Lecture 23: Support vector machines

Reading: Chapter 9

STATS 202: Data mining and analysis

November 15, 2019

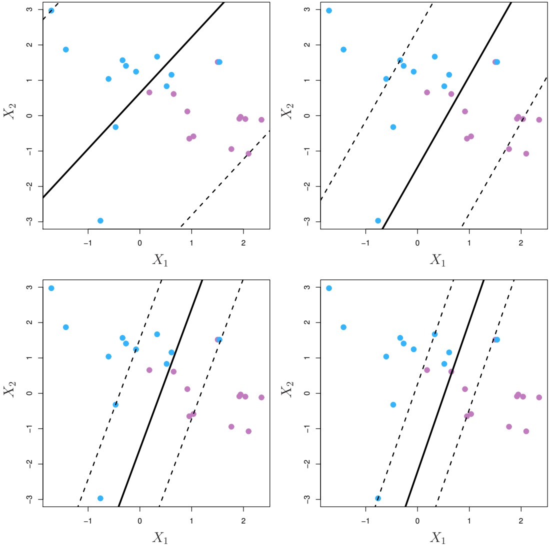
Review of support vector classifier

- The **support vector classifier** defines a hyperplane and two margins.
- **Goal:** to maximize the width of the margins, with some budget C for "violations of the margins", i.e.

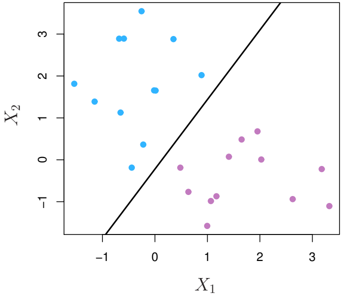
$$\sum_{\substack{x_i \text{ on the wrong} \\ \text{side of the margin}}} \text{Distance from } x_i \text{ to the margin} \;\leq\; C.$$

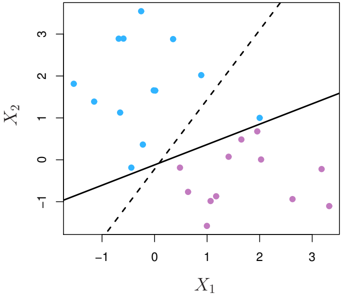
- The only points that affect the orientation of the hyperplane are those at the margin or on the wrong side of it.
- Low budget $C \iff$ Few samples used $\iff$ High variance $\iff$ Tendency to overfit.

Tuning the budget, C (high to low)



If the budget is too low, we tend to overfit





Maximal margin classifier, $C = 0$. Adding one observation dramatically changes the classifier.

Key fact about the support vector classifier

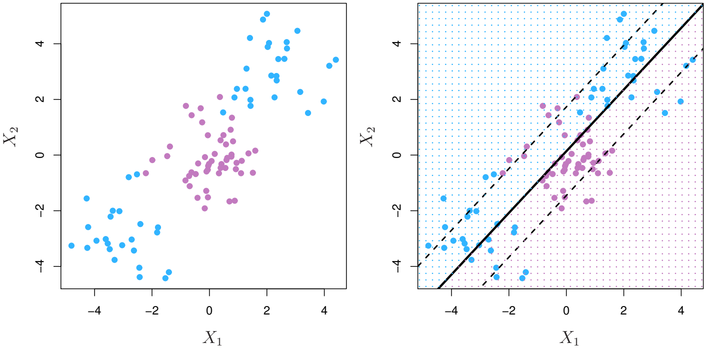
To **find the hyperplane** and **make predictions** the only information we need from our input vectors is the dot product between any pair of input vectors:

$$K(i,k) = (x_i \cdot x_k) = \langle x_i, x_k \rangle = \sum_{j=1}^{p} x_{ij} x_{kj}$$

We call this the **kernel matrix**.

How to deal with non-linear boundaries?

The support vector classifier can only produce a linear boundary.



How to deal with non-linear boundaries?

- In **logistic regression**, we dealt with this problem by adding transformations of the predictors.
- The original decision boundary is a line:

$$\log\left[\frac{P(Y=1|X)}{P(Y=0|X)}\right] = \beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0.$$

- With a quadratic predictor, we get a quadratic boundary:

$$\log\left[\frac{P(Y=1|X)}{P(Y=0|X)}\right] = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 = 0.$$

How to deal with non-linear boundaries?

- With a **support vector classifier** we can apply a similar trick.
- The original decision boundary is the hyperplane defined by:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0.$$

- If we expand the set of predictors to the 4D space (X_1, X_2, X_1^2, X_2^2), the decision boundary becomes:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 = 0.$$

- This is in fact a linear boundary in the augmented 4D space but a quadratic boundary in (X_1, X_2).

