Session – Summary

Support Vector Machines

SVM was developed only in the late 1990s. And it was developed over some simpler models like the perceptron model and maximum margin classifiers etc.

You will start by learning to understand these models, coming to grips with the basic ideology of these kinds of models, before gradually moving towards SVM. In fact, we'll call models of this family **SVM-type models** because they are based on the same principles.

In the SVM type models, specifying the threshold probability is not a problem. We do not need to specify a threshold for the classification because the classification is based on a geometrically drawn boundary, not probabilities.

Concept of Hyperplanes

So in the data set, there were two features – length and weight of whales and dolphins. The task was to classify a new animal as a whale or a dolphin based on the features. The data contained the weights, lengths and the class of the animal. And in KNN, there was no model as such - the algorithm uses the entire data set to assign the class, without explicitly making a model. The task will remain the same, only the model changes.

Task	**Classify whale/dolphin based on weight and length**
Data	**Marine Data**
Features	**Length, Weight**
Model	**Perceptron - Hyperplane**

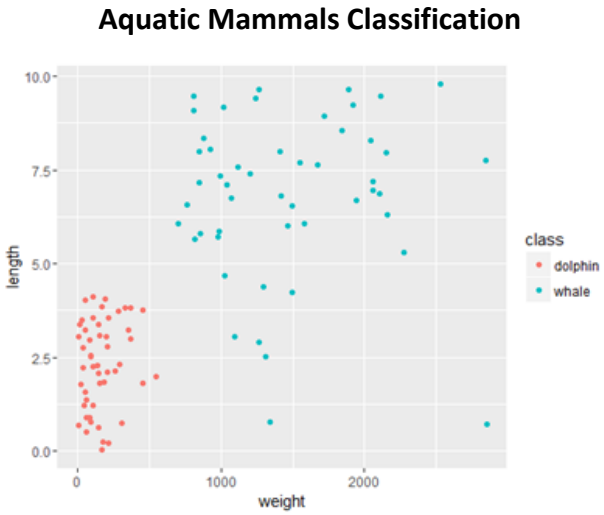

Figure 3.7.1: Scatter Plot showing various whale/dolphin data points

The hyperplane, in this case, is a plane separating the two classes. It can be seen below:

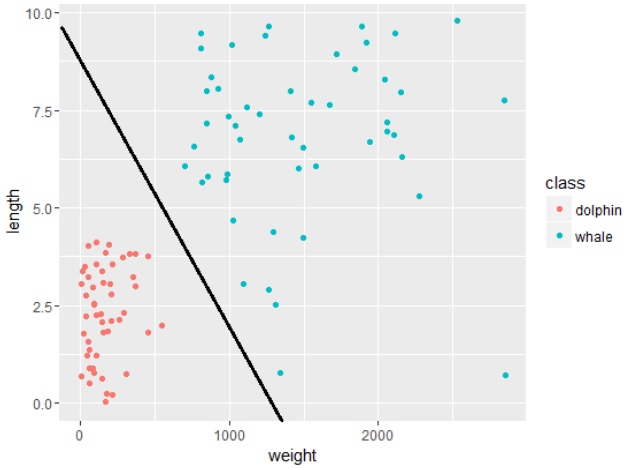

Figure 3.7.2: Line separating the two classes

This line separating the two classes is the model. This looks like an ideal line. Many such possible lines can separate the two lines and what we desire is a line that separates the two classes. The line can be found out by obtaining its equation.

In two dimensions, this straight line is called the hyperplane. In general, a hyperplane is a flat surface of dimensions one less than the original space. For example, this problem is an example of two-dimensional space because we have two features – weight and length, and so the hyperplane is one dimensional - a straight line.

In case we had three features like weight, length and colour, this would be a three-dimensional space, thus the hyperplane would be a two-dimensional plane. So in general, if you are working in an n-dimensional space, or have n features, the hyperplane will be a separating plane of n - 1 dimensions.

