Support Vector Machines & Neural Networks

Lu Haibo

Monday, Jun 15, 2015

Support Vector Machines (SVM)

SVM is an approach for classification that was developed in the computer science community in the 1990s.

Maximal Margin Classifier

- In general, if our data can be perfectly separated using a hyperplane, then there will in fact exist an infinite number of such hyperplanes.

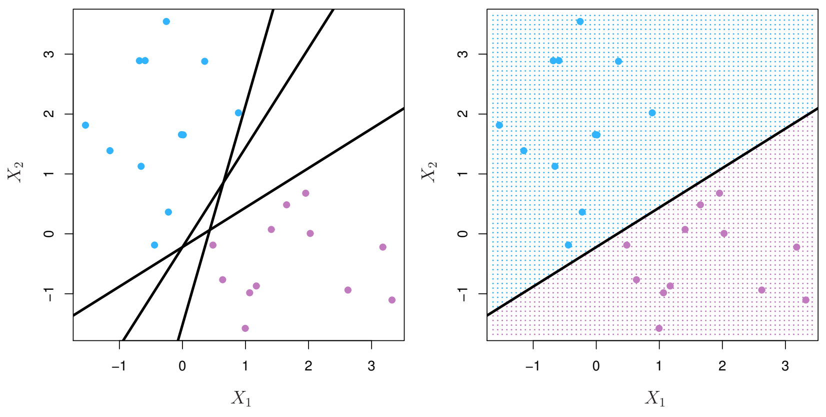

Figure 1: Left: There are two classes of observations, shown in blue and in purple, each of which has measurements on two variables. Three **separating hyperplanes**, out of many possible, are shown in black. Right: A separating hyperplane is shown in black. The blue and purple grid indicates the decision rule made by a classifier based on this separating hyperplane: a test observation that falls in the blue portion of the grid will be assigned to the blue class, and a test observation that falls into the purple portion of the grid will be assigned to the purple class.

- In order to construct a classifier based upon a separating hyperplane, we must have a reasonable way to decide which of the infinite possible separating hyperplanes to use.
- A natural choice is the *maximal margin hyperplane*, see Figure 2. [1]

[1] also known as the *optimal separating hyperplane*

- *Support Vector*: The three training observations that lie along the dashed lines are known as *support vectors*, since they "support" the maximal margin hyperplane in the sence that if these points were moved slightly then the maximal margin hyperplane would move as well.

Construction of the Maximal Margin Classifier

The training observations $x_1, \ldots, x_n \in \mathbf{R}^p$ and associated class labels $y_1, \ldots, y_n \in \{-1, 1\}$.

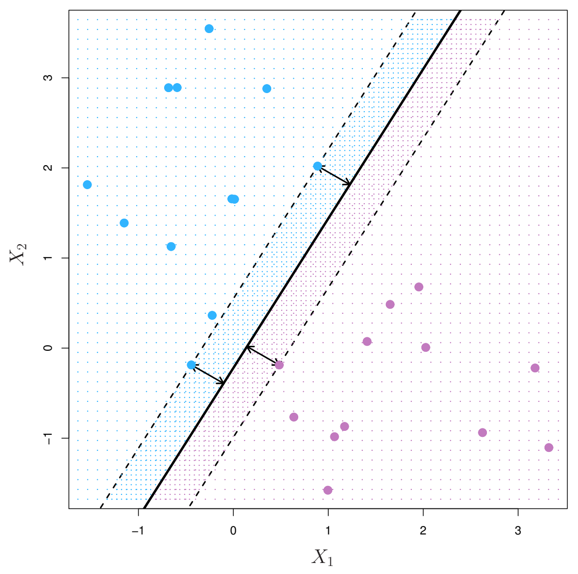

Figure 2: The maximal margin hyperplane is shown as a solid line. The margin is the distance from the solid line to either of the dashed lines. The two blue points and the purple point that lie on the dashed lines are the **support vectors**, and the distance from those points to the margin is indicated by arrows. The purple and blue grid indicates the decision rule made by a classifier based on this separating hyperplane.

The maximal margin hyperplane is the solution to the optimization problem

$$\max_{\beta_0,\beta_1,\dots,\beta_p} M$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M, \quad \forall i = 1,2,\ldots,n.$$

- M represents the margin of our hyperplane, and the optimization problem chooses $\beta_0, \beta_1, \ldots, \beta_p$ to maximize M.

The maximal margin classifier is

- a very natural way to perform classification, if a *separating hyperplane exists*. However, in many cases no separating hyperplane exists, and so **there is no maximal margin classifier**.
- extremely sensitive to a chage in a single observation suggests that it may have **overfit the training data**, see Figure 3.

Support Vector Classifiers

Consider a classifier based on a hyperplane that does **not** perfectly separate the two classes in the interest of

- Greater robustness to individual observations
- Better classification of *most* of the training observations

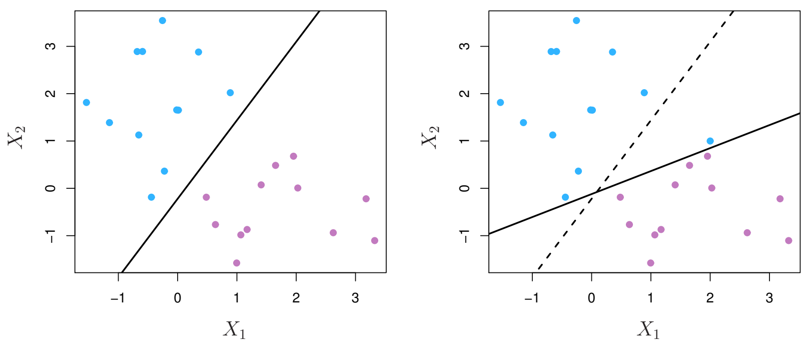

Figure 3: Left: Two classes of observations are shown in blue and in purple, along with the maximal margin hyperplane. Right: An additional blue observation has been added, leading to a dramatic shift in the maximal margin hyperplane shown as a solid line. The dashed line indicates the maximal margin hyperplane that was obtained in the absence of this additional point.

It is the solution to the optimization problem [2]

$$\begin{aligned}&\max_{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n} M\\ &\text{subject to} \sum_{j=1}^{p}\beta_j^2 = 1,\\ &y_i(\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\ldots+\beta_p x_{ip}) \geq M(1-\epsilon_i),\\ &\epsilon_i \geq 0, \sum_{i=1}^{n}\epsilon_i \leq C,\end{aligned} \tag{1}$$

[2] Observations that lie directly on the margin, or on the wrong side of the margin for their class, are known as support vectors. These observations do affect the support vector classifier.

where C is a nonnegative turning parameter. $\epsilon_1, \ldots, \epsilon_n$ are *slack variables* that allow individual observations to be on the wrong side of the margin or the hyperplane.

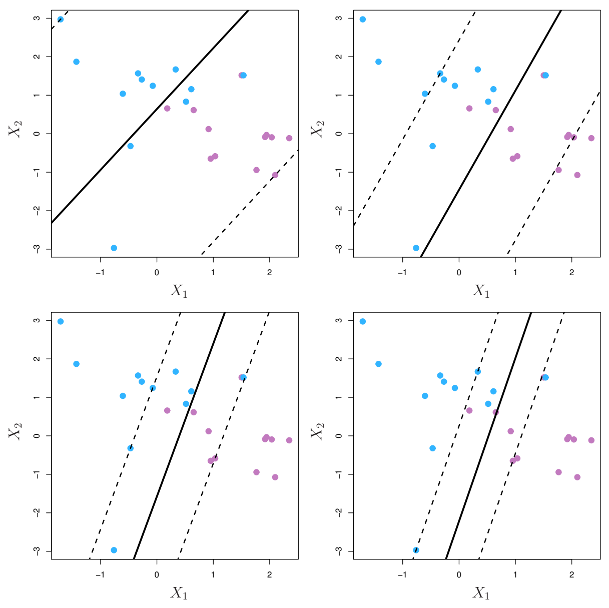

Figure 4: A support vector classifier was fit using four different values of the tuning parameter C. The largest value of C was used in the top left panel, and smaller values were used in the top right, bottom left, and bottom right panels. When C is large, then there is a high tolerance for observations being on the wrong side of the margin, and so the margin will be large. As C decreases, the tolerance for observations being on the wrong side of the margin decreases, and the margin narrows.

Classification with Non-linear Decision Boundaries

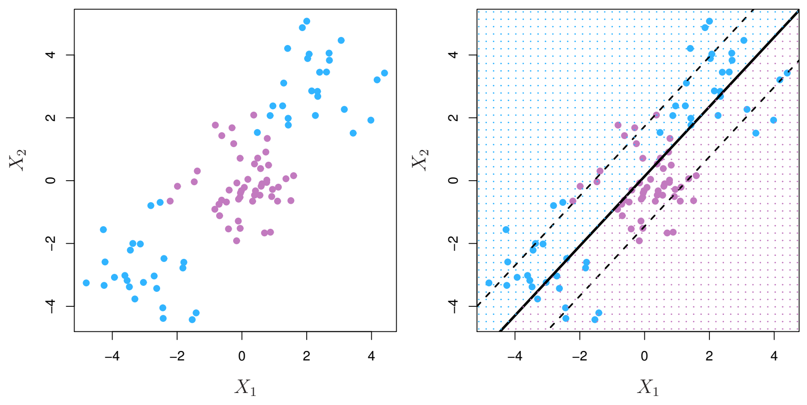

Figure 5: Left: The observations fall into two classes, with a non-linear boundary between them. Right: The support vector classifier seeks a linear boundary, and consequently performs very poorly.

