Artificial neural networks, advanced course, 2D1433

Lab 2: Support Vector Machines

March 13, 2007

1 Background

Support vector machines, when used for classification, find a hyperplane $\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$ that separates two classes of data. Here $\mathbf{w}$ and b are parameters specifying the hyperplane and $\mathbf{x}$ are points in space. If the data to be classified is written as a set of points $\mathbf{x}_i$ along with class labels $y_i \in \{-1, +1\}$ the condition that the data is correctly classified becomes $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 0$.

1.1 Primal formulation

Of many potential hyperplanes that might fulfil this basic separation criterion, we would like to select one such that *the distance between the hyperplane and any data point is as large as possible.* This distance is known as the *margin.* When the hyperplane is put on a canonical form (more than one parameter set $\mathbf{w}, b$ may describe the same hyperplane) this gives the following optimisation problem. Since we will later derive a "dual" version, we refer to this as the *primal* problem.

$$\text{minimise:} \quad \langle \mathbf{w}, \mathbf{w} \rangle \tag{1}$$

$$\text{when:} \quad y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \quad i \in \{1..m\} \tag{2}$$

The data points $\mathbf{x}_i$ closest to the separating hyperplane will be precisely the ones where equation 2 is satisfied with equality. Those data points are known as *support vectors.* In the non-degenerate case they will be the only data points directly determining the location of the separating hyperplane, in the sense that a small displacement of these points would shift the separating hyperplane. The resulting hard classifier is:

$$f(x) = \operatorname{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b) \tag{3}$$

1.2 Dual formulation

Equivalently, we may write the *dual* problem. We arrive at the dual problem by assigning positive Lagrange multipliers α_i to the inequality constraints in equation 2. The dual problem becomes:

$$\begin{aligned} \text{maximise:} \quad & \textstyle\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle & & (4) \\ \text{when:} \quad & \textstyle\sum_{i=1}^{m} \alpha_i y_i = 0 & & (5) \\ & 0 \leq \alpha_i & i \in \{1..m\} \quad & (6) \end{aligned}$$

We may recognise the support vectors also in the dual formulation. They are the data points for which $\alpha_i > 0$. Once the dual problem has been solved, the primary variables are retrieved as

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \qquad (7)$$

$$b = -\frac{1}{2} \langle \mathbf{w}, \mathbf{x}_p + \mathbf{x}_n \rangle \qquad (8)$$

where $\mathbf{x}_p$ is a positive support vector, having $\alpha_p > 0$ and $y_p = +1$ and $\mathbf{x}_n$ is a negative support vector, having $\alpha_n > 0$ and $y_n = -1$. Alternatively, for better numerical precision, all support vectors may be used to calculate the bias;

$$b = \frac{1}{|\{i : \alpha_i \neq 0\}|} \sum_{i:\, \alpha_i \neq 0} (y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle) \qquad (9)$$

where the denominator in the first factor is the count of support vectors.

2 Build your own SVM

In this part of the lab, you will build your own support vector machine, by directly solving the above optimisation problems. You will be doing this twice, once using the primal formulation, once using the dual formulation.

The data you will be working on is linearly separable, available in `Matlab` format in the file `linsep.mat`. The data file contains two classes. The positive instances are listed in the data structure `classA`, the negative instances in `classB`. For your convenience, there are also ready-made data structures `x` and `y`. They contain all of the data points and class labels, respectively. The vector `permute` describes how they were created from `classA` and `classB`.

2.1 Primal formulation

You will perform a constrained optimisation in the variables $[\mathbf{w}, b]$. You may use the `Matlab` function `fmincon` for this. This function takes as arguments an objective function and the applicable constraints. You should not have any problem passing the objective function $\langle \mathbf{w}, \mathbf{w} \rangle = \mathbf{w}^T \mathbf{w}$ to `fmincon`. Just define it in a separate `myfun.m` file and pass it as `@myfun`. Remember that the elements of the vector $\mathbf{w}$ make up the first part of the variable `X`, in which you are optimising, the bias b is the final element. The constraints (equation 2) are linear. Therefore, they may be expressed as a matrix `A` and a vector `B`; the second and third parameters to `fmincon`. (Hint: The vector `B` can be made to have all its elements equal -1.) When the optimisation has finished, you may find which inequality constraints are active in the return value `LAMBDA.ineqlin`.