In two dimensions, the hyperplane is a straight line and its mathematical definition is pretty simple –

ay + bx + c = 0, where y is weight, x is length and c is a constant. It can also be written as

β0 + β1x1 + β2x2 = 0

where x1 and x2 are the features and beta0 is the constant. The reason we prefer to use the notation of beta1, beta2 etc. is that this can be easily extended to more than two dimensions. If you have three features, the hyperplane is simply **β0 + β1x1 + β2x2 + β3x3 = 0**, which is a two-dimensional plane. You can easily extend it to n-features.

So when we say the equation **β0 + β1x1 + β2x2 = 0** defines the hyperplane, it means that if any point x1, x2 lies on the line, where x1 = weight and x2 = length, then **β0 + β1x1 + β2x2 = 0**.

e.g.

Considering the following line whose equation is given by **0.0073x1 + x2 – 9.5 = 0.** Any point (x1,x2) lying on this line will satisfy any the equation and result in the equation being satisfied. Any point towards the upper or right(more towards the positive axes) will return a value greater than zero implying that it is a whale. Similarly, any point towards the left(more towards the negative axes) will return a negative value (less than zero), implying that it belongs to the dolphin class.

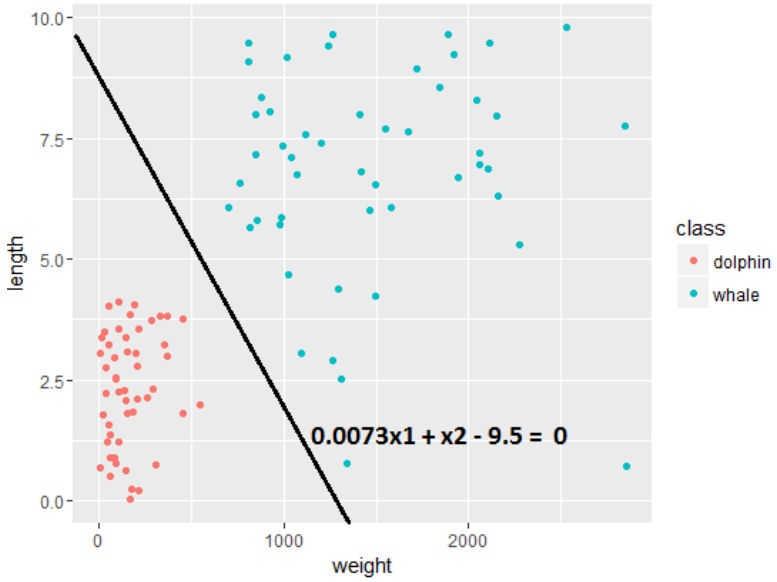

Figure 3.7.3: Equation of separating line

If you consider a point (2000,5) and plug in the values of x1 and x2 in the expression 0.0073x1 + x2 - 9.5, you'll get 0.0073(2000) + (5) - 9.5 = 10.1.

Now if you take another whale, say (1500 kg, 7 ft), the value of f is 0.0073(1500) + (7) - 9.5 = 8.45. As the point gets closer to the line so does the value get closer to zero.

Since f(x1,x2) > 0 for points to the right and f(x1,x2) < 0 for points to the left, respectively. Further, let us also assign numbers for the class labels, or the target variable Y as we always do. So let's say that

Y = 1 for whales and
Y= -1 for dolphins.

So if a data point is whale, y = 1 and f(x1, x2) > 0. If it's a dolphin, then y = -1 and f(x1,x2) < 0. In general, the product of y and f should be always positive. In other words, y.f(x1, x2) > 0 for all possible points.

Now, which line is the best? Consider the following figure where there is a hyperplane that is too close to the dolphin's class. When there is a new point coming to the classifier, by intuition it can be said that it belongs to the dolphin class. But since the line is too close to the dolphin's category, t

Figure 3.7.4: Line that incorrectly classifies unseen data

By intuition, it can be seen that the best line to classify would be the one that is midway between both the classes and has the highest margin.

