When using soft-margin estimation for a support vector machine/classifier (SVM/SVC), we allow for some training observations to violate the margin, but we still want to minimize the number of violations as much as possible. We use slack variables to represent the amount of violation for each observation.

If we have a budget B for the total sum of slack variables, then we can express this constraint as:

$\sum \xi_i \leq B$

where ξ_i is the slack variable for the i-th training observation.

However, it is possible for more than B observations to have non-zero slack variables. This can happen if some observations are very difficult to classify correctly and require a large margin violation. In such cases, the SVM/SVC will still try to minimize the total sum of slack variables, subject to the budget constraint.

We can express this constraint by introducing additional slack variables δ_i that represent the difference between the amount of violation for the i-th observation and the budget:

$\xi_i - \delta_i \leq 0$

$\sum \delta_i \leq B$

Regenerate response

class labels $y_1, \ldots, y_n \in \{-1, 1\}$. Briefly, the maximal margin hyp
is the solution to the optimization problem

$$\begin{aligned} &\underset{\beta_0, \beta_1, \ldots, \beta_p, M}{\text{maximize}} \; M \\ &\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1, \\ &y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M \;\; \forall\, i = 1, \ldots, n \end{aligned}$$

and

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} < 0 \text{ if } y_i = -1.$$

Equivalently, a separating hyperplane has the property that

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) > 0$$

This optimization problem (9.9)–(9.11) is actually simpler than it
First of all, the constraint in (9.11) that

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

Sure, I can provide the equations that define the optimization problems for each classifier, as well as the role of the separating hyperplane and how the bias-variance trade-off is controlled.

A maximal margin classifier tries to find a hyperplane that separates the two classes with the maximum possible margin, subject to the constraint that all the training observations are correctly classified. The optimization problem can be formulated as:

minimize:

||w||

subject to:

$y_i(w^T x_i + b) >= 1$, for all i

where w is the vector of hyperplane coefficients, b is the bias term, x_i and y_i are the feature vector and class label for the i-th training observation, respectively.

The bias-variance trade-off in a maximal margin classifier is controlled by the regularization parameter C, which determines the trade-off between maximizing the margin and minimizing the norm of the hyperplane coefficients. A larger value of C will result in a smaller margin but a more complex hyperplane with lower bias and higher variance, while a smaller value of C will result in a larger margin but a simpler hyperplane with higher bias and lower variance.

A support vector classifier (SVC) tries to find a hyperplane that separates the two classes with some margin of error, by allowing for some observations to be misclassified or within the margin. The optimization problem can be formulated as:

minimize:

$||w||^2 + C(\sum_i \xi_i)$

subject to:

$y_i(w^T x_i + b) >= 1 - \xi_i$, for all i

$\xi_i >= 0$, for all i

where ξ_i are the slack variables that allow for misclassifications or observations within the margin.

Regenerate response

If δ_i is non-zero, it means that the i-th observation is violating the margin by more than what is allowed by the budget. By introducing these additional slack variables, we ensure that the total sum of slack variables is still bounded by B, even if more than B observations have non-zero slack variables.

In terms of the geometric interpretation of SVM/SVC, if an observation has a non-zero slack variable δ_i, it means that it is either on the wrong side of the hyperplane or it is within the margin but too close to the decision boundary. If an observation has a non-zero δ_i, it means that it is on the wrong side of the margin, beyond the allowed budget.

- Single linkage: 0.5
- Complete linkage: 1.0

The centroid of the final merged cluster would be at the coordinates (0.5, 0.5).

To calculate the dissimilarity values, we need to use the distance metric that is being used by the hierarchical clustering algorithm. In this case, we are using the Euclidean distance metric. The Euclidean distance between two points is calculated as follows:

Code snippet

```
d(x, y) = sqrt((x1 - y1)^2 + (x2 - y2)^2)
```

Use code with caution. Learn more

where x and y are the coordinates of the two points.

For single linkage, the dissimilarity between two clusters is the minimum distance between any two points in the two clusters. In this case, the minimum distance is 0.5, which is the distance between the points (0, 0) and (0.5, 0.5).

For complete linkage, the dissimilarity between two clusters is the maximum distance between any two points in the two clusters. In this case, the maximum distance is 1.0, which is the distance between the points (0, 0) and (1, 1).

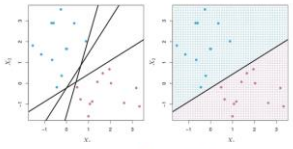
Maximal Margin Classifier

$$\boldsymbol{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots\ldots + \beta_p X_p = 0}$$

$$\boldsymbol{\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0}$$

Maximal Margin Classifier

Now the problem is that if we divide data based on hyperplane, then there could be many possible hyperplanes to divide same set of numbers. How do we decide the finite number of hyperplanes to choose from?