Task: Plot the data points for the two classes along with the optimal separating hyperplane. Mark the support vectors in the plot, e.g. by circles; `plot(x,y,'o')`. Make sure that your plot is readable in black and white. You may use the below code to do some of the plotting.

Question: What is the dimension of the $\mathbf{w}$ vector?

Question: How many support vectors do you get?

In the code below, `myfunction` is your classifier function, which is equation 3, *without* the thresholding "sgn" part. It should accept an $\mathbf{x}$ vector as its first argument. Other arguments that your function requires, such as the $\mathbf{w}$ vector and the bias b, can be listed last in the call to `meshapply`, as indicated. The function file `meshapply.m` is found in the lab directory.

```
clf; hold on;
[X,Y] = meshgrid(min(x(:,1)):.1:max(x(:,1)), ...
                 min(x(:,2)):.1:max(x(:,2)));
Z = meshapply(@myfunction, X, Y, [other args]);
contour(X, Y, Z, [-1 0 1]);
plot (classA(:,1), classA(:,2), 'r*')
plot (classB(:,1), classB(:,2), 'b+')
```

2.2 Dual formulation

This time, the optimisation happens over the Lagrangian vector α. What is its dimension? Again, the `Matlab` function `fmincon` may be useful to solve the problem. If you would like, you can calculate the matrix $K_{ij} := \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ (or even $M_{ij} := y_i y_j K_{ij}$) in advance and have the objective function `FUN` access K or M as a global variable. Note that you need linear equality constraints in addition to inequality constraints. You can accomplish this by using the parameters `Aeq` and `Beq` of `fmincon`.

Task: Again plot the data points, the separating hyperplane and mark the support vectors.

Question: How many support vectors do you get this time?

Question: Is the separating hyperplane different from what you got using the primal formulation? Can you see from looking at the geometry of the problem (the data points near the border between classes) if there is a unique maximal margin hyperplane in this case?

3 Linearly non-separable data

The above methods only work for linearly separable data.

Task: Redo the above experiments using the data set `nlinsep.mat`. Do not attempt to fix your code, just run it naively for the new data.

Question: What happens when you try to apply the primal and dual approaches, respectively?

When data are not linearly separable, some data points will necessarily be misclassified. We loosen the bounds in equation 2 to allow for this, but any deviation is penalised in a new objective function (equation 10). The primal problem now reads

$$\text{minimise:} \quad \langle \mathbf{w}, \mathbf{w} \rangle + C \textstyle\sum_{i=1}^{m} \xi_i^{\mu} \tag{10}$$

$$\text{when:} \quad y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i \quad i \in \{1..m\} \tag{11}$$

$$\xi_i \geq 0 \tag{12}$$

The ξ_i measure the degree of misclassification for each data point. Note that they are new variables in the problem. They will be zero for data points that are correctly classified and outside of the margin. C (a coefficient) and μ (an exponent) are positive parameters that determine how misclassifications are penalised.

The dual formulation becomes particularly simple when the parameter $\mu = 1$, so for simplicity we will constrain ourselves to this case for the lab. When $\mu = 1$, the dual problem is the same as before, except for the constraints on the dual variables. These are now also bounded from above:

$$0 \leq \alpha_i \leq C \quad i \in \{1..m\} \tag{13}$$

In addition, we have to restrict the set of data points over which we compute the bias to those with $\xi_i = 0$, meaning $\alpha_i \leq C$[1]. We replace equation 9 with the following:

$$b = \frac{1}{|\{i : 0 < \alpha_i < C\}|} \sum_{i:\, 0 < \alpha_i < C} (y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle) \tag{14}$$

3.1 Classify linearly non-separable data

Modify the *dual* version of your algorithm according to equation 13. You will have to introduce the new constraints involving C from equation 13 and modify the way the bias is calculated according to equation 14.