Intuitively, the optimal hyperplane should have a 'safe distance' from the closest points on either side. In other words, the most appropriate hyperplane will have a good 'margin of safety' from the closest data points. The margin is shown in the figure by the two parallel lines about the hyperplane.

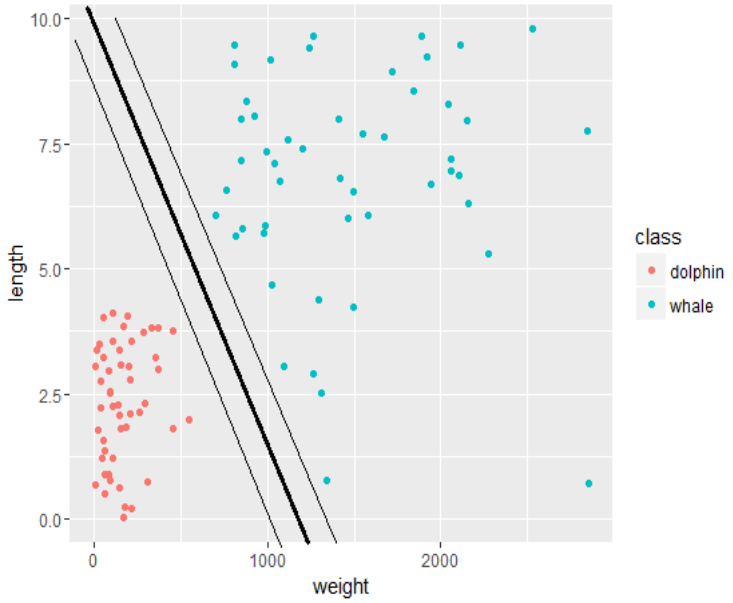

Figure 3.7.6: Maximum Margin Classifier

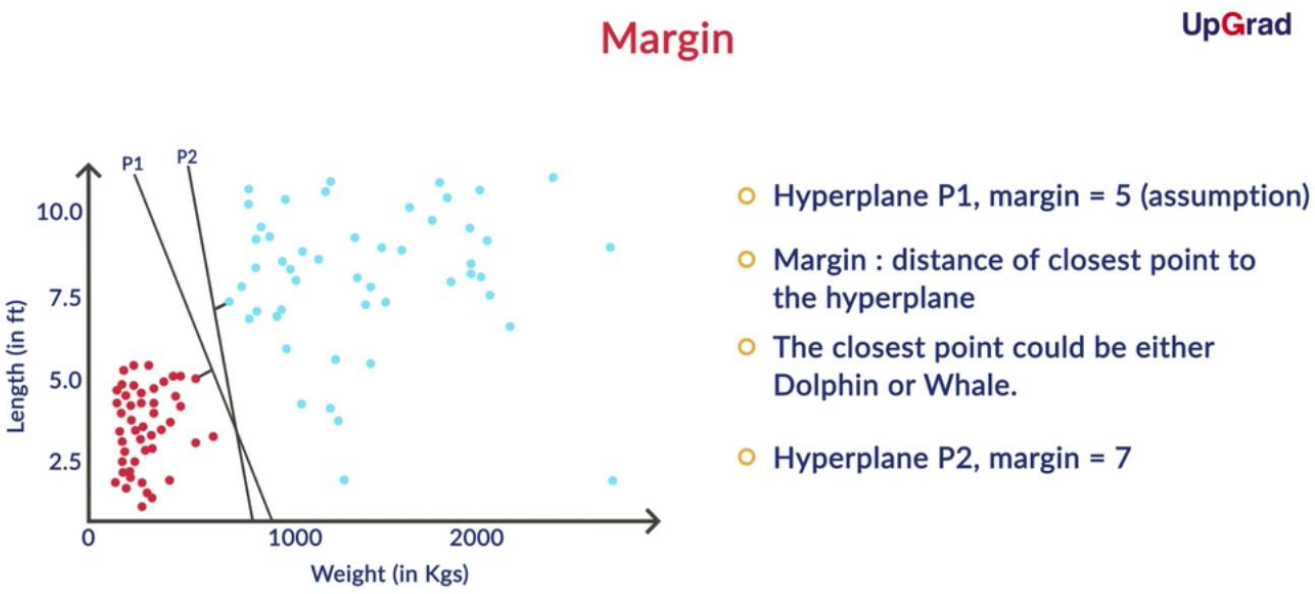

Figure 3.7.7: Line-Margin examples

Given a list of margins (distances of nearest points) from each line, the line that has the maximum margin needs to be selected as the optimal separating hyperplane

Hyperplane Margin

UpGrad

Hyperplane	Margin
P1	5
P2	7
P3	10
P4	12
P5	9
P6	3
P7	1.5