Solution: **Enlarge the feature space** using quadratic, cubic, and even higher-order polynomial functions of the predictors.

For instance, rather than fitting a support vector classifier using p features $X_1, X_2, \ldots, X_p$, we could instead fit a support vector classifier using $2p$ features

$$X_1, X_1^2, X_2, X_2^2, \ldots, X_p, X_p^2.$$

The Support Vector Machine

The SVM is an extension of the support vector classifier that results from enlarging the feature space in a specific way, using *kernels*.

- The linear support vector classifier can be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i < x, x_i >,$$

  where there are n parameters α_i, $i = 1, \ldots, n$, one per training observation. $< a, b >= \sum_{i=1}^{r} a_i b_i$ is the *inner product* of two r-vectors a and b.

- SVM: We place the inner product with a *kernel*

$$K(x, x_i).$$

  In this case the (non-linear) function has the form

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

- A popular choice of K is the *radial kernel*:

$$K(a, b) = exp(-\gamma \sum_{j=1}^{p} (a_j - b_j)^2)$$

```
set.seed(1)
library(ISLR)
data(Carseats)
High <- ifelse(Carseats$Sales <= 8, "No", "Yes")
Carseats$High <- as.factor(High)
Carseats$Sales <- NULL
train <- sample(1:nrow(Carseats), nrow(Carseats) *
    0.75)

library(e1071)
fit <- svm(High ~ ., data = Carseats[train, ],
    kernel = "radial", gamma = 0.1, cost = 1)
pre.svm <- predict(fit, Carseats[-train, ])
table(pre.svm, Carseats$High[-train])
```

```
##
## pre.svm No Yes
##     No  50   7
##     Yes  9  34
```

```
# perform cross-validation using tune()
cv.out <- tune(svm, High ~ ., data = Carseats[train,
    ], kernel = "radial", ranges = list(cost = 10^(-5:2),
    gamma = c(0.1, 0.5, 1, 2, 3, 4)))
cv.out
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.1
##
## - best performance: 0.1633333
```

Neural Networks

- A class of learning methods that was developed separately in statistics and artificial intelligence.
- The central idea is to extract linear combinations of the inputs as derived features, and then model the target as a nonlinear function of these features.

Projection Pursuit Regression

Assume we have an input vector X with p components, and a target Y. Let ω_m, $m = 1, 2, \ldots, M$, be unit p-vectors of unknown parameters.

The projection pursuit regression (PPR) model has the form

$$f(X) = \sum_{m=1}^{M} g_m(\omega_m^T X).$$

- This is an additive model, but in the *derived features* $V_m = \omega_m^T X$ rather than the inputs themselves.
- The function g_m are unspecified and are estimated along with the directions ω_m using some flexible smoothing method.

However the projection pursuit regression model has not been widely used in the field of statistics, perhaps because at the time of its introduction (1981), its computational demands exceeded the capabilities of most readily available computers. But it does represent an important intellectual advance, one that has blossomed in its reincarnation in the field of neural networks.

Neural Networks

There has been a great deal of hype surrounding neural networks, making them seem magical and mysterious. As we make clear in this section, they are just nonlinear statistical models, much like the projection pursuit regression model discussed above.

- For regression, typically $K = 1$ and there is only one output unit Y_1 at the top.
- For K-class classification, there are K units at the top, with the kth unit modeling the probability of class k.

Derived features Z_m are created from linear combinations of the inputs, and then the target Y_k is modeled as a function of linear combinations of the Z_m:

$$\begin{aligned} Z_m &= \sigma(\alpha_{0m} + \alpha_m^T X), \ m = 1, \ldots, M, \\ T_k &= \beta_{0k} + \beta_k^T Z, \ k = 1, \ldots, K, \\ f_k(X) &= g_k(T), \ k = 1, \ldots, K, \end{aligned}$$

where $Z = (Z_1, Z_2, \ldots, Z_M)$, and $T = (T_1, T_2, \ldots, T_K)$.

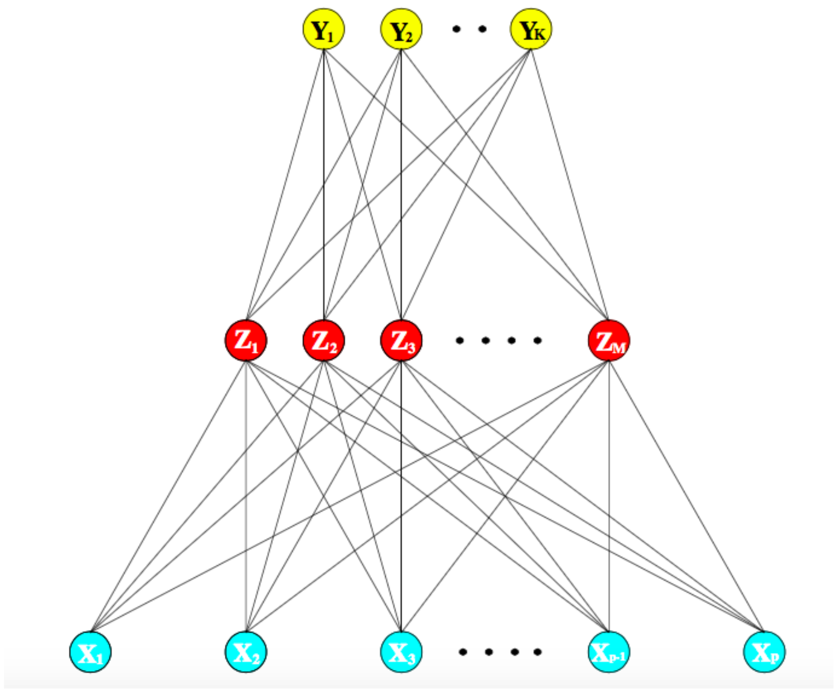

Figure 6: Schematic of a single hidden layer, feed-forward neural network.

- The *activation function* $\sigma(v)$ is usually choosen to be the *sigmoid*

$$\sigma(v) = 1/(1 + e^{-v})$$

- For regression, we typically choose the identity function $g_k(T) = T_k$
- For K-class classification, we choose the *softmax* function

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}$$

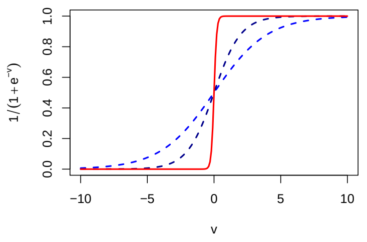

Figure 7: Plot of the sigmoid function $\sigma(sv) = 1/(1 + exp(-sv))$ commonly used in the hidden layer of a neural network. The scale parameter s controls the activation rate.

Notice also that the *neural network model with one hidden layer* has exactly the same form as the projection pursuit model described above. The difference is that the PPR model uses nonparametric functions $g_m(v)$, while the neural network uses a far simpler function based on $\sigma(v)$, with three free parameters in its argument.

Finally, we note that the name "neural networks" derives from the fact that they were first developed as models for the human brain. Each unit represents a neuron, and the connections represent synapses. In early models, the neurons fired when the total signal passed to that unit exceeded a certain threshold. In the model above, this corresponds to use of a step function for $\sigma(Z)$ and $g_m(T)$. Later the neural network was recognized as a useful tool for nonlinear statistical modeling, and for this purpose the step function is not smooth enough for optimization. Hence the step function was replaced by a smoother threshold function, the sigmoid.

```
library(nnet)
nn <- nnet(High ~ ., Carseats[train, ], size = 15,
    maxit = 200)
```

```
## # weights:  196
## initial  value 222.320050
## iter  10 value 193.046327
## iter  20 value 173.597581
## iter  30 value 146.475265
## iter  40 value 137.268556
## iter  50 value 134.722788
## iter  60 value 133.114499
## iter  70 value 131.612666
## iter  80 value 130.211534
## iter  90 value 129.051981
## iter 100 value 128.415814
## iter 110 value 128.170223
## iter 120 value 126.849998
## iter 130 value 120.899625
## iter 140 value 103.581878
## iter 150 value 83.648779
## iter 160 value 80.085325
## iter 170 value 79.283024
## iter 180 value 78.923323
## iter 190 value 78.423260
## iter 200 value 77.929887
## final  value 77.929887
## stopped after 200 iterations
```

```
pred.nn <- predict(nn, Carseats[-train, ])
pred.nn <- ifelse(pred.nn > 0.5, "Yes", "No")
table(pred.nn, Carseats$High[-train])
```

```
##
## pred.nn No Yes
##     No  55   5
##     Yes  4  36
```