> **Task:** Classify the linearly non-separable data using your modified algorithm. Plot the data points and the separating hyperplane as before. Based on your visual impression of how the separating hyperplane is positioned, find your favourite value for the parameter C.
>
> **Question (optional):** How does changing the value of C alter the placement of the separating hyperplane and the number of support vectors? Show a representative result. (It may be hard to see what is happening to the hyperplane; it will be easier in the next section.)

4 The kernel trick

It is often advantageous to map data into a higher-dimensional space, since a problem that was not linearly separable in the original space, can often be made

[1] If there are no data points $0 \leq \alpha_i \leq C$ we could in principle still compute the bias from equation 11, but for this lab we will just give up in that case.

so in higher dimensional space. The real power of support vector machines comes from the following realisation: If the data points are mapped to a different space; $x \mapsto \phi(\mathbf{x})$ before finding a hyperplane to separate them, the only operation that needs to be carried out in the new space is the computation of the inner product $\langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle$.

We denote the inner product in the new space $\phi(\Re^m)$ as $K_\phi(\mathbf{x}_1, \mathbf{x}_2) := \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle$ and refer to it as a *kernel function.* For every transformation $\phi(\mathbf{x})$ there exists, by this definition, a kernel $K_\phi(\mathbf{x}_1, \mathbf{x}_2)$.

> **Question (optional):** Not every function $f(\mathbf{x}_1, \mathbf{x}_2) : \Re^m \times \Re^m \mapsto \Re$ is a kernel. There are many functions that cannot be expressed as $f(\mathbf{x}_1, \mathbf{x}_2) = \langle \phi(\mathbf{x}_1), \phi(\mathbf{x}_2) \rangle$ for *any* transformation $\phi(\mathbf{x})$. Can you find an example of a function that cannot be?

4.1 Disposing of w

We now proceed to implement the above claim; that we can express the linear classifier in ϕ-space without any geometric concepts except for the inner product. To do this, we insert equation 7 into equation 3 and 14, moving scalars out of the inner product brackets:

$$f(\mathbf{x}) = \operatorname{sgn}\Big(\sum_{i:\, \alpha_i \neq 0} (\alpha_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle) + b \Big) \tag{15}$$

$$b = \frac{1}{|\{i : 0 < \alpha_i < C\}|} \sum_{i:\, 0<\alpha_i<C} \left(y_i - \sum_{j:\, \alpha_j \neq 0} \alpha_j y_j \langle \mathbf{x}_j, \mathbf{x}_i \rangle \right) \tag{16}$$

Now, we are free to do the separation in any space, without explicit reference to that space. Just replace the inner product brackets in equations 4, 15 and 16 by the kernel function of your choice and the support vector machine will automatically carry out its job in the corresponding space.

4.2 Popular kernels

One common family of kernel functions is the polynomial kernels; $K(\mathbf{x}_1, \mathbf{x}_2) = (\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + 1)^d$. Another is radial basis kernels; $K(\mathbf{x}_1, \mathbf{x}_2) = e^{-\|\mathbf{x}_1 - \mathbf{x}_2\|^2/(2\sigma^2)}$. Linear separation in the original space is of course expressed by the kernel $K(\mathbf{x}_1, \mathbf{x}_2) = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle$. Which kernel to choose, and how to set the kernel parameters, depends on characteristics of the problem data.

4.3 Change your kernel

Rewrite your algorithm again, starting from the last version, the one that solved the dual problem relaxed to linearly non-separable data (section 3.1). Change the algorithm so that it uses the kernelised versions of equations 4, 15 and 16. Don't forget to substitute the kernel function $K(\mathbf{x}_1, \mathbf{x}_2)$ for every inner product bracket $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$. When you have found the α_i you will no longer calculate an explicit $\mathbf{w}$ vector; this vector now lives "hidden" in ϕ-space. Instead, your classifier will be derived from equation 15. As you can see, the computation of this function requires passing the data points $\mathbf{x}_i$ as well as the Lagrange multipliers α_i and the bias b.

Use a polynomial kernel of low degree. You may pass a value for the degree parameter d to the kernel function as a global variable, if you like. (Hint: The implementation becomes easier if you pre-compute the matrix K as suggested in section 2.2.)

> **Task:** Apply your algorithm to the linearly non-separable data. Find values for the parameters C and d that give you reasonable performance on the data set.
>
> **Task:** Illustrate how your classifier divides up the plane using a contour plot, as before. Use equation 15 (without the sgn) for the contour plot. Also plot the training examples, and mark the support vectors.
>
> **Question (optional):** Tweak the C and d parameters. Report on how the two parameters affect classification performance, the number of support vectors ($\alpha_i \neq 0$) and the way the plane is subdivided. For what values would you expect good generalisation performance?

5 Use the `osu-svm` library

The `osu-svm` library is a set of ready-made tools for support vector machines. To use the library, which is free software, add it to your `Matlab` path;

```
addpath /info/annfk07/labs/lab2/osu-svm
```

or download it from `http://sourceforge.net/projects/svm/` if you are working on your own computer.

5.1 Breast cancer data

You will be working with the Wisconsin Prognostic Breast Cancer database. The database contains attributes describing cells, along with information about whether a particular cell was taken from a cancer or not. We would like to use this information to build a classifier that, given cell attributes, would tell us whether it believes a cell to be cancerous. The data set is available in the file `bcw.asc`, which can be loaded directly into `Matlab`. There are 699 data points in the file, one per row. The first nine columns in each row describe cell parameters, the final column contains the class labels.

To begin with, you will divide the available data into training and test sets. Randomise the order of the data points, then designate 400 of them as the training set and the remainder as the test set. Save the two sets into new files and keep them fixed for the duration of this lab, in order to make the comparison between different methods as fair as possible.

> **Task:** Create a linear support vector machine based on your training set. Use the `osu-svm` function `LinearSVC`. Test your support vector machine on both the training and the test sets.
>
> **Question:** What are the classification error rates for the training and test sets? How many support vectors were found?

Question (optional): Print out the cell attributes for the support vectors, along with their class labels. Does it make sense somehow that those particular cells were chosen as support vectors? Browse through the data file and try to figure out yourself what characterises a cancerous cell to answer this.

5.2 Optimising the classifier

You will now work on improving the performance of the classifier you just built. What is important is of course the behaviour on the test set. You will experiment with the following parameters and try to get the best performance out of the classifier;

- Preprocessing
- Choice of kernel
- Kernel parameters

You will select which combinations of preprocessing methods and kernel family to try from the table below. Make at least two selections, by checking boxes in the table, in addition to the already chosen combination ("Normalisation + Polynomial kernel"). If you select "other" as the preprocessing method, you should yourself come up with a way to pre-process the data, before applying the support vector technique. The rest of the methods correspond directly to functions in the `osu-svm` package; try "`help osu-svm`".

Kernel \ Pre-processing	Scaling	Normalisation	Other
Linear			
RBF			
Polynomial		X	

Task: Choose reasonable parameters for each kernel. State for each chosen combination of pre-processing and kernel type the classification error rate on each of the training and test sets.

5.2.1 Tuning the kernel (optional part)

Above you chose kernel parameters rather arbitrarily. To tune the parameters for your chosen kernels, you will use a validation method. The simplest way to do this is to further divide the training set into one set used for actual training and one set used for "validation". This means that you use the validation set, which is really part of the training set, as a kind of internal test set for the training phase. Performance on the validation set determines which kernel parameters you choose.

Divide off the validation set from the training set, as you did the test set before.[2] Train your support vector machine, for different kernel parameters, on the remaining part of the training set, then test it on the validation set. Finally, for the best parameter combination, as measured by validation set performance, calculate the prediction error for the test set.

[2]You may instead choose to use a cross-validation technique if you are familiar with such.

> **Question:** Why should you never use the test set for tuning the kernel parameters? If you try this anyway, you will find that it in fact leads to smaller prediction errors on the test set. Why is it still not a good idea?
>
> **Task:** Find the best kernel parameters for each preprocessing/kernel combination that you chose above. (Again don't use the test set for the parameter search.) State for each case the best parameter choices that you could find and the corresponding errors on each of the training, validation and test sets.

You may use manual search to find the best parameter combinations, or wrap your entire program in a `Matlab` function and use `fmincon` to automatically find the optimal parameters. Note that polynomial kernels may be harder to optimise, since they have more parameters in `osu-svm`.

Good luck!