Figure 3.7.8: Hyperplane distances list

Now that the requirement is of the distance of all the points from the line to be greater than M, the new constraint changes to:

$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}) \geq M$, Where M is the margin. So the optimisation problem is to find **β_0, β_1** etc. such that the margin M is maximised and that

$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}) \geq M$.

The new constraint is that the product y.fi should be >= M and not just greater than 0.

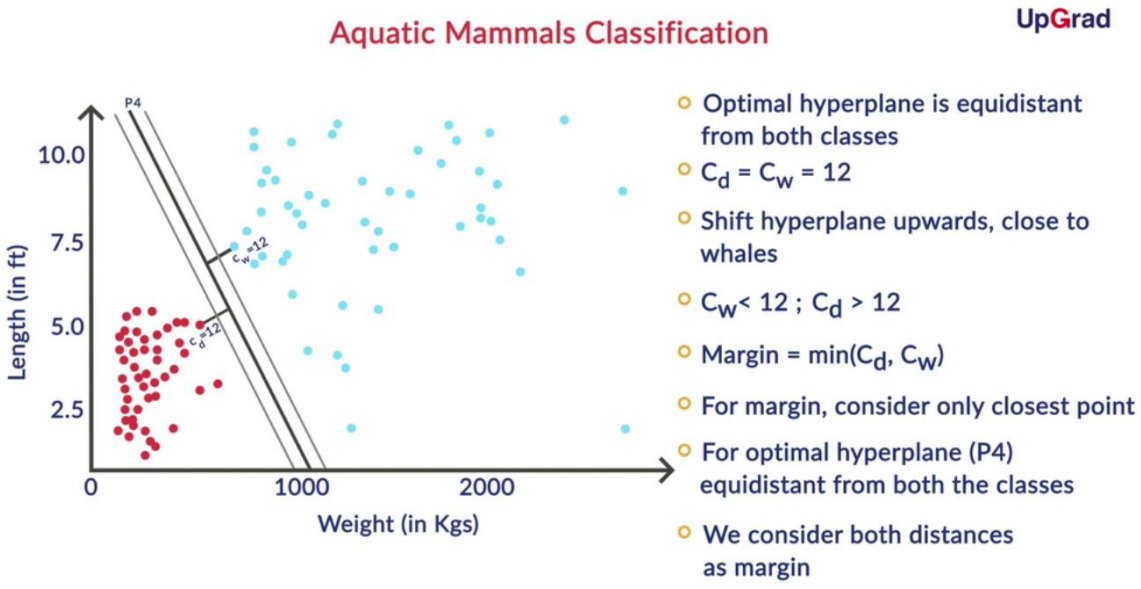

Figure 3.7.9: Properties of Maximum margin classifier

The new requirement is that the points should lie on the right side of the margin and not only on the right side of the hyperplane.