Therefore, maximal margin hyperplane is the hyperplane that has the largest margin, meaning, which has the largest distance between the hyperplane and the training observations. Using that hyperplane we can classify testing data. If our model has

$$\beta_0 + \beta_1 + \beta_2 \ldots \beta_p \text{as coefficients}$$

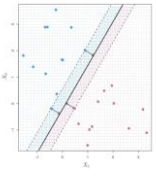
then the maximal margin classifier can classify new test observations based on the sign of

$$f(x^*) = \beta_0 + \beta_1 x^*_1 + \beta_2 x^*_2 \ldots \beta_p x^*_p$$

That is, for any testing observation *x(theta)* we can classify it based on the sign of the equation below. If the sign is negative, then class -1 is assigned and if the sign is positive the sign 1 is assigned.

$$\boldsymbol{f(x^*) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots\ldots + \beta_p X_p.}$$

Suppose if based on the training data, we can construct a hyperplane that can perfectly separate all training observations according to classes labeled. The classifier will work like this:



Considering the task of constructing the maximal margin hyperplane based on a set of n training observations $x1,\ldots.xn$ associated class "y" labeled as either -1 or 1, we need to consider three equations.

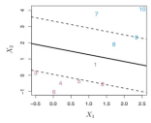
$$\boldsymbol{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots\ldots + \beta_p X_p < 0 \; if \; y = -1}$$

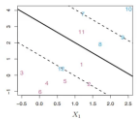
$$\boldsymbol{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots\ldots + \beta_p X_p > 0 \; if \; y = 1}$$

1. Maximizing M (margin) given $\beta_0 + \beta_1 + \beta_2 \ldots \beta_p$
2. $y_i\left(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots\ldots + \beta_p X_{ip}\right) \geq M$, *where* $i = 1 \ldots n$

$$\boldsymbol{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ \ldots\ldots + \beta_p X_p < 0}$$

$$\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ \ldots\ldots + \beta_p X_{ip}$$





tures $X_1, \ldots, X_4$ make up the units in the *input layer*. The arrows indicate that each of the inputs from the input layer feeds into each of the K *hidden units* (we get to pick K; here we chose 5). The neural network model has the form

$$\begin{aligned} f(X) &= \beta_0 + \textstyle\sum_{k=1}^{K} \beta_k h_k(X) \\ &= \beta_0 + \textstyle\sum_{k=1}^{K} \beta_k g(w_{k0} + \textstyle\sum_{j=1}^{p} w_{kj} X_j). \end{aligned} \tag{10.1}$$

It is built up here in two steps. First the K *activations* A_k, $k = 1, \ldots, K$, in the hidden layer are computed as functions of the input features $X_1, \ldots, X_p$:

$$A_k = h_k(X) = g(w_{k0} + \textstyle\sum_{j=1}^{p} w_{kj} X_j), \tag{10.2}$$

$$g(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

Details:

Adding this new condition the equations look like this:

1. Maximizing M (margin) given $\beta_0 + \beta_1 + \beta_2 \ldots \beta_p$
2. $y_i(\beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ \ldots\ldots + \beta_p X_{ip}) \geq M\,(1 - \epsilon_i)$, *where* $i = 1 \ldots . n$
3. $\epsilon_i \geq 0,\ \sum_i^n \epsilon_i \leq C$

Now comes the trade-off between bias and variance. C here is working as a tuning parameter that is chosen via cross-validation. When C is small, the model seeks narrow margins because it want to allow small number of observation to fall on wrong side, and hence the model become highly fit but with high variance. Similarly, when C is large it is more tolerant and allowing boundaries to be violated more often. This model has high biased but lower variance.

The bias-variance trade-off is controlled differently for each classifier. For a maximal margin classifier, the bias-variance trade-off is controlled by the choice of the regularization parameter, which determines the importance of the norm of the hyperplane coefficients in the optimization problem. A larger regularization parameter will result in a simpler hyperplane with lower variance but higher bias, while a smaller regularization parameter will result in a more complex hyperplane with higher variance but lower bias.

$$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise} \end{cases}$$

To train this network, since the response is qualitative, we look for coefficient estimates that minimize the negative multinomial log-likelihood

$$\begin{aligned} A_k^{(1)} &= h_k^{(1)}(X) \\ &= g(w_{k0}^{(1)} + \textstyle\sum_{j=1}^{p} w_{kj}^{(1)} X_j) \end{aligned} \qquad \begin{aligned} A_\ell^{(2)} &= h_\ell^{(2)}(X) \\ &= g(w_{\ell 0}^{(2)} + \textstyle\sum_{k=1}^{K_1} w_{\ell k}^{(2)} A_k^{(1)}) \end{aligned} \qquad \begin{aligned} Z_m &= \beta_{m0} + \textstyle\sum_{\ell=1}^{K_2} \beta_{m\ell} h_\ell^{(2)}(X) \\ &= \beta_{m0} + \textstyle\sum_{\ell=1}^{K_2} \beta_{m\ell} A_\ell^{(2)}, \end{aligned} \qquad f_m(X) = \Pr(Y = m|X) = \frac{e^{Z_m}}{\sum_{\ell=0}^{9} e^{Z_\ell}}$$

$$-\sum_{i=1}^{n} \sum_{m=0}^{9} y_{im} \log(\hat{f}_m(z_i)), \tag{10.14}$$

also known as the *cross-entropy*. This is a generalization of the criterion (4.5) for two-class logistic regression. Details on how to minimize this objective are given in Section 10.7. If the response were quantitative, we would instead minimize squared-error loss as in (10.9).

cross-entropy

The analyst would be able to reduce dimensionality down to 2 dimensions. The first two eigenvalues, 35 and 25, account for 60% of the variance, which is greater than or equal to 50%. The standard deviations of the data for these selected dimensions are 5.91 and 4.33, respectively.

The total variance of the data is 100. The first two eigenvalues, 35 and 25, account for 60% of the variance. This means that the first two principal components account for 60% of the variation in the data. The remaining three principal components account for 40% of the variation in the data.

The standard deviation of a principal component is the square root of its eigenvalue. The standard deviation of the first principal component is 5.91. The standard deviation of the second principal component is 4.33.

The analyst can use these two principal components to reduce the dimensionality of the data from 5 dimensions to 2 dimensions. This will allow them to visualize the data in a lower-dimensional space while still retaining most of the variation in the data.

To calculate the centroids of the two clusters, we simply take the mean of the observations each cluster:

centroid(C1) = (1 + 2 + 3 + 4) / 4 = 2.5
centroid(C2) = (-9 - 8 - 7 - 6) / 4 = -7.5

So the centroids of the two clusters are 2.5 and -7.5, respectively.

The k-means algorithm searches through a large parameter space to find a local optimum which means that it can converge to a suboptimal solution depending on the initial starting points of the centroids. The total number of clusterings that the algorithm would have to check to find a global optimum is exponential in the number of data points and the number of clusters. Specifically, the number of possible ways to assign N data points to K clusters is K^N. In this case, we have N = 8 data points and K = 2 clusters, so the total number of possible clusterings is 2^8 = 256. Therefore, the k-means algorithm would have to check 256 possible clusterings to guarantee that it has found the global optimum. However, in practice, the algorithm typically uses heuristics and random initialization to converge to a good local optimum much faster than checking all possible clusterings.

The centroid of cluster C1 is 2.5 and the centroid of cluster C2 is -7.5. The total number of clusterings we would have to check to find a global optimum is 2^4 = 16.

The centroid of a cluster is the point that is at the center of the cluster. It is calculated by taking the average of the coordinates of all of the points in the cluster. In this case, the centroid of cluster C1 is 2.5, because there are four points in the cluster with coordinates 1, 2, 3, and 4, and the average of these coordinates is 2.5. The centroid of cluster C2 is -7.5, because there are four points in the cluster with coordinates -9, -8, -7, and -6, and the average of these coordinates is -7.5.

The k-means algorithm searches through a large parameter space to obtain a local optimum. The parameter space is the set of all possible clusterings of the data. The number of possible clusterings is exponential in the number of data points. In this case, there are 4 data points, so there are 2^4 = 16 possible clusterings. The k-means algorithm will only find a global optimum if it checks all of these possible clusterings. However, in practice, the k-means algorithm will usually find a local optimum before it checks all of the possible clusterings.

feature space. The kernel trick uses inner product of two vectors. The inner product of two r-vectors a and b is defining as

$$\langle a,b \rangle \ \sum_{i=1}^{r} a^i b^i$$

Where a and b are nothing but two different observations.

Let's assume we have two vectors X and Z, both with 2-D data.

X= (x1,x2)

Z= (z1,z2)

Applying kernel trick of dot product will give us —

$$K(X,Z) = (X.Z)^2 = (x1z1 + x2z2)^2 = x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1 z_1 x_2 z_1$$