IEOR 165 – Lecture 12
Support Vector Machines

---

1 Classification

1.1 Motivating Example

As a motivating example, suppose we have measurements from people of different health markers (e.g., blood pressure, resting heart rate, weight) and would like to predict whether or not the person smokes. In order to construct such a model, we need to use data from some set of individuals. Here, we can aggregate the different health markers for the i-th person into a vector $x_i \in \mathbb{R}^p$. Similarly, we can assign a *label* to the i-th person. If this person smokes, we label them with $y_i = -1$; and if this person does not smoke, then we label them with $y_i = 1$.

1.2 Abstract Setting

In principle, this is a regression problem where the inputs are x_i and the outputs are y_i. However, there is additional structure in this specific class of regression problems. This structure is that the outputs take on a small discrete set of values. And the most basic situation is when the outputs are $y_i \in \{-1, 1\}$. Because these outputs are representative of labels, the numeric value is not important as long as the labels are self-consistent. For instance, we could specify $y_i \in \{0, 1\}$ or $y_i \in \{\mathsf{A}, \mathsf{B}\}$. However, it is mathematically convenient to use the labels $y_i \in \{-1, 1\}$.

The problem of *classification* consists of the following setup: We are given a set of measurements $x_i \in \mathbb{R}^p$ from the i-th object, along with a corresponding $y_i \in \{-1, 1\}$ that gives a label on the object. And we would like to construct a model that takes x_i as inputs and generates y_i as outputs. The more general case when the labels are $y_i \in \{0, 1, 2, \ldots, C\}$ is known as *multiclass* classification.

2 Linear Support Vector Machine

Consider the following nonlinear model:

$$y_i = \mathsf{sign}(x_i'\beta + \beta_0) + \epsilon_i,$$

where $y_i \in \{-1, 1\}$, $x_i, \beta \in \mathbb{R}^p$, $\beta_0 \in \mathbb{R}$, and ϵ_i is noise. We can think of the y_i as labels of each x_i, and the ϵ_i noise represents (potential) mislabeling of each x_i. The intuitive picture is that $x_i'\beta + \beta_0 = 0$ defines a hyperplane that separates $\mathbb{R}^p$ into two parts, in which points on one side of the hyperplane are labeled 1 while points on the other side are -1. Also note that there is a normalization question because $x_i'(\gamma \cdot \beta) + \beta_0 = 0$ defines the same hyperplane for any $\gamma > 0$.

The key to identifying models is to observe that the boundaries of the half-spaces

$$x_i'\beta + \beta_0 \leq -1$$
$$x_i'\beta + \beta_0 \geq 1$$

are parallel to the separating hyperplane $x_i'\beta + \beta_0 = 0$, and the distance between these two half-spaces is $2/\|\beta\|$. So by appropriately choosing β, we can make their boundaries arbitrarily close (or far) to $x_i'\beta + \beta_0 = 0$. Observe that for noiseless data, we would have

$$y_i(x_i'\beta + \beta_0) \geq 1.$$

So we could define our estimate by the coefficients that maximize the distance between the half-spaces (which is equivalent to minimizing $\|\hat{\beta}\|$ since the distance is $2/\|\hat{\beta}\|$). This would be given by

$$\begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta} \end{bmatrix} = \arg\min_{\beta_0,\beta} \|\beta\|^2$$
$$\text{s.t. } y_i(x_i'\beta + \beta_0) \geq 1, \forall i.$$

This optimization problem can be solved efficiently on a computer because it is a special case of quadratic programming (QP).

The above formulation assumes there is no noise. But if there is noise, then we could modify the optimization problem to

$$\begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta} \end{bmatrix} = \arg\min_{\beta_0,\beta} \|\beta\|^2 + \lambda \sum_{i=1}^{n} u_i$$
$$\text{s.t. } y_i(x_i'\beta + \beta_0) \geq 1 - u_i, \forall i$$
$$u_i \geq 0$$

The interpretation is that u_i captures errors in labels. If there are no errors, then $u_i = 0$, and if there is high error then u_i is high. The term $\lambda \sum_{i=1}^{n} u_i$ denotes that we would like to pick our parameters to tradeoff maximizing the distance between half-spaces with minimizing the amount of errors. There is another interpretation of

$$y_i(x_i'\beta + \beta_0) \geq 1 - u_i$$

as a soft constraint. This interpretation is that we would ideally like to satisfy the constraint $y_i(x_i'\beta + \beta_0) \geq 1$, but we cannot do so because of noise. And so the soft constraint interpretation is that u_i gives a measure of the constraint violation of $y_i(x_i'\beta + \beta_0) \geq 1$ required to make the optimization problem feasible.