Drawbacks:

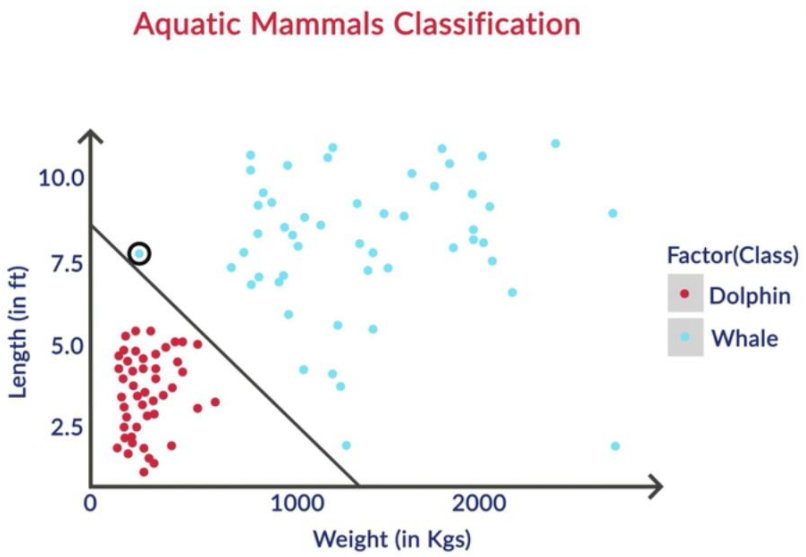

Figure 3.7.10: Highly biased towards avoiding misclassifications

It can be observed that it is trying to overfit the available data and tends to generalise on any future data. There needs to be a way to allow for some misclassifications to help the classifier in general to classify unseen data.

Support Vector Classifier

By building a hyperplane which has the flexibility to misclassify certain points, we can achieve two objectives:

1. Less sensitivity to individual observations, and
2. Better classification of most of the training observations

Thus, it is desired that our hyperplane or model allows for certain misclassification for better model robustness. This is what the support vector classifier actually does.

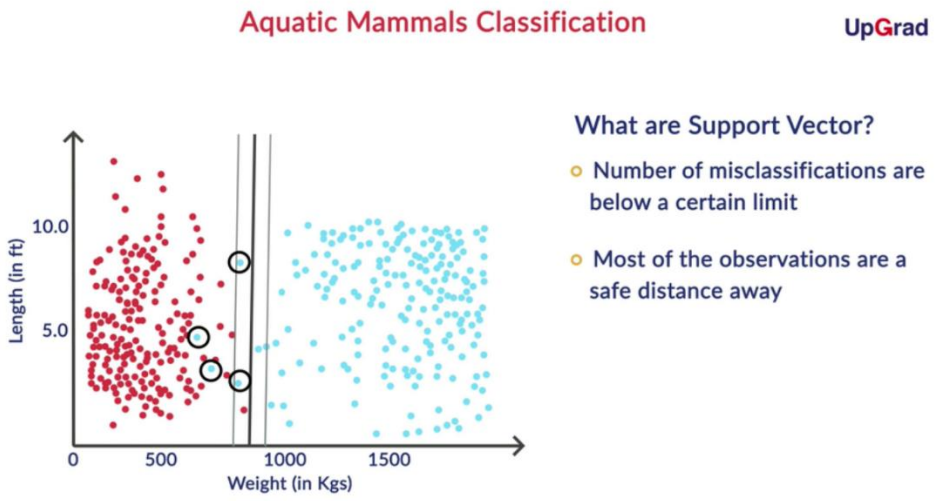

Figure 3.7.11: Optimisation Problem

The hyperplane misclassifies four whales shown in circles. The so-called 'soft-margin' is shown by the two lines along the hyperplane. It is called soft-margin because it is allowed to misclassify certain points.

Obviously, the requirement is for a broader margin and the misclassifications to be minimized - both at the same time. So now we have to balance both the things. This is done by introducing slack variables that allow for some misclassifications on the training data in order to generalise well on unseen data.

