Machine Learning II
Week 6 Lecture – 13th November 2019
Support Vector Machines 2

Contents

- Review of the linear SVM
- Linear SVM expressed using a Kernel
- Non-linear SVMs
- More than two classes

Recap

- We have a classification problem with $K = 2$ classes and p predictor variables.
- Each of the n observations can be treated as a p-dimensional vector $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$. We will use $p = 2$ for learning purposes.
- For convenience we label the two outcome classes using -1 and 1, so $y_i \in \{-1, 1\}$.
- The aim is to develop a boundary function $f(\boldsymbol{x})$ so that our classifier is determined by the sign of $f(\boldsymbol{x})$.

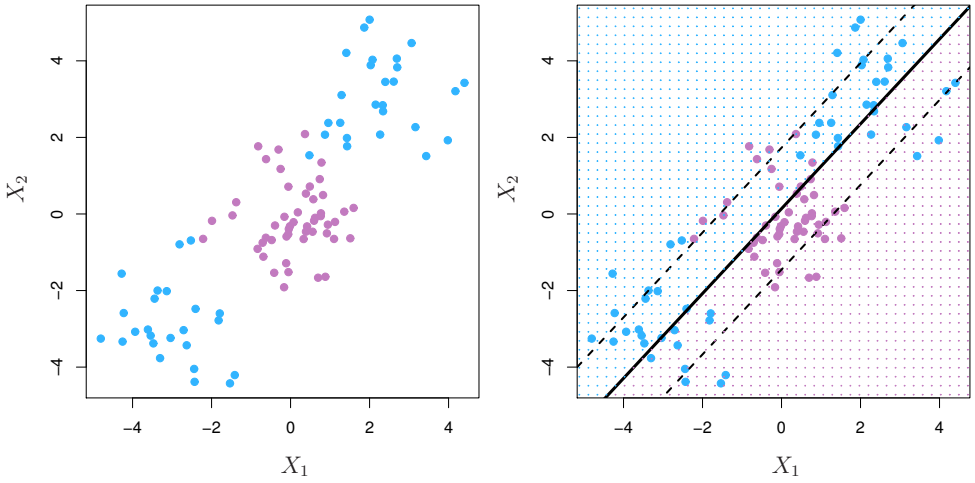
  I.e. for the test data point $\boldsymbol{x}^*$ the classifier is:

  Assign $\widehat{y}^* = 1$ if $f(\boldsymbol{x}^*) \geqslant 0$, otherwise assign $\widehat{y}^* = -1$

- Last week we developed a method to separate the two classes in the training data using a linear boundary called a hyperplane.
- To either side of the boundary there is a margin. In the ideal case no data points lie in the margin and no data points lie on the wrong side of the boundary: *Margin classifier algorithm*.
- In practice it is better to use a “soft margin” so a few points are allowed in the margin or on the wrong side of the boundary, but at a cost, and the cost is held to a specified budget: *Support vector classifier*

Non-linear SVMs

There are situations when a linear boundary $f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$ is insufficient.



One approach is to use a quadratic boundary:

$$f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_2 + \beta_4 x_2^2,$$

and/or interaction terms:

$$f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_2 + \beta_4 x_2^2 + \beta_5 x_1 x_2.$$

For higher degree polynomials or higher values of p the number of betas to estimate increases quickly, and is still not able to find closed loops or wiggly regions.

The kernel approach used for an SVM **transforms** the data from the original data space[1] and transform it into higher dimensional space.

The support vector classifier from last week is then used to find a linear hyperplane in the high-dimension space.

[1] also called the original feature space

Linear SVM expressed using a Kernel

It is possible to express the linear classifier function in an *over-parametrised* form

$$f(\boldsymbol{x}) = \beta_0 + \sum_{i=1}^{n} \alpha_i K(\boldsymbol{x}, \boldsymbol{x}_i),$$

where $\boldsymbol{x}$ is any vector in $\mathbb{R}^p$ and $\boldsymbol{x}_i$ is the i-th data vector.

We now have $n + 1$ parameters to estimate, $\alpha_0, \alpha_1, \ldots, \alpha_n$, instead of $p + 1$, $\beta_0, \beta_1, \ldots, \beta_p$. But this more general framework leads to other advantages.

The function $K(\cdot, \cdot)$ is called a **kernel** function. You met this term in the lecture on loess-smoothing (ML2, Week 3).

Kernel function:
A kernel function is the same function repeated many times centred on different locations, defined by the data. A weighted sum of all the kernel functions defines the predictor function.

