Lecture 8 - *Nonlinear Regression*

OBJECTIVE: Linear learning algorithms can be re-written in terms of dot products of the data vectors. We will show that nonlinear algorithms can be re-written in terms of dot products of functions (features) of data vectors. Moreover, these dot products can be interpreted as kernels on the data, so we never need to know the features explicitely. This will enable us to apply all the things we learned with linear methods and yet be able to solve nonlinear problems.

Textbook: Pages 115–117, 144–148 and 387–389

The main idea is to introduce nonlinear functions $\boldsymbol{\phi} \in \mathcal{F}$ that map the data points $\mathbf{x}_i \in \mathbb{R}^d$ to features $\boldsymbol{\phi}(\mathbf{x}_i)$. In this space of features, it is easy to apply linear methods!

⋆

The question is how do we come up with the features $\boldsymbol{\phi}$ to ensure that we are mapping $\mathbf{x}$ to a linear space?

Ridge in Terms of Dot Products

As mentioned earlier, the ridge method is a regularised version of least squares

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \|\mathbf{y}-\mathbf{X}\boldsymbol{\theta}\|_2^2+\delta^2\|\boldsymbol{\theta}\|_2^2$$

where the input matrix $\mathbf{X}\in\mathbb{R}^{n\times d}$ and the output vector $\mathbf{y}\in\mathbb{R}^n$. That is,

$$\begin{bmatrix}\mathbf{x}_1\\ \mathbf{x}_2\\ \vdots\\ \mathbf{x}_n\end{bmatrix}$$

where $\mathbf{x}_i\in\mathbb{R}^d$. The solution is obtained by differentiating the above cost function and equating to zero, yielding:

$$\boldsymbol{\theta}=(\mathbf{X}^T\mathbf{X}+\delta^2\mathbf{I}_d)^{-1}\mathbf{X}^T\mathbf{y}$$

The ridge solution can be written as $\boldsymbol{\theta}=\mathbf{X}^T\boldsymbol{\alpha}=\sum_{i=1}^n\alpha_i\mathbf{x}_i^T$, where $\boldsymbol{\alpha}=\delta^{-2}(\mathbf{y}-\mathbf{X}\theta)$, with $\boldsymbol{\alpha}\in\mathbb{R}^n$.

⋆

Next, we show that $\boldsymbol{\alpha}$ can also be written as follows: $\boldsymbol{\alpha} = (\mathbf{X}\mathbf{X}^T + \delta^2\mathbf{I}_n)^{-1}\mathbf{y}$.

$\star$

Assume we are given a new data point $\mathbf{x}^\star \in \mathbb{R}^{1\times d}$, for which we don't know its label. Using the estimates of θ and α we can write 2 different expressions that enable us to compute $\widehat{\mathbf{y}}$ in terms of $\mathbf{x}^\star$, δ, $\mathbf{X}$ and $\mathbf{y}$ only.

$\star$

Although our new estimate requires that we invert a large $n \times n$ matrix, it can be interpreted in terms of dot products of the data points.

Let $\mathbf{K}$ be a **kernel matrix** with entries $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \mathbf{x}_j^T$. Then, we can re-write the ridge solution as follows:

$\star$

Going Nonlinear

Let's assume we want to fit 2-D data with a quadratic function:

$\star$

It makes sense then to map the i-th input data point $\mathbf{x}_i = (x_{i1}, x_{i2})$ to the space of quadratic features, say

$$\boldsymbol{\phi}(\mathbf{x}_i) = (x_{i1}^2, x_{i2}^2, \sqrt{2}x_{i1}x_{i2})$$

$\star$

The prediction is then:

$$\widehat{y_i} = x_{i1}^2\theta_0 + x_{i2}^2\theta_1 + \sqrt{2}x_{i1}x_{i2}\theta_2$$

This equation is linear in θ. However, you should note that in general we would need an exponential, $O(d^p)$, number of terms for high-order polynomials. Next, we will use kernels to surmount this **curse of dimensionality** and avoid the problem of having to come up with good features $\boldsymbol{\phi}$.

The ridge solution in feature space can be written as

$$\boldsymbol{\theta} = \boldsymbol{\Phi}^T\boldsymbol{\alpha} = \sum_{i=1}^{n} \alpha_i \boldsymbol{\phi}(\mathbf{x}_i)^T,$$

where

$$\boldsymbol{\alpha} = (\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \delta^2\mathbf{I}_n)^{-1}\mathbf{y}$$

and

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(\mathbf{x}_1) \\ \boldsymbol{\phi}(\mathbf{x}_2) \\ \vdots \\ \boldsymbol{\phi}(\mathbf{x}_n) \end{bmatrix}$$

All we have done is replace $\mathbf{X}$ with $\boldsymbol{\Phi}$. However, we usually don't know how to construct $\boldsymbol{\Phi}$. Fortunately, we saw before that $\mathbf{X}$ never appears alone in the solution, but only as a dot product. In the linear case, we were able to replace the dot product by a kernel. We will use the same kernel trick again.

Let us consider the dot product of the features in the previous quadratic regression example:

⋆

$\boldsymbol{\phi}(\mathbf{x}_i)\boldsymbol{\phi}(\mathbf{x}_j)^T =$

Hence, we could adopt the kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i\mathbf{x}_j^T\right)^2$$

and write the nonlinear solution as follows:

$$\boldsymbol{\alpha} = (\mathbf{K} + \delta^2\mathbf{I}_n)^{-1}\mathbf{y}$$

$$\widehat{\mathbf{y}}(\mathbf{x}^\star) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}^\star, \mathbf{x}_i)$$

Under some mild conditions, the **Reproducing Kernel Hilbert Space Theorem** states that we can replace dot products of features with kernels. This means that we only need to specify the kernel function when carrying out nonlinear regression. The most popular kernels are the following:

- Polynomial

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i\mathbf{x}_j^T + b)^p$$

- Gaussian

$$k(x_i, x_j) = e^{-\frac{1}{2\sigma}(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T}$$

- Sigmoid (logistic, neural network)

$$k(x_i, x_j) = tanh(\alpha\mathbf{x}_i\mathbf{x}_j^T - \beta)$$

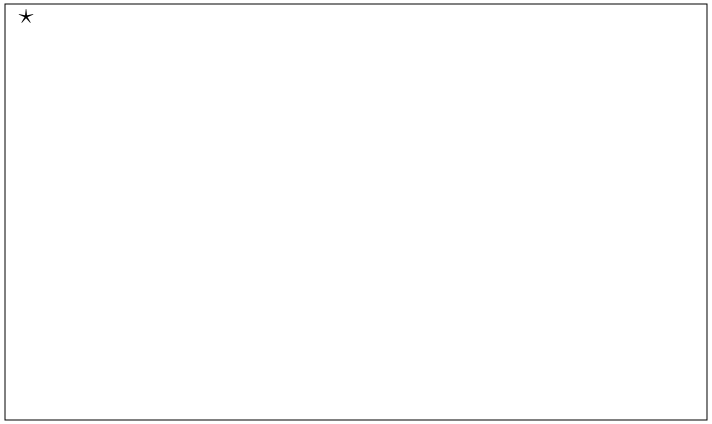
Lecture 9 - *Classification and Support Vector Machines*

OBJECTIVE: Introduce classification and support vector machines.

Textbook: Pages 79–84 and 370–389.

A Simple Linear Classifier

Consider the following classification task:



That is, only one of c outputs (classes) is 1 and the remaining outputs are 0. We could use our standard linear regression estimators with:

$$\begin{bmatrix} y_{11} & \cdots & y_{1c} \\ \vdots & \vdots & \vdots \\ y_{n1} & \cdots & y_{nc} \end{bmatrix} = \begin{bmatrix} x_{10} & \cdots & x_{1d} \\ \vdots & \vdots & \vdots \\ x_{n0} & \cdots & x_{nd} \end{bmatrix} \begin{bmatrix} \theta_{01} & \cdots & \theta_{0c} \\ \vdots & \vdots & \vdots \\ \theta_{d1} & \cdots & \theta_{dc} \end{bmatrix}$$

We can then generate predictions $\widehat{Y} = X\widehat{\theta}$ and choose a class according to

$$class = \arg\max_c \quad \widehat{Y}_c$$

This is an easy to implement classifier. One can start the analysis using it. However, we will learn more sophisticated ways of doing this.

