, (h_N, t_N)\}$ where each t_n takes on the values of Male or Female. Now we get a new height measurement, h_{new}, for an individual and we have to now assign a label (Male or Female) to that individual. To do this we could simply find the closest height value in the *training* set and adopt the target value associated with this closest value. So in other words we find the *Nearest Neighbour* (h_{NN}, t_{NN}) of our test case h_{new} and make the assignment $t_{new} = t_{NN}$. This seems an incredibly simple decision but it is one which in fact will perform well in many applications.

Now then lets say that we generalise our *Nearest Neighbour* rule and take the $K - NearestNeighbours$ to make our decision. In this case we will have one vote from each of the K nearest-neighbours so the sensible thing to do would be to take the *Majority Vote* as the imputed target value for our new classification. If there is no clear majority then a simple random decision could be taken in this case.

This is a very simple classification rule and yet it is very effective in many cases. So now consider the more general case where our objects are represented by a D-dimensional feature vector $\mathbf{x} \in \mathbb{R}^D$ and that we can compute some *distance* between any two vectors in this space, we will denote the distance between example m & n by $\delta(\mathbf{x}_m, \mathbf{x}_n)$. So for a test point $\mathbf{x}_{new}$ the decision on the majority vote obtained from the K smallest values of $\{\delta(\mathbf{x}_{new}, \mathbf{x}_n)\}_{n=1\cdots N}$ will give us our predicted value t_{new}.

2.1 Implementation

There is no model as such in KNN and so no training is required. All the computational effort occurs when making classifications. Here is a naive implementation of the KNN classification method in Matlab.

```
function [error,tpred] = knn_multi_class(X,t,Xtest,ttest,k)

Ntest = size(Xtest,1);
N = size(X,1);
tpred = zeros(Ntest,1);

C = max(t);

for n=1:Ntest
    Dist = sum(( repmat(Xtest(n,:),N,1) - X ).^2,2 );
    [sorted_list,sorted_index] = sort(Dist);
    [max_k, index_max_k] = max(histc(t(sorted_index(1:k)),1:C));
    tpred(n) = index_max_k;
end

error = 100*sum(tpred ~= ttest)/Ntest;
```

2.1.1 Description of Code

A training set of features, in an $N \times D$ dimensional matrix $\mathbf{X}$, and $N \times 1$ target vectors $\mathbf{t}$ are passed to the KNN script along with an $Ntest \times D$ dimensional matrix $\mathbf{X}_{test}$ of test data. The corresponding true target values of the test data are in the $Ntest \times 1$ vector $\mathbf{t}_{test}$. As this naturally accommodates multiple classes then the target values will take on values in $1 \cdots C$ where C is the number of distinct classes.

The value of k, the number of neighbours to be considered is also passed to the function.

We loop over every test point to be considered, and for each one we compute the distance between the test point under consideration and every one of the training points. In this case the distance is simply the squared distance between each point. In the code above an $N \times 1$ vector which has only the values `Xtest(n,:)` is created using the `repmat` command, which simply repeats a matrix or vector by stacking in row or column format depending on the argument passed to it (refer to the online help for details on repmat). We can then compute the squared Euclidean distance

$$\delta(\mathbf{x}_{test}, \mathbf{x}_n) = \sqrt{\sum_{d=1}^{D} (x_{test,d} - x_{nd})^2}$$

for all n in one simple matrix operation which Matlab is particularly efficient in doing.

Now we have to find the K-nearest neighbours which we do by sorting the vector of distances from $\mathbf{x}_{new}$ using, somewhat unsurprisingly, the Matlab `sort` command. The next step is to take the K target values of the K-nearest neighbours and select the dominant class amongst them. We achieve this in one line of Matlab using the original indices of the sorted distances returned by the `sort` command to select the target values of the K-nearest neighbours using `t(sorted_index(1:k))`. We can make a count of the number of occurences of each of the C classes using the `histc` command and then simply find the maximum.

3 Computational Complexity

The computational complexity for a single prediction will be dominated by the function to compute the distances from each training point. For the simple squared distance then the scaling is linear in the dimensionality of the feature vector and the number of training points, $\mathcal{O}(DN)$. The sorting required will scale as around $\mathcal{O}(N \log N)$ which will tend to dominate the overall cost. So clearly as your training set gets larger the testing time for KNN will be adversely affected, although we would of course anticipate improved predictions.

There are a number of ways in which this cost can be reduced, for example using efficient search and data condensation methods. We will not consider these any further in this course however you should be aware that KNN computational scaling can be significantly improved over the naive implementation given here.

4 Distances & Metrics

Whilst the KNN classifier is model free there are in fact 2 'hyperparameters' associated with the method. The most obvious one is the number of neighbours K to employ when making a classification. Selection of this can be made using Leave-One-Out Cross Validation however the other tunable term is the distance function employed. The KNN classifier relies on the definition of an appropriate metric in the defined feature space. This metric gives meaning to the notion of distances within the feature space and hence how similar (or close) one feature vector is to another.

There are four properties that a metric must have for all vectors $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z} \in \mathbb{R}^D$

1. **Non-Negativity**: $\delta(\mathbf{x}, \mathbf{y}) \geq 0$
2. **Reflexivity**: $\delta(\mathbf{x}, \mathbf{y}) = 0 \;\; iff \;\; \mathbf{x} = \mathbf{y}$
3. **Symmetry**: $\delta(\mathbf{x}, \mathbf{y}) = \delta(\mathbf{y}, \mathbf{x})$
4. **Triangle Inequality**: $\delta(\mathbf{x}, \mathbf{y}) + \delta(\mathbf{y}, \mathbf{z}) \geq \delta(\mathbf{x}, \mathbf{z})$

So the Euclidean distance $\delta(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{d=1}^{D}(x_d - y_d)^2}$ satisfies all the above properties and is hence a metric defining a distance in $\mathbb{R}^D$. However if the space is transformed such that each axis is scaled by some arbitrary constant then distance relations between point can be quite different. Figure (1) shows a simple example of axis being scaled and the impact this has on the distance relationships between the three points.