The kernel function for the *linear* classifier is the “inner-product”. The inner product of two p dimensional vectors $\boldsymbol{u} = (u_1, \ldots u_2)$ and $\boldsymbol{v} = (v_1, \ldots v_2)$ is

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \sum_{i=1}^{p} u_i v_i.$$

The inner product has many useful features in linear algebra, such as the inner product is zero when the two vectors are orthogonal.
The inner product of the same vector is the squared euclidean distance

$$\langle \boldsymbol{u}, \boldsymbol{u} \rangle = \sum_{i=1}^{p} u_i^2.$$

The inner product is symmetric

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \langle \boldsymbol{v}, \boldsymbol{u} \rangle .$$

$$f(\boldsymbol{x}) = \alpha_0 + \sum_{i=1}^{n} \alpha_i \langle \boldsymbol{x}, \boldsymbol{x}_i \rangle$$

To estimate the parameters $\alpha_0, \alpha_1, \ldots, \alpha_n$ we first obtain the inner products between all the pairs of data points.

As the inner product is symmetric we need to calculate $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ for $i = 1, \ldots, n-1$ and $j = i+1, \ldots n$, which is $\frac{n(n-1)}{2}$ inner products. Each inner product requires order p calculations.
This is expensive, but these values can be calculated just once at the start of the algorithm and do not need to be recalculated with each iteration.

- The fitted α_i coefficients are zero for the non-support vectors. We can re-express the boundary classifier as

$$f(\boldsymbol{x}) = \alpha_0 + \sum_{i \in \mathcal{S}} \alpha_i \langle \boldsymbol{x}, \boldsymbol{x}_i \rangle ,$$

  where $\mathcal{S}$ is the set of support vectors.

- As we expect that the number of support vectors is small compared to n the complexity of the boundary function is reduced significantly.

- In particular: when classifying a test data point $\boldsymbol{x}^*$, we do not need to calculate n inner products just $|\mathcal{S}|$ inner products.

Non-linear SVM

For a non linear boundary, we replace the inner product kernel with another kernel function.

For a curved boundary use a *polynomial kernel function of degree* $d > 1$

$$K(\boldsymbol{u}, \boldsymbol{v}) = (1 + \langle \boldsymbol{u}, \boldsymbol{v} \rangle)^d .$$

Another common choice is the *radial kernel*

$$K(\boldsymbol{u}, \boldsymbol{v}) = \exp\left(-\gamma \sum_{i=1}^{p} (u_i - v_i)^2 \right) , \qquad \text{with } \gamma > 0.$$

A third kernel, which is not as commonly used, is the *neural network kernel*

$$K(\boldsymbol{u}, \boldsymbol{v}) = \tanh\left(\gamma \langle \boldsymbol{u}, \boldsymbol{v} \rangle + \delta \right) .$$

The values d, γ, and/or δ are hyperparameters, which have to be chosen. This is often done using cross-validation.

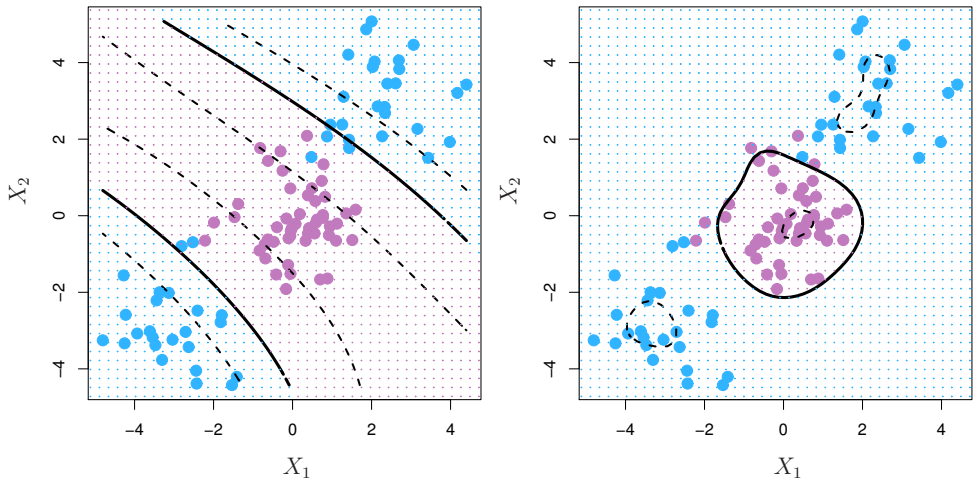
The extra computation at the start, when calculating the pairwise kernel values $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, is offset by allowing a very large extended feature space to be implicitly explored, while remaining computationally efficient.