How do we expand the space of predictors?

- **Idea:** Add polynomial terms up to degree d:

  $$Z = (X_1, X_1^2, \ldots, X_1^d, X_2, X_2^2, \ldots, X_2^d, \ldots, X_p, X_p^2, \ldots, X_p^d).$$

- Does this make the computation more expensive?
- Recall that all we need to compute is the dot product:

  $$x_i \cdot x_k = \langle x_i, x_k \rangle = \sum_{j=1}^{p} x_{ij} x_{kj}.$$

- With the expanded set of predictors, we need:

  $$z_i \cdot z_k = \langle z_i, z_k \rangle = \sum_{j=1}^{p} \sum_{\ell=1}^{d} x_{ij}^{\ell} x_{kj}^{\ell}.$$

The kernel trick

Example. The polynomial kernel with $d = 2$:

$$K(i,k) = k(x_i, x_k) = (1 + \langle x_i, x_k \rangle)^2$$

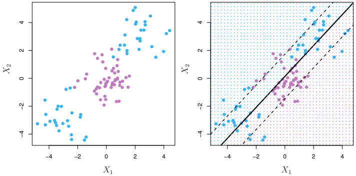
This is equivalent to the expansion:

$$\begin{aligned}\Phi(X) =&(1, \sqrt{2}X_1, \ldots, \sqrt{2}X_p,\\ &X_1^2, \ldots, X_p^2,\\ &\sqrt{2}X_1X_2, \sqrt{2}X_1X_3, \ldots, \sqrt{2}X_{p-1}X_p)\end{aligned}$$

- Computing $K(i,k)$ directly is $O(p)$.
- Computing the kernel using the expansion is $O(p^2)$.

How are kernels defined?

- Proving that a bilinear function $k(\cdot, \cdot)$ is positive definite (PD) is not always easy.
- However, we can easily define PD kernels by combining those we are familiar with:
  - Sums and products of PD kernels are PD.
- Intuitively, a kernel $k(x_i, x_k)$ defines a *similarity* between the samples x_i and x_k. This intuition can guide our choice in different problems.



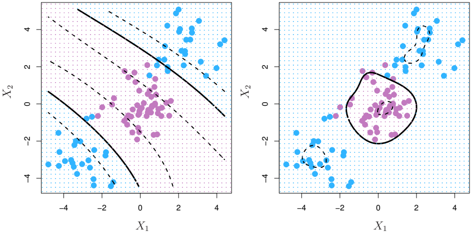
Common kernels

- The polynomial kernel:

$$k(x_i, x_k) = (1 + \langle x_i, x_k \rangle)^d$$

- The radial basis kernel:

$$k(x_i, x_k) = \exp\Big(- \gamma \underbrace{\sum_{j=1}^{p} (x_{ip} - x_{kp})^2}_{\text{Euclidean } d(x_i, x_k)} \Big)$$



What we know so far about support vector machines

- A support vector classifier is a generalization of the max margin classifier that allows the margin to be violated by an amount governed by a budget.
- A **support vector machine** is a support vector classifier applied on an expanded set of predictors, e.g.

$$\Phi : (X_1, X_2) \to (X_1, X_2, X_1X_2, X_1^2, X_2^2).$$

- One way to fit a SVM is to explicitly apply the feature map Φ to each input vector x_i and then run the support vector classifier on the augmented observations $(\Phi(x_i), y_i)$.
- However, we only need to know the dot products:

$$\langle \Phi(x_i), \Phi(x_k) \rangle \equiv K(i, k)$$

for every pair of samples (x_i, x_k).

The kernel trick

- Often, the dot product:

$$\langle \Phi(x_i), \Phi(x_k) \rangle \equiv K(i,k)$$

  is an easily computed function $k(x_i, x_k)$ of the original vectors, even if the mapping Φ significantly expands the space of predictors.

- **Example 1:** Polynomial kernel of degree 2

$$K(i,k) = (1 + \langle x_i, x_k \rangle)^2.$$

- With two predictors, this corresponds to the mapping:

$$\Phi : (X_1, X_2) \to (1, \sqrt{2}X_1, \sqrt{2}X_2, \sqrt{2}X_1X_2, X_1^2, X_2^2).$$

The kernel trick

- Often, the dot product:

$$\langle \Phi(x_i), \Phi(x_k) \rangle \equiv K(i,k)$$

  is an easily computed function $k(x_i, x_k)$ of the original vectors, even if the mapping Φ significantly expands the space of predictors.

- **Example 2:** RBF kernel

$$K(i,k) = \exp(-\gamma d(x_i, x_k)^2),$$

  where d is the Euclidean distance between x_i and x_k.

- In this case, the mapping Φ is an expansion into an infinite number of features! We can apply the method even if we don't what these transformations are.