As each dimension can have different scales then it is common practice to normalise, or standardise the vectors. So if there are N data points each point can be set to have a zero-mean value by simply subtracting the sample mean, i.e.

$$\widetilde{\mathbf{x}} \leftarrow \mathbf{x} - \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$$

Likewise the variance of the data can be set to one so that each axis shares the same mean, zero, and a common variance, one.

$$\widetilde{\widetilde{\mathbf{x}}} \leftarrow \frac{N\widetilde{\mathbf{x}}}{\sum_{n=1}^{N}\widetilde{\mathbf{x}}_n^2}$$

Instead of rescaling the data explicitly and then employing the squared distance metric

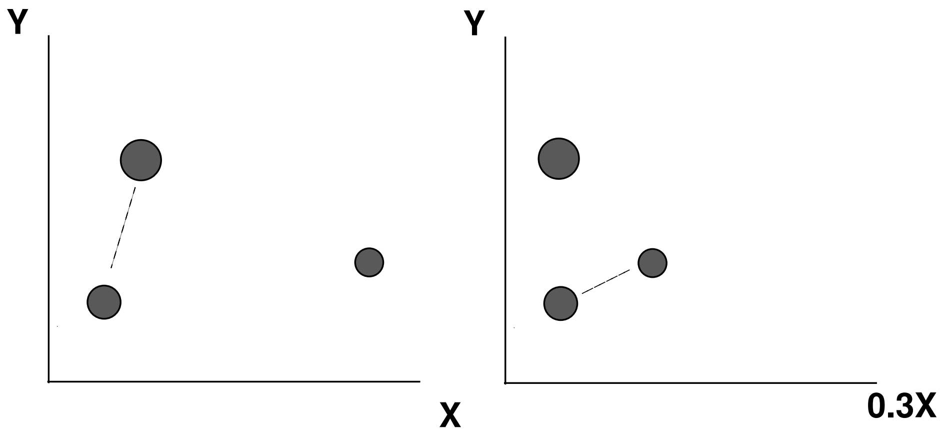

Figure 1: The left hand plot shows three points in a 2-d space defined by axes x and y. The two points with the shortest distance between them is denoted by the dashed line. If we rescale the x-axis by a factor of 0.3, then we see that the closest pair of points changes.

$$\delta(\mathbf{x},\mathbf{y})^2 = \sum_{d=1}^{D}(x_d - y_d)^2 = (\mathbf{x}-\mathbf{y})^{\mathsf{T}}(\mathbf{x}-\mathbf{y})$$

we can actually change the metric directly such that

$$\delta(\mathbf{x},\mathbf{y})^2 = (\mathbf{x}-\mathbf{y})^{\mathsf{T}}\boldsymbol{\Sigma}(\mathbf{x}-\mathbf{y})$$

where $\boldsymbol{\Sigma}$ is a local transformation based on a number of nearest neighbours around the point of interest, say $\mathbf{x}$, which provides a local distortion. The definition of $\boldsymbol{\Sigma}$ is outwith the course content but it does provide a nice of locally adapting the metric used in KNN classification for each test point.

Other metrics often used in KNN is the Minkowski family of metrics defined by

$$\delta(\mathbf{x},\mathbf{y}) = \left(\sum_{d=1}^{D}(x_d - y_d)^p\right)^{\frac{1}{p}}$$

when $p = 1$ this metric is often referred to as the L_1 norm or the *Manhattan* or *city block* distance as it measures the shortest path based on segments which run parallel to the axes. When $p = \infty$ then the L_∞ norm defines the distance between $\mathbf{x}$ and $\mathbf{y}$ as the maximum distance along each of the D axes.

A set theoretic metric which could be used when our objects are defined as sets, for example a bag-of-words representation of a document, is the *Tanimoto* metric defined below

$$\delta(\mathbf{x}, \mathbf{y}) = \frac{n_x + n_y - 2n_{xy}}{n_x + n_y - n_{xy}}$$

where n_x and n_y are the number of elements in each set $\mathbf{x}$ and $\mathbf{y}$. So in last weeks example of text classification where we used a binary model n_x would be the number of distinct words occurring in document $\mathbf{x}$ and n_{xy} is the number of words that both documents share in common.

The choice of metric in KNN is very important and this is an active area of research in Pattern Recognition.

5 KNN Performance

So how well can we expect the KNN classifier to perform? There is some analysis of the KNN rule which indicates that the error rate of a KNN classifier where K=1 is never more than twice the Bayes error rate (this is of course and asymptotic result where $N \to \infty$) but nevertheless it is indicative that good performance with the KNN can be expected. Intuitively we can think of KNN as making a local approximation of the posterior class probability in the region of the test point.

6 Experiments

Taking the two-dimensional data we used last week we standardise the data by making it have zero-mean and unit variance in both dimensions. This can be achieved simply using the Matlab commands

`X = X - repmat(mean(X),size(X,1),1);` and

`X =X./repmat(std(X),size(X,1),1);`

and then simply apply the KNN classifier to the test data and log the test error for a range of values of K from 1 to 100. The minimum test error rate achieved is 8.8% which is rather impressive given that the theoretical optimal is about 8.0%. The Logistic Regression using a Polynomial order of 3 and a prior variance of $\alpha = 10$ achieved an error rate of 9.2% on this data.

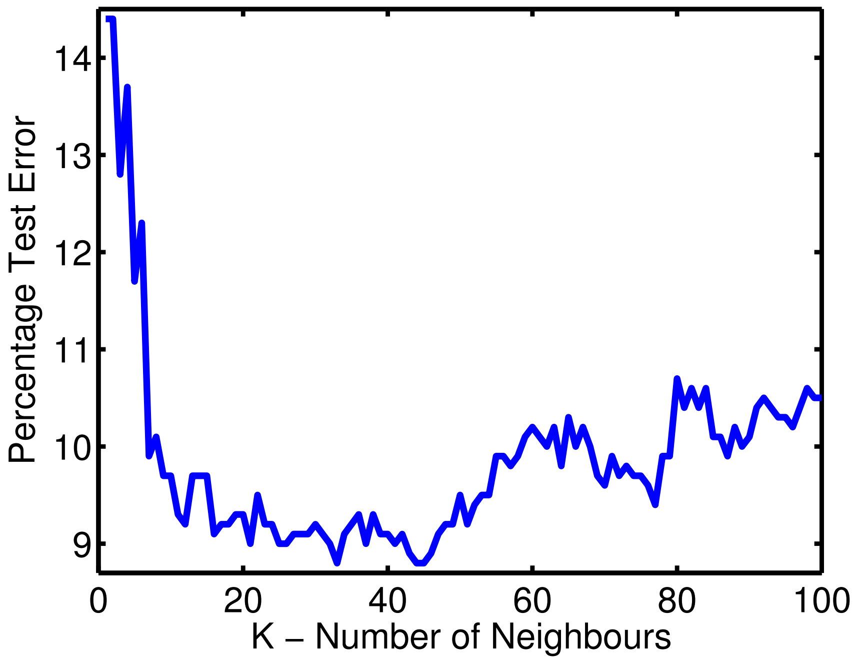

Figure 2: The plot shows the actual percentage of classification errors made on 1000 test points for values of K ranging from 1 to 100. A minimum test error of 8.8% is achieved using $K = 33$. This compares very favourably to the theoretical optimal Bayes Error rate achievable of around 8.0%.

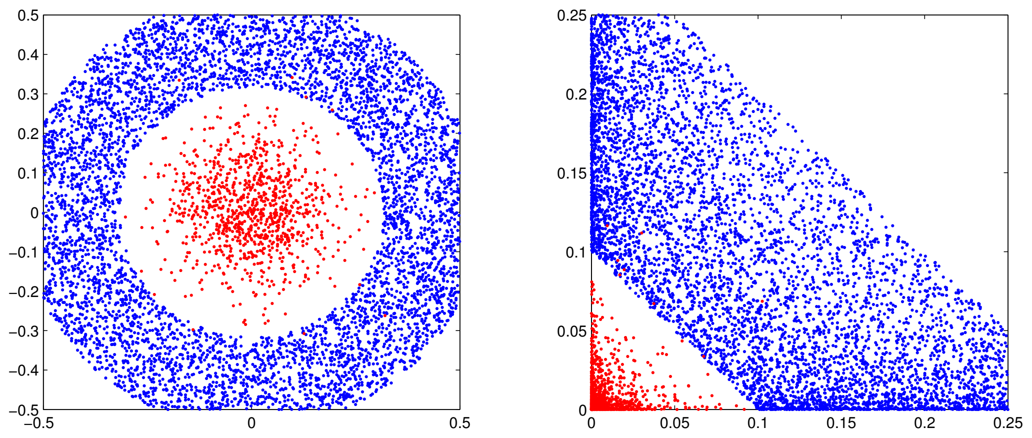

Figure 3: The left hand plot shows the original data comprising of two classes one class is distributed as an annular ring whilst the second class is a simple Gaussian cloud. If we perform a simple transformation of the feature space into a polynomial of order 2, then we see that the transformed points of each of the two classes are such that they can be separated in a linear manner. This is in contrast to the highly nonlinear nature of the separating boundary in the original space. So by this simple transformation we have converted what was a nonlinear classification problem into a linear one.

7 Distance, Metrics & the Kernel Trick

Let us revisit the previous discussion about distances and metrics. We have already seen that by using polynomial or radial basis expansions of our feature vectors such that $\mathbf{x} \rightarrow \phi(\mathbf{x})$ then we can model functions of arbitrary complexity or define arbitrarily shaped decision surfaces for classification with a linear model. So what we are doing is replacing what is a nonlinear problem in the original feature space to a linear problem in our transformed feature space.

Figure (3) shows this effect rather nicely where a simple polynomial of order two is used to transform the data.

Now then when we compute a Euclidean distance between two objects in their original feature space the distance between is defined simply as

$$\delta(\mathbf{x}, \mathbf{y})^2 = (\mathbf{x} - \mathbf{y})^{\mathsf{T}}(\mathbf{x} - \mathbf{y})$$

If we perform a transformation $\mathbf{x} \rightarrow \phi(\mathbf{x})$ then the distance between the objects in the transformed feature space will be defined as

$$\begin{aligned}\delta(\phi(\mathbf{x}), \phi(\mathbf{y}))^2 &= (\phi(\mathbf{x}) - \phi(\mathbf{y}))^\mathsf{T}(\phi(\mathbf{x}) - \phi(\mathbf{y})) \\ &= \phi(\mathbf{x})^\mathsf{T}\phi(\mathbf{x}) + \phi(\mathbf{y})^\mathsf{T}\phi(\mathbf{y}) - 2\phi(\mathbf{x})^\mathsf{T}\phi(\mathbf{y})\end{aligned}$$

We can see that the distance is simply defined by the inner-products computed in the transformed feature space. Now here is where the world famous kernel trick comes in.