The same algorithm is used for the linear and non-linear boundaries, the only difference is the type of kernel function to use.

The same concept of the margin is used in linear and non-linear SVMs. The same support vector classifier algorithm in last week's lecture can be applied:

- $\max_{\alpha_0, \alpha_i, \epsilon_i} M$, such that
- $y_i \left(\alpha_0 + \sum_{j=1}^{n} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}_j) \right) \geqslant M(1 - \epsilon_i)$
- subject to the constraints $\sum_{i=1}^{n} \alpha_i^2 = 1$, $\epsilon_i \geqslant 0$ and $\sum_{i=1}^{n} \epsilon_i < C$.

$C \geqslant 0$ is the cost parameter and i=1, . . . n,



Support vector machine classifier with Left: a polynomial kernel of degree 3 and Right: a radial kernel.

Notice that the cubic polynomial representation does not give a boundary which is cubic. The kernel function is cubic.

The plots above show the boundary transformed back to the "input space" $\mathbb{R}^p$ after the linear boundary in the transformed space has been obtained.

Classifying more than two classes

$K > 2$

The SVM approach works well when K=2, because the sign of $y\left(\alpha_0 + \sum_{j=1}^{n} \alpha_i K(\boldsymbol{x}, \boldsymbol{x}_j)\right)$ gives us the predicted class of $\boldsymbol{x}$.

Unfortunately there is no mathematical method to extend the classification to more than two classes.

A work-around lies in artificially creating a $K=2$ problem, running the SVM algorithm and repeating for all possible two-class combinations.

One-versus-one Classification

Let the possible class outcomes be $a_1, \ldots, a_K$.
Create a subset of the data containing only those observations, in which $y_i \in \{a_1, a_2\}$. Define the SVM outcomes

$$y_i' = \begin{cases} +1 & \text{for } y_i = a_1 \\ -1 & \text{for } y_i = a_2 \end{cases}$$

Find the SVM model for this data subset.

Obtain $\frac{K(K-1)}{2}$ SVM Models using each pairwise combination of $a_1, \ldots, a_K$ and associated data subset.

To calculate the predicted values, find the classification each given by each SVM model and use the majority vote method to get the overall classification.

There are now $K(K-1)/2$ SVM algorithms to run, however calculating the kernel function values $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ only needs to be done once on all the data, before taking each subset.

One-versus-all Classification

Set a_1 to be the chosen class and the corresponding outcomes to be

$$y_i' = \begin{cases} +1 & \text{for } y_i = a_1 \\ -1 & \text{otherwise} \end{cases}$$

Find the SVM model using this outcome variable and obtain the SVM boundary function $f_1(\boldsymbol{x})$.

Repeat so that each class $a_1, \ldots, a_K$ is the chosen class, giving K boundary functions $f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x})$.

The predicted value at the point $\boldsymbol{x}'$ is the class giving the largest value of $f_k(\boldsymbol{x}')$, i.e.

$$\widehat{y}' = \arg\max\{f_k(\boldsymbol{x}')\}.$$

We are choosing the binary classifier for which $\boldsymbol{x}'$ is "furthest away" from the boundary. We can think of this as being the class, that we have the most confidence in.

SVMs in the context with other machine learning methods

Machine learning sits at the intersection of computer science and statistical science. SVMs grew out of computer science, where as e.g. logistic regression and spline smoothing were developed by statisticians.

The linear SVM model can be expressed in terms of minimise a loss function and a penalty term

$$\min\left(L(x, y, \beta) + \lambda P(\beta)\right),$$

in exactly the same way as ridge regression and spline smoothing.

Example
Ridge regression (ML1): L is the sum of squared errors of the fitted values and $P(\beta)$ is the sum of the squared parameters.

In linear SVM the loss function is very similar to the implicit loss function in logistic regression. The penalty term in SVM is equivalent to the width of the margin. There is a form of logistic regression called *logistic ridge regression* which is very similar to a linear SVM.

James et al recommends: *When the classes are well separated, SVMs tend to behave better than logistic regression; in more overlapping regimes, logistic regression is often preferred.*

An advantage with logistic regression is that the interpreting the model formula can helps us to understand the structure in the data. This is not easy to do from the results of a SVM model, which concentrates on prediction.

The logistic regression approach can be generalised to non-linear classifiers, although SVMs are *much* more widely used for non-linear classifiers than logistic regression.

Please read section 9.5 in James et al, which discusses this subject in more detail.