The kernel trick

- If we don't know which transformations we are using, why would we expect the SVM to work?
  - The kernel $K(i, k) = k(x_i, x_k)$ measures the similarity between samples x_i and x_k.
  - We can evaluate whether k is a good measure of similarity without understanding the feature expansion Φ.
- Remember that not every similarity function is a valid kernel function. A valid kernel function will give rise to a positive semidefinite matrix K whenever it is applied to a collection of input vectors, where $K(i, k) = k(x_i, x_k)$.
- Fortunately, it is not too hard to show that many useful functions k are positive semi-definite.
- Conversely, if the matrix K is positive semi-definite, then there exists *some* mapping Φ to *some* feature space, such that $K(i, k) = \langle \Phi(x_i), \Phi(x_k) \rangle$.

Kernels for non-standard data types

- We can define families of kernels (with tuning parameters), which capture similarity between non-standard kinds of data:
    1. Text, strings (e.g. documents or tweets)
    2. Molecules (e.g. proteins or DNA)
    3. Images
    4. Graphs (e.g. social network graphs)
    5. Histograms
- Sometimes we know the mapping Φ, but there are algorithms that are fast for computing $K(i, k)$ without doing the expansion explicitly.
- Other times, the expansion Φ is infinite-dimensional or simply not known, but we can still compute the kernel function.

Example. Kernels for strings (HW 7)

Suppose we want to compare two strings in a finite alphabet:

$$x_1 = ACCTATGCCATA$$

$$x_2 = AGCTAAGCATAC$$

- Alphabet=$\{A, C, T, G\}$; define a "word" to be any sequence composed of letters from the alphabet.
- **Stringdot kernel:** For each word u of length p, we define a feature:

$$\Phi_u(x_i) = \# \text{ of times that } u \text{ appears in } x_i$$

- $\#$ features = (size of alphabet)^p
- $\#$ non-zero features $\leq L - p + 1$, where L is the length of x_i
- Can efficiently compute kernel function in $O(Lp)$ time

Example. Kernels for strings (HW 7)

Suppose we want to compare two strings in a finite alphabet:

$$x_1 = ACCTATGCCATA$$

$$x_2 = AGCTAAGCATAC$$

- **Gap weight kernel:** For each word u of length p, we define a feature:

$$\Phi_u(x_i) = \sum_{\substack{v=u; \\ v \text{ a subsequence of } x_i}} \lambda^{\text{\# of gaps in v}}$$

  with $0 < \lambda \leq 1$.

- The number of features can be huge! However, the kernel can be computed in $\mathcal{O}(L^2 p)$ time, where L is the length of x_i and x_k.

The kernel trick can be applied beyond SVMs

Kernel PCA:

- Suppose we want to do PCA with an expanded set of predictors, defined by the mapping Φ.
  - Can replace each x_i with $\Phi(x_i)$ in the data matrix
  - This allows us to find nonlinear curves that best explain data variability.
- To compute the PC scores of each datapoint, all we need to know is the kernel or Gram matrix:

$$K(i,k) = \langle \Phi(x_i), \Phi(x_k) \rangle.$$

- Even if Φ expands the predictors to a very high dimensional space, we can do PCA!
- The cost only depends on the number of observations n.

Kernel ridge regression, Kernel logistic regression, Kernel linear discriminant analysis, Kernel K-means, ...

Applying SVMs with more than 2 classes

- SVMs don't generalize nicely to the case of more than 2 classes.
- Two main approaches:
  1. **One vs. one:** Construct $\binom{K}{2}$ SVMs comparing every pair of classes. Apply all SVMs to a test observation and classify to the class that wins the most one-on-one challenges.
  2. **One vs. all:** For each class k, construct an SVM $\beta^{(k)}$ coding class k as 1 and all other classes as -1. Assign a test observation to the class k^*, such that the distance from x_i to the hyperplane defined by $\beta^{(k^*)}$ is largest (the distance is negative if the sample is misclassified).

Relationship to logistic regression

We can formulate the method for finding a support vector classifier

$$f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

as a Loss + Penalty optimization:

$$\min_{\beta_0, \beta} \sum_{i=1}^{n} \max[0, 1 - y_i f(x_i)] + \lambda \sum_{j=1}^{p} \beta_j^2.$$

For each sample (x_i, y_i) we incur a loss $\max[0, 1 - y_i f(x_i)]$ by using this classifier. In logistic regression, we minimize the loss

$$\min_{\beta_0, \beta} \sum_{i=1}^{n} \log[1 + e^{-y_i f(x_i)}]$$

(Could also add a penalty to logistic regression, which reduces overfitting and solves instability problems when classes are separable.)

Comparing the losses