A *kernel* function is defined for all $\mathbf{x}$ and $\mathbf{y}$ from a feature space $\mathcal{X}$ such that

$$K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^\mathsf{T}\phi(\mathbf{y})$$

where ϕ is a mapping from the original feature space $\mathcal{X}$ to an inner-product feature space $\mathcal{F}$ i.e. $\phi : \mathbf{x} \rightarrow \phi(\mathbf{x}) \in \mathcal{F}$.

Lets think of a simple example. Let $\mathbf{x}$ and $\mathbf{y} \in \mathbb{R}^2$ and lets define the mapping

$$\phi : \mathbf{x} = (x_1 x_2)^\mathsf{T} \rightarrow \phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)^\mathsf{T} \in \mathcal{F} = \mathbb{R}^3$$

So we can evaluate the inner-product in $\mathcal{F}$ as

$$\begin{aligned}\phi(\mathbf{x})^\mathsf{T}\phi(\mathbf{y}) &= (x_1^2, x_2^2, \sqrt{2}x_1x_2)(y_1^2, y_2^2, \sqrt{2}y_1y_2)^\mathsf{T} \\ &= x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2 \\ &= (x_1y_1 + x_2y_2)^2 \\ &= \left(\mathbf{x}^\mathsf{T}\mathbf{y}\right)^2\end{aligned}$$

So this is a pretty cool result as it shows that we can compute the inner-product in feature space $\mathcal{F}$ by simply computing the kernel function $K(\mathbf{x}, \mathbf{y}) = \left(\mathbf{x}^\mathsf{T}\mathbf{y}\right)^2$ in the original feature space. So this means that we do not need to explicitly define and compute the mapping ϕ to compute inner-products in $\mathcal{F}$. So we compute distances between points in $\mathcal{F}$ simply by computing the kernel function on the points in $\mathcal{X}$.

$$\begin{aligned}\delta(\phi(\mathbf{x}), \phi(\mathbf{y}))^2 &= (\phi(\mathbf{x}) - \phi(\mathbf{y}))^\mathsf{T}(\phi(\mathbf{x}) - \phi(\mathbf{y})) \\ &= K(\mathbf{x}, \mathbf{x}) + K(\mathbf{y}, \mathbf{y}) - 2K(\mathbf{x}, \mathbf{y})\end{aligned}$$

There are a number of properties which have to be satisfied for a function K to be a valid kernel function. The main one being that it is symmetric

$K(\mathbf{x}_i, \mathbf{x}_j) = K(\mathbf{x}_j, \mathbf{x}_i)$ and an $N \times N$ dimensional matrix whose elements are $K(\mathbf{x}_i, \mathbf{x}_j) \;\; \forall \;\; i, j = 1 \cdots N$ must be positive semi-definite. There is a whole class of kernel functions which can be employed and we have already met $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(\beta |\mathbf{x}_i - \mathbf{x}_j|^2)$ in previous laboratory work.

8 Support Vector Machines

We have already met discriminative classifiers which directly provide a discriminant function of the form

$$f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^{\mathsf{T}} \phi(\mathbf{x})$$

Consider a simple binary linear discriminant function (separating two classes) operating on a two-dimensional feature vector

$$g(\mathbf{x}; w_2, w_1, w_0) = w_2 x_2 + w_1 x_1 + w_0 = \mathbf{w}^{\mathsf{T}} \mathbf{x} + w_0$$

where we now denote $\mathbf{w}$ as the column vector comprising of w_2 and w_1. If our class labels (target values) take on the values of ± 1 for each of the two classes then our decision function would simply test whether $g(\mathbf{x}; w_2, w_1, w_0)$ was positive or negative, in which case decisions are made based on the sign of the linear expansion of the feature vectors i.e.

$$f(\mathbf{x}; w_2, w_1, w_0) = \text{sign}(w_2 x_2 + w_1 x_1 + w_0) = \text{sign}(\mathbf{w}^{\mathsf{T}} \mathbf{x} + w_0)$$

Now let us say that we have N data points in our available sample for training $(\mathbf{x}_1, t_1) \cdots (\mathbf{x}_N, t_N)$ and we assume that the two classes are completely linearly separable then the training data will be correctly classified if

$$t_n(\mathbf{w}^{\mathsf{T}} \mathbf{x} + w_0) > 0 \;\; \forall \;\; n = 1 \cdots N$$

Now then if we cast our minds far back to last weeks lecture on probabilistic classification we saw that there was a posterior distribution over the parameters, $\mathbf{w}$, of our classifier, which of course means that there will be a number of $\mathbf{w}$ which could separate the training data perfectly see Figure (4).

Whilst in the Bayesian methodology you would take the posterior average of the $\mathbf{w}$ when making class predictions the Support Vector Machine (SVM) exploits some results which come from Statistical Learning Theory to choose which of the possible decision functions should be used.

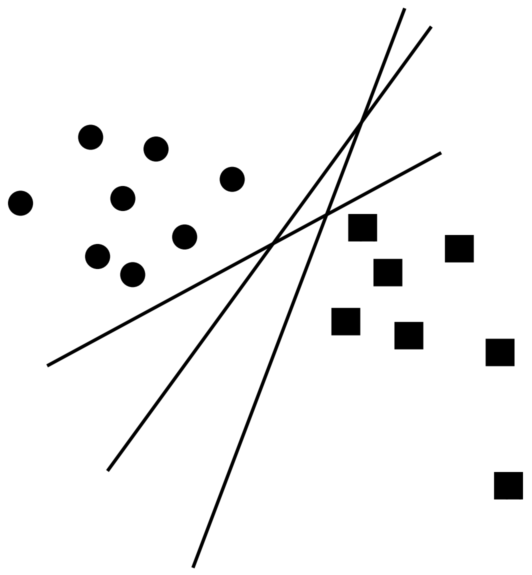

Figure 4: The samples of two classes denoted by sold circles and squares can be separated perfectly with no miss-classifications by a number of possible $\mathbf{w}$ some examples of which are drawn on this cartoon.

There are theoretical results from analysis of the generalisation error of a binary classifier which show that an upper-bound on the generalisation error is inversely proportional to the perpendicular distance from the separating hyperplane, $\mathbf{w}$, and a hyperplane through the closest points from both classes see Figure(5). This distance is called the *margin* in the SVM literature and so to minimise the bound on the generalisation error we would then seek to maximise the *margin* of our classifier.

So we will seek the hyperplane $\mathbf{w}$ which provides the maximum margin of separation between the two classes. Vector geometry shows that the distance of an arbitrary point $\mathbf{x}$ in some D-dimensional space to a hyper-plane H within this space which is defined by all points that satisfy $\mathbf{w}^{\mathsf{T}}\mathbf{x} + w_0 = 0$ is given by

$$\frac{\mathbf{w}^{\mathsf{T}}\mathbf{x} + w_0}{||\mathbf{w}||}$$

So if $\mathbf{x}_1^*$ and $\mathbf{x}_2^*$ are the closest points from each class to the separating hyper-

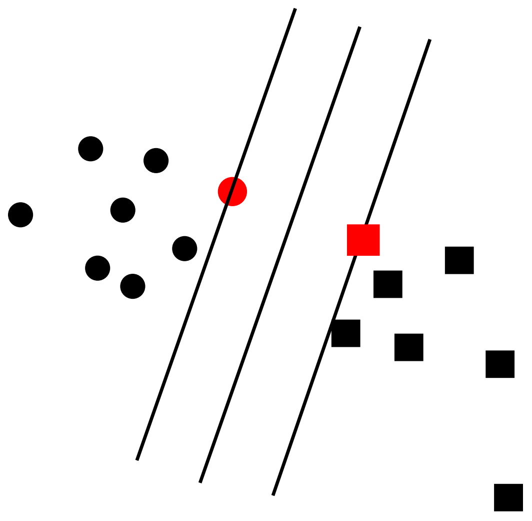

Figure 5: The hyper-plane which maximises the margin.

plane $\mathbf{w}$ then the margin of separation is

$$\frac{\mathbf{w}^\mathsf{T}\mathbf{x}_1^* + w_0}{||\mathbf{w}||} - \frac{\mathbf{w}^\mathsf{T}\mathbf{x}_2^* + w_0}{||\mathbf{w}||} = \frac{\mathbf{w}^\mathsf{T}}{||\mathbf{w}||}(\mathbf{x}_1^* - \mathbf{x}_2^*)$$

As the SVM discriminant function is $\text{sign}(\mathbf{w}^\mathsf{T}\mathbf{x} + w_0)$ then clearly the decision made will be invariant to an arbitrary rescaling of the argument $\mathbf{w}^\mathsf{T}\mathbf{x} + w_0$ in which case we can define a *canonical* hyper-plane $\mathbf{w}$ such that $\mathbf{w}^\mathsf{T}\mathbf{x}_1^* + w_0 = 1$ and $\mathbf{w}^\mathsf{T}\mathbf{x}_2^* + w_0 = -1$ in which case the margin is now simply $\frac{2}{||\mathbf{w}||}$. So to maximise this margin we need to minimise $||\mathbf{w}||$ subject to all the points being correctly classified. The SVM optimisation can be written as

$$\min \frac{1}{2}||\mathbf{w}||^2$$

subject to

$$t_n(\mathbf{w}^\mathsf{T}\mathbf{x} + w_0) \geq 1 \ \ \forall \ \ n = 1 \cdots N$$

and by finding the solution to the above we will be using the $\mathbf{w}$ in our classifier which will minimise the bound on the achievable generalisation error.

What we now have to do is find a way to identify the solution of the constrained optimsation above.

8.1 SVM Optimisation

Optimisation theory is a huge mathematical subject which has applications in about every area of science, technology, economics,... everywhere. We will have to use one result from this theory to make progress in obtaining our SVM classifier and so the optimisation methods we now require are simply going to be stated and used in devising our SVM[1].

Given a constrained optimisation problem of the form

$$\min f(\mathbf{w})$$

subject to

$$\begin{aligned} g_i(\mathbf{w}) &\leq 0 \quad i = 1 \cdots K \\ h_i(\mathbf{w}) &= 0 \quad i = 1 \cdots M \end{aligned}$$

we form the Lagrangian function as

$$\mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{i=1}^{K} \alpha_i g_i(\mathbf{w}) + \sum_{i=1}^{M} \beta_i h_i(\mathbf{w})$$

We now find the maximum of $\mathcal{L}(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ with respect to $\mathbf{w}$ which is denoted as $\theta(\boldsymbol{\alpha}, \boldsymbol{\beta})$ and then we now have to solve the following optimisation problem

$$\max \theta(\boldsymbol{\alpha}, \boldsymbol{\beta})$$

subject to

$$\alpha_i \geq 0 \quad \forall \;\; i = 1 \cdots K$$

Following the above let us now define the Lagrangian function we need for our SVM. Noting that there is only one set of inequality constraints and no equality constraints then

$$\mathcal{L}(\mathbf{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 + \sum_{n=1}^{N} \alpha_n \left(1 - t_n(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + w_0)\right)$$

[1]If you are interested in studying and understanding a bit more about the result we use here the book *Convex Optimisation* is available online at `http://www.stanford.edu/~boyd/cvxbook/`. Be careful and don't print it out in your first flush of enthusiasm it is a rather large book of 730 pages.

where we have defined each $g_i(\mathbf{w}) = 1 - t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + w_0) \leq= 0$ which comes from our original constraint.