Support Vector Machines

Assume we are given the training data $\{\mathbf{x}_i, y_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$. For example, for $d = 2$:

For a new point (without label), the optimal linear classifier is:

$$\widehat{y}(\mathbf{x}^\star) = sign(\mathbf{x}^\star \boldsymbol{\theta} + b)$$

and it maximises the **margin**.

Using the geometry of the problem, we can cast the SVM problem as a quadratic optimization task (just as in Lasso):

$$\max_{C \geq \alpha \geq 0} \left[-\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i y_i \mathbf{x}_i \mathbf{x}_j^T y_j \alpha_j + \sum_{i=1}^{N} \alpha_i \right]$$

or, equivalently, in matrix notation:

$$\min_{C \geq \alpha \geq 0} \frac{1}{2} \boldsymbol{\alpha}^T diag(y) \mathbf{X} \mathbf{X}^T \mathrm{diag}\,(y) \boldsymbol{\alpha} - \mathbf{1}^T \boldsymbol{\alpha}$$

where C is a very large upper-bound, $\boldsymbol{\alpha} \in \mathbb{R}^n$ are the unknown coefficients, $\mathbf{1}$ is a vector of ones, and $diag(y) \in \mathbb{R}^{n \times n}$ is a diagonal matrix with $\mathbf{y}$ on the diagonal.

As in Lasso, most of the weights α are equal to zero. In fact, the Karush-Kuhn-Tucker Theorem tells us that only the α_i corresponding to points in the boundaries can be different from 0. That is, only the N_s points in the boundaries are needed for specifying the optimal classifier. These N_s points are called the **support vectors**.

When we get a new data point $\mathbf{x}^\star$, we generate class predictions as follows:

$$\widehat{y} = sign\left[\sum_{i=1}^{N_s} \alpha_i y_i \mathbf{x}^\star \mathbf{x}_i^T + b\right]$$

To compute b, we proceed as follows. First, we compute it for each training data point using the estimate of $\boldsymbol{\alpha}$:

$$b_i = y_i - \sum_{j=1}^{N_s} \alpha_j y_j \mathbf{x}_i \mathbf{x}_j^T$$

Next, we average

$$b = \frac{1}{N_s} \sum_{i=1}^{N_s} b_i$$

or take the median. Note that we only need to store the N_s support vectors and not the entire data set.

Nonlinear SVMs

Note that x allways appears in the form of a dot product $x_i' x_j$. This enables us to use the kernel tricks again to transform nonlinear classification in low dimensions to linear classification in high dimensions.

In the nonlinear setting, the optimization problem is:

$$\min_{C \geq \alpha \geq 0} \frac{1}{2} \boldsymbol{\alpha}^T diag(y) \mathbf{K} \operatorname{diag}(y) \boldsymbol{\alpha} - \mathbf{1}^T \boldsymbol{\alpha}$$

and the classifications are given by:

$$\widehat{y} = sign\left[\sum_{i=1}^{N_s} \alpha_i y_i k(\mathbf{x}^\star, \mathbf{x}_i) + b\right]$$

The full derivation of SVMs appears in:

(1) C. J. C. Burges., "*A Tutorial on Support Vector Machines for Pattern Recognition.*", Knowledge Discovery and Data Mining, 2(2), 1998.

(2) A. J. Smola and B. Schőlkopf., "*A Tutorial on Support Vector Regression.*", NeuroCOLT Technical Report NC-TR-98-030, Royal Holloway College, University of London, UK, 1998.