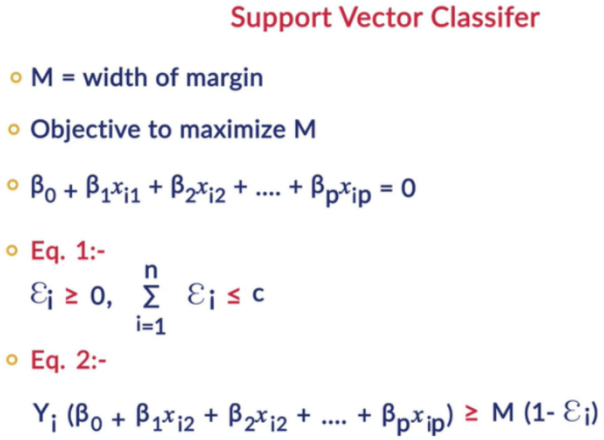

Figure 3.7.12: New classifier constraints

The support vector classifier essentially allows for certain points to be (deliberately) misclassified. By doing this, the classifier is able to classify most of the points correctly and is also more robust. The equations are given as:

$\epsilon_i \geq 0, \sum \epsilon_i \leq c$, where i = 0,1,2 .... n.

$$Y(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3) \geq M(1 - \epsilon_i)$$

The first equation shows that the value of the slack variable is always greater than zero and given a tuning parameter c, the sum of all the values of the slack variable shouldn't exceed the value of c. The second equation is the modified version of the maximum margin classifier with the addition of the slack variable to allow for some misclassifications in order to avoid overfitting the data.

In support vector classifier, we still desire the margin to be as large as possible. The only difference is we have to account for the number of points allowed for misclassification.

The margin was earlier defined by the hard constraint of

$$Y(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3) \geq M$$

We allow for some slackness in the constraints so as to reduce the effect of outliers on the margin. This slack variable is ϵ (also represented as ζ). The value of the slack variable is always greater than zero, and it lies between 0 and 1.

$$\epsilon_i = 0$$

This condition is true for all points on the correct side of the hyperplane and outside the margin. All correctly classified points have $\epsilon_i = 0$

$$0 < \epsilon_i \leq 1$$

This condition is true for those data points that lie somewhere between the margin and the correct side of the hyperplane.

$$\epsilon_i > 1$$

This condition is true for misclassified point lying on the wrong side of the hyperplane.

Summary:

1. In support vector classifier, the margin is called soft-margin because some points can lie inside it and may even be misclassified. If a point is correctly classified but lies inside the margin, we say that it has violated the margin.
2. Then we defined the slack variable epsilon which tells us the location of the point - epsilon can be 0, > 0 or > 1. Epsilon = 0 means the point falls on the correct side of the margin, and > 1 means that it is misclassified.

3. The tuning parameter C helps us control the number of misclassified points and is akin to the budget for the number of violations.

Support Vectors & Cost

It turns out that only the observations that either lie on the margin or that violate the margin will affect the hyperplane. The points which lie far away from the plane are not even considered. Such observations, which lie on or inside the margin, are known as Support vectors, because in a way they support the classifier.

When the tuning parameter C is large, the budget for violations is huge. So the margin will be wide, and many of them will violate the margin. In short, there will be many support vectors. The support vector classifier is allowed to misclassify certain observations. The cost represents the penalty of misclassifying observations. Therefore, if you assign a very high cost of misclassification, you want to be careful to avoid misclassifying points. On the other hand, if you choose a low cost, you are willing to increase the misclassified points because it will give you a higher margin and robustness.