3 Feature Selection

An important aspect of modeling is *feature selection*. The idea is that, in many cases, the relationship between the inputs x_i and the outputs y_i might be some complex nonlinear function

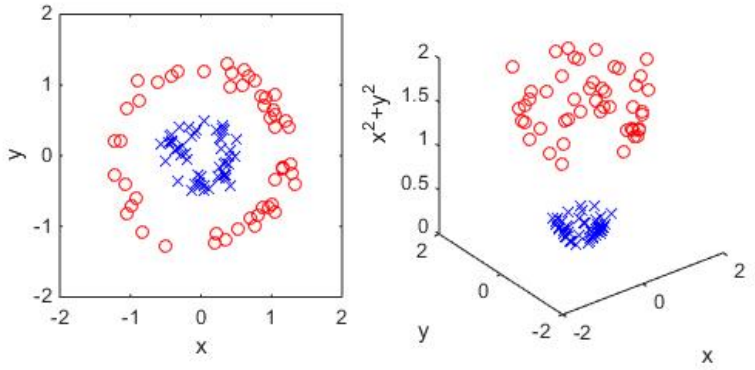
$$y_i = f(x_i) + \epsilon_i.$$

However, there may exist some nonlinear transformation of the inputs

$$z_i = g(x_i)$$

which makes the relationship between the inputs z_i and outputs y_i "simple". Choosing these nonlinear transformations requires use of either domain knowledge or trial-and-error engineering.

As an example, consider the following raw data on the left below. In this figure, the predictors are x, y, the "ex"-marks are points with label -1, and the "circle"-marks are points with label $+1$. A linear hyperplane cannot separate the two sets of points. However, suppose we add a new (third) predictor that is equal to $x^2 + y^2$. This is shown in the fighre on the right below. With this new predictor, we can now separate the data points using a linear hyperplane.



Feature selection is relevant to classification because it is often the case that the points corresponding to the classes cannot be separated by a hyperplane. However, it can be the case that a linear hyperplane can separate the data after a nonlinear transformation is applied to the input data. In machine learning, it is common to use the "kernel trick" to implicitly define these nonlinear transformations.

4 Kernel Trick

The first thing to observe is that using duality theory of convex programming, we can equivalently solve the formulation of the linear SVM by solving the following optimization problem:

$$\begin{aligned}\hat{\alpha} = \arg\min_{\alpha} \ & \textstyle\sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j x_i' x_j \\ \text{s.t.} \ & \sum_{i=1}^n \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq \lambda, \forall i\end{aligned}$$

This problem is also a QP and can be solved efficiently on a computer. And we can recover $\hat{\beta}_0, \hat{\beta}$ by computing

$$\begin{aligned}\hat{\beta} &= \sum_{i=1}^n \alpha_i y_i x_i \\ \hat{\beta}_0 &= y_k - \hat{\beta}' x_k, \text{for any } k \text{ s.t. } 0 < \alpha_k < \lambda.\end{aligned}$$

The key observation here is that we only need to know the inner product between two input vectors x_i and x_j in order to be able to compute the parameters of the SVM.

And so the kernel trick is that instead of computing a nonlinear transformation of the inputs $z_i = g(x_i)$ and then computing inner products

$$z_i' z_j = g(x_i)' g(x_j),$$

we will instead directly define the inner product

$$z_i' z_j = K(x_i, x_j)$$

where the $K(\cdot, \cdot)$ is a Mercer kernel function. A Mercer kernel function $K(\cdot, \cdot)$ means this function is positive semidefinite. So the resulting optimization problem to compute a Kernel SVM is

$$\begin{aligned}\hat{\alpha} = \arg\min_{\alpha} \ & \textstyle\sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \cdot K(x_i, x_j) \\ \text{s.t.} \ & \sum_{i=1}^n \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq \lambda, \forall i\end{aligned}$$

This will still be a QP and efficiently solvable on a computer.

It is interesting to summarize some common kernel choices.

Kernel	Equation
Polynomial of Degree d	$K(x_i, x_j) = (x_i' x_j)^d$
Polynomial of Degree Up To d	$K(x_i, x_j) = (1 + x_i' x_j)^d$
Gaussian	$K(x_i, x_j) = \exp(-\|x_i - x_j\|^2 / 2\sigma^2)$

The choice of kernel depends on domain knowledge or trial-and-error.

The final question is how can we make predictions if we compute an SVM using the kernel trick? Because a new prediction is given by

$$\text{sign}(z'\hat{\beta} + \hat{\beta}_0)$$

where

$$\hat{\beta} = \sum_{i=1}^{n} \hat{\alpha}_i y_i z_i$$
$$\hat{\beta}_0 = y_k - \hat{\beta}' z_k, \text{for any } k \text{ s.t. } 0 < \hat{\alpha}_k < \lambda,$$

we can again use the kernel trick to write the predictions as

$$\text{sign}\left(\sum_{i=1}^{n} \hat{\alpha}_i K(x, x_i) + y_k - \sum_{i=1}^{n} \hat{\alpha}_i y_i K(x_k, x_i)\right), \text{for any } k \text{ s.t. } 0 < \hat{\alpha}_k < \lambda.$$