We should know the drill by now, to find the stationary point of $\mathcal{L}(\mathbf{w}, w_0, \boldsymbol{\alpha})$ we take derivatives and so

$$\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{w}, w_0, \boldsymbol{\alpha}) = \mathbf{w} - \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n$$

and

$$\frac{\partial}{\partial w_0}\mathcal{L}(\mathbf{w}, w_0, \boldsymbol{\alpha}) = -\sum_{n=1}^{N} \alpha_n t_n = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n t_n = 0$$

Now using the above we need to define our $\theta(\boldsymbol{\alpha})$ so let us plug-in the results to $\mathcal{L}(\mathbf{w}, w_0, \boldsymbol{\alpha})$.

Using the result $\mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n$ we should see that

$$\begin{aligned}\frac{1}{2}||\mathbf{w}||^2 = \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} &= \frac{1}{2}\left(\sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n^\mathsf{T}\right)\left(\sum_{m=1}^{N} \alpha_m t_m \mathbf{x}_m\right) \\ &= \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m \mathbf{x}_n^\mathsf{T}\mathbf{x}_m\end{aligned}$$

Now the second component of our Lagrangian needs to be considered

$$\begin{aligned}&\sum_{n=1}^{N} \alpha_n \left(1 - t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + w_0)\right) \\ =& \sum_{n=1}^{N} \alpha_n - \sum_{n=1}^{N} \alpha_n t_n \mathbf{w}^\mathsf{T}\mathbf{x}_n - w_0 \sum_{n=1}^{N} \alpha_n t_n \\ =& \sum_{n=1}^{N} \alpha_n - \sum_{n=1}^{N} \alpha_n t_n \mathbf{w}^\mathsf{T}\mathbf{x}_n \\ =& \sum_{n=1}^{N} \alpha_n - \sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m \mathbf{x}_n^\mathsf{T}\mathbf{x}_m\end{aligned}$$

So combining the two parts we obtain

$$\theta(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m \mathbf{x}_n^\mathsf{T}\mathbf{x}_m$$

Now this has to be maximised with respect to all α_n, the constraints that $\alpha_n \geq 0 \quad \forall \quad n = 1 \cdots N$ and the additional constraint which emerges from our stationary conditions that is $\sum_{n=1}^{N} \alpha_n t_n = 0$

So at long last we arrive at the SVM optimisation problem So combining the two parts we obtain

$$\max \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m \mathbf{x}_n^{\mathsf{T}} \mathbf{x}_m$$

subject to

$$\begin{aligned} \alpha_n &\geq 0 \quad \forall \; n = 1 \cdots N \\ \sum_{n=1}^{N} \alpha_n t_n &= 0 \end{aligned}$$

There are a number of ways to solve this problem and we will employ a simple quadratic optimisation solver which is written in Matlab. We will not consider the details of such solvers but suffice to say this has been an important area of development for SVM's so that the required optimisation can be solved efficiently for large data sets.

9 Support Vectors

What we find is that a number of the α_n parameters are returned as having zero value from the optimisation. The α_n which have non-zero values are important and as they are associated with each vector in the training sample $\mathbf{x}_n$ these are referred to as the Support Vectors as the *support* the decision boundary between the two classes.

Now as the discriminant function for a new point $\mathbf{x}_{new}$ is of the form $\mathbf{w}^{\mathsf{T}}\mathbf{x}_{new}+w_0$ then we can write the following by noting that, $\mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n$, and we will only need to take a summation over the non-zero α values, in other words sum over the Support Vectors. Also noting that we are relying on an inner-product between the $\mathbf{x}_n$ and $\mathbf{x}_{new}$ we can employ the kernel trick and write the inner-product in the feature space as the kernel function $K(\mathbf{x}_n, \mathbf{x}_{new})$. Then our SVM discriminant function becomes

$$
\begin{aligned}
f(\mathbf{x}_{new};\mathbf{w},w_0) &= \mathrm{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x}_{new}+w_0) \\
&= \mathrm{sign}\left(\sum_{n=1}^{N} t_n\alpha_n\mathbf{x}_n^{\mathsf{T}}\mathbf{x}_{new}+w_0\right) \\
&= \mathrm{sign}\left(\sum_{n\in SV} t_n\alpha_n\mathbf{x}_n^{\mathsf{T}}\mathbf{x}_{new}+w_0\right) \\
&= \mathrm{sign}\left(\sum_{n\in SV} t_n\alpha_n K(\mathbf{x}_n,\mathbf{x}_{new})+w_0\right)
\end{aligned}
$$

Now w_0 can be obtained by noting that as we defined $\mathbf{w}^{\mathsf{T}}\mathbf{x}_1^* + w_0 = 1$ and $\mathbf{w}^{\mathsf{T}}\mathbf{x}_2^* + w_0 = -1$ then adding together we obtain $w_0 = -0.5\mathbf{w}^{\mathsf{T}}(\mathbf{x}_1^* + \mathbf{x}_2^*)$ where each $\mathbf{x}$ are support vectors from each class.

The optimsation problem which we require to solve can, of course, be written in matrix format as below where we denote the diagonal matrix $\mathsf{diag}(\mathbf{t})$ by $\mathbf{\Lambda}$ and the $N \times 1$ dimensional vector of ones as $\mathbf{1}$. The $N \times N$ dimensional kernel matrix $\mathbf{K}$ is the matrix whose elements are $K(\mathbf{x}_i, \mathbf{x}_j)$ which in the linear case is simply $\mathbf{X}\mathbf{X}^{\mathsf{T}}$. This of course opens up the door for more 'exotic' kernel functions to be used in our SVM classifier.

$$
\max \boldsymbol{\alpha}^{\mathsf{T}}\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^{\mathsf{T}}\mathbf{\Lambda}\mathbf{K}\mathbf{\Lambda}\boldsymbol{\alpha}
$$

subject to

$$
\begin{aligned}
\alpha_n &\geq 0 \quad \forall \; n = 1\cdots N \\
\boldsymbol{\alpha}^{\mathsf{T}}\mathbf{t} &= 0
\end{aligned}
$$

A little code example is over the page. Fifty example of two well separated classes are drawn from two Gaussians and the constrained quadratic solver `monqp0` is called[2] we pass the matrix $\mathbf{\Lambda}\mathbf{K}\mathbf{\Lambda}$,the vectors $\mathbf{1}$ & $\mathbf{t}$, a value C, which we shall discuss shortly and a convergence threshold value as parameters. The function then passes back the non-zero α values (support vectors), the value of w_0 and the indices of the support vectors. A contour grid of points $\mathbf{x}$ is created and the value of the SVM discriminant function is computed at

[2]This Matlab implementation was obtained from Stépahne Canu `http://asi.insa-rouen.fr/ scanu/`

each one of these points and the disciminant hyperplane is then plotted out. You should be able to see clearly that the solution is very sparse and a very small portion of the original data points end up being used in the SVM. As usual the code is available at the class website.

9.1 Toy Implementation

```
clear
Step=0.5;
N = 50;
C = 1000;

X = [randn(N/2,2);randn(N/2,2)+[ones(N/2,1).*6 zeros(N/2,1)]];
t=[ones(N/2,1);-ones(N/2,1)];
[alpha,w_0,alpha_index]=monqp0(diag(t)*X*X'*diag(t),ones(N,1),t,C,1e-6);

%Define contour grid
mn = min(X);
mx = max(X);
[x1,x2]=meshgrid(floor(mn(1)):Step:ceil(mx(1)),floor(mn(2)):Step:ceil(mx(2)));
[n11,n12]=size(x1);
[n21,n22]=size(x2);
XG=[reshape(x1,n11*n12,1)
reshape(x2,n21*n22,1)];

f = (t(alpha_index).*alpha)'*X(alpha_index,:)*XG' + w_0;

plot(X(alpha_index,1),X(alpha_index,2),'go'); hold
plot(X(1:N/2,1),X(1:N/2,2),'.') plot(X(N/2+1:N,1),X(N/2+1:N,2),'r.')
contour(x1,x2,reshape(f,[n11,n12]),[0 0]); hold off
```

Figure (6) shows the SVM decision plane and the support vectors for this little toy data set.

For the case where the samples from the two classes may not be completely linearly separable then the SVM optimisation problem can be posed in such a way as to take these possible errors into account. It turns out that a very simple change to the SVM optimisation is required and it changes the

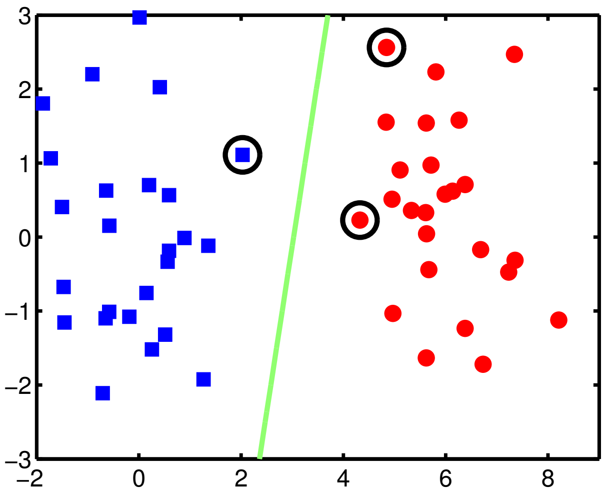

Figure 6: The SVM decision plane separating examples from two classes along with the support vectors which are highlighted. Note that there are only three non-zero α components and so only three points in the data set which are supporting the decision surface.

positivity constraint from $\alpha_n \geq 0$ to $0 \leq \alpha_n \leq C$, for all n, where C is a box constraint parameter.

9.2 SVM Hyper-Parameters

It is clear then that the SVM will have a number of hyper-parameters which will have to be tuned using for example LOOCV. The box constraint parameter C is one and any parameters associated with the kernel function will be the other(s).

If we take the two-dimensional binary class data set which was used last week and employ an SVM in classifying the test examples we can study how well the SVM will do on this example for varying levels of value C. Figure (7) shows the value of the classification error as C is varied from $C = 1$ to $C = 20$ in unit steps when an SVM is trained using a third-order Polynomial kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^\mathsf{T}\mathbf{x}_j)^3$. A minimum value of test error, 9.4%, is achieved at $C = 2$. This is comparable with the best performance achieved with the Bayesian classifier using a cubic polynomial expansion.

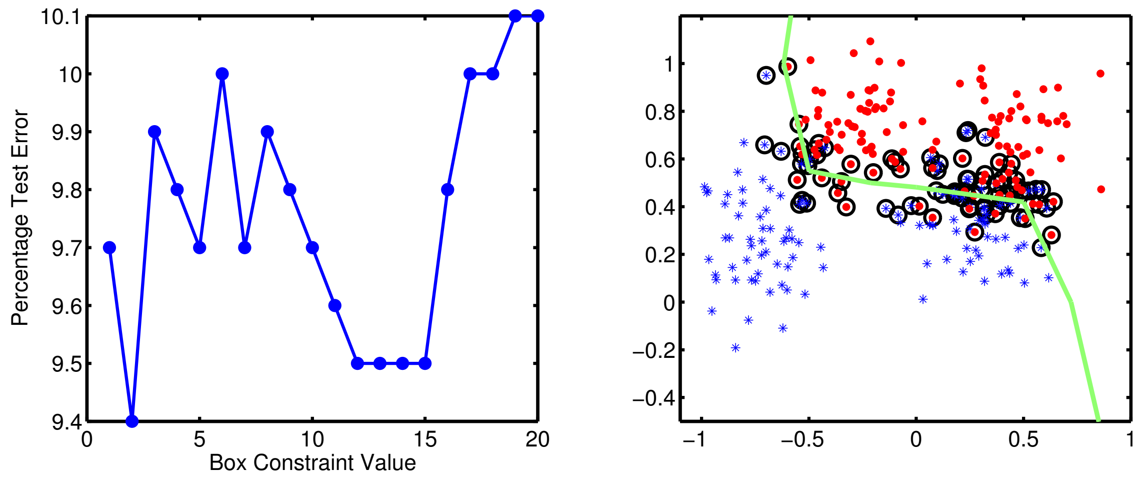

Figure 7: The left hand plot shows the test error achieved for varying values of C when using a polynomial order kernel function. The right hand plot shows the training data and the decision surface. The support vectors are highlighted and they can all be seen to be clumped around the decision surface.

Clearly there are many types of kernel function which could be used and if we were to consider the radial basis style function, which we met in the Week 3 Laboratory then we will need to search over all combinations of width parameter β and box constraint parameter C. Figure (8) shows results of such a search.

10 Conclusions

We have introduced the KNN and SVM classifiers along with the kernel trick in this weeks classes. There has been an explosion of research in the last ten years focused on SVM and kernel methods and this class barely scratches the surface of the literature on these methods. There have been some impressive applications of SVM's in computational biology and Information Retrieval due to their excellent classification performance in general. However, the incredibly simple KNN classifier outperforms the SVM on the two class problem we have been looking at. This is not always the case but it should be said that for many applications the KNN classifier performs rather impressively.

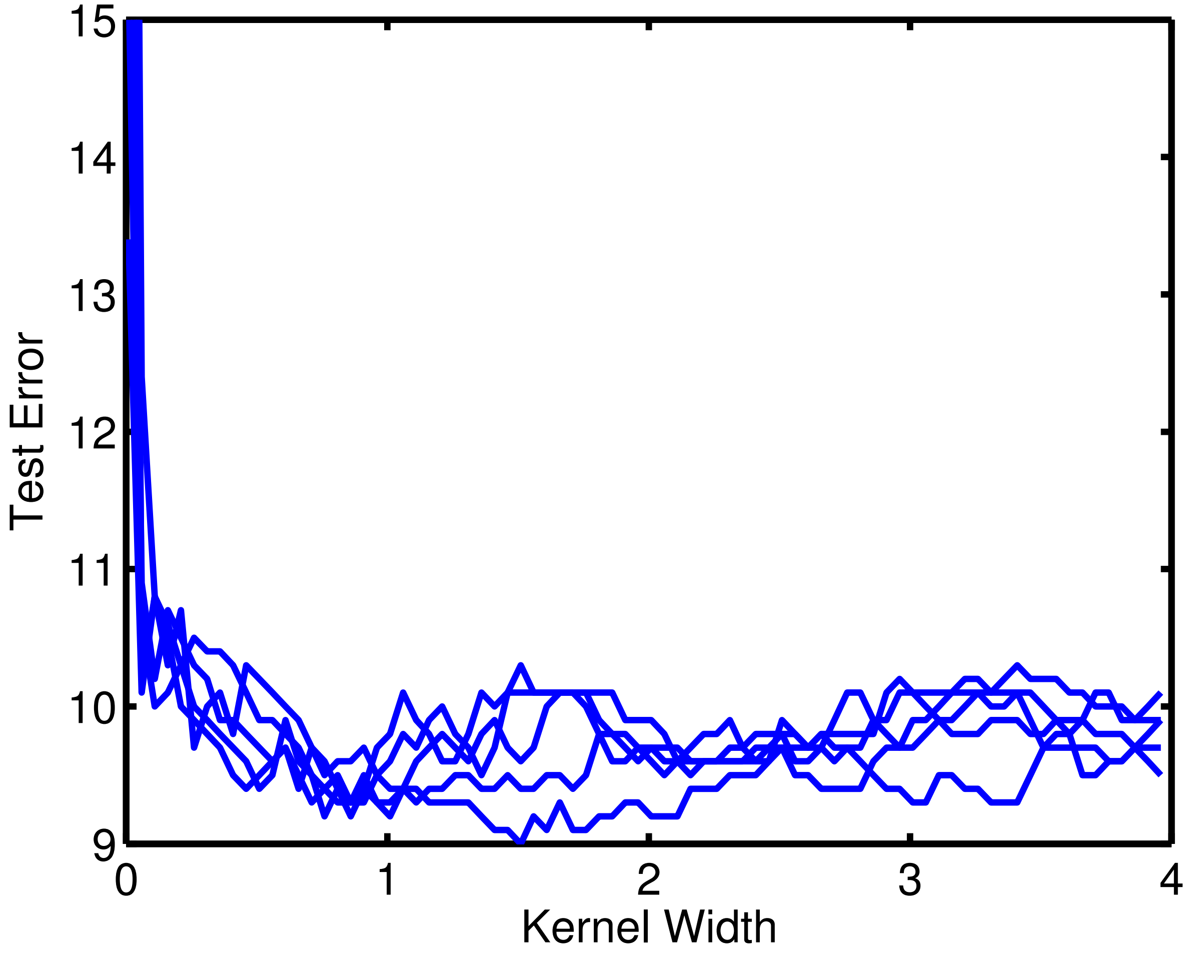

Figure 8: The percentage error achieved by an SVM using a Radial Basis Kernel function with a width parameter ranging from 0.01 to 4.0 in step sizes of 0.05. For each of these ranges a value of C was selected from 1 to 4 and we can see that the minimum test error of 9.0% was achieved with hyper-parameter values of $C = 1$ and $\beta = 1.4$.