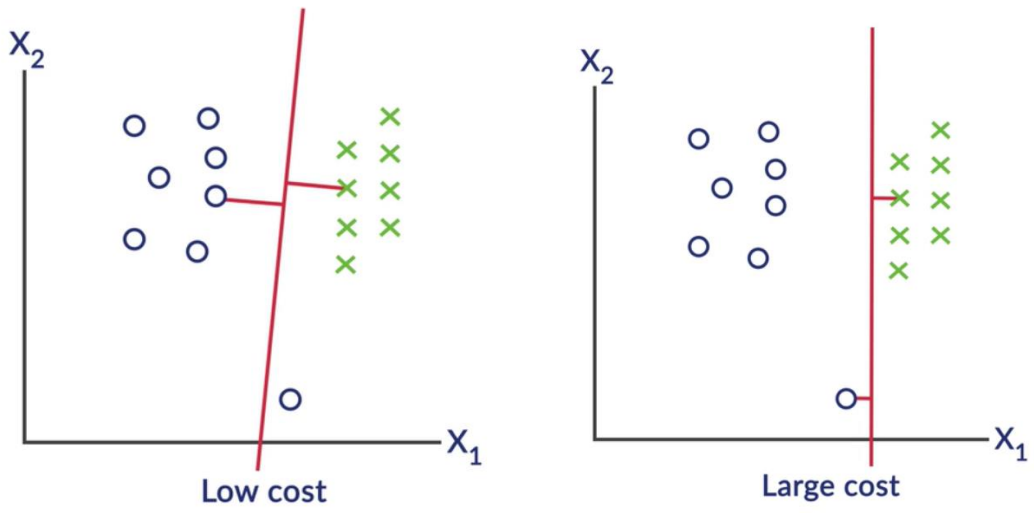

Figure 3.7.13: Low cost doesn't overfit the data and has a wide margin. In contrast, as the cost gets higher, it leads to a narrower margin and tries to avoid misclassifying.

Cost and the tuning parameter c quantify the misclassifications. The cost, in general, is the quantity for misclassifications, i.e., more misclassifications lead to a higher cost, and hence we need to minimise it. On the other hand, the tuning parameter c is the budget for misclassifications. A higher value of c means you want fewer misclassifications, and hence it will try and overfit the training data. A higher value of c also implies that the margin will be narrower. Hence a small value of c will lead to more budget for misclassifications and hence, a higher cost. This is why they are inversely proportional.

The svm() function of the e1071 library can be used for SVM models. The syntax is somewhat similar to earlier models where we specify the dependent and independent variables. The other arguments include the data to use, the type of kernel which can be linear, rbf, polynomial or sigmoid, the cost which in this case is the tuning parameter and scale as true or false. The list of all arguments can be found in the e1071 package document.

```
22  # model 0 with cost = 0.1
23  model.svm.0 = svm(class ~., data = train.data, kernel = "linear", cost = 0.1, scale = F)
24  plot(model.svm.0, train.data)
25  summary(model.svm.0)
26
27
28  # model 1 with cost = 100
29  model.svm.1 = svm(class ~., data = train.data, kernel = "linear", cost = 100, scale = F)
30  plot(model.svm.1, train.data)
31  summary(model.svm.1)
32
33  # model 2 with cost = 10000
34  model.svm.2 = svm(class ~., data = train.data, kernel = "linear", cost = 10000, scale = F)
35  plot(model.svm.2, train.data)
36  summary(model.svm.2)
```

Figure 3.7.14: SVM function

The svm function of the e1071 package has a parameter called "scale". What does it really do?

The main advantage of scaling is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation. Scaling should be used when there are values over various ranges, which could cause the smaller ranged values to have less effect on building a classifier.

Many times it is desirable to explicitly allow some training points to be misclassified in order to have an "overall better" position of the separating hyperplane. Mathematically, "better" translates to "optimising cost function valuing mistakes with certain coefficient". Intuitively, "better" implies "wider cushion, a few mistakes allowed". Practically "better" is to be understood as "performs well on real data".

Hence the value of the cost argument (which essentially is the tuning parameter c in the svm function) doesn't need to be so high that it overfits the training data and neither too low to make it highly biased. Rather, it needs to be at a certain level such that it is able to successfully generalise unseen data. Tuning is a good way to find the appropriate cost function.

```
# finding the optimal value of cost using cross-validation
# svm cross-validation is done using the tune function
# result: higher costs yield lower error rates

tune.svm = tune(svm, class ~., data = train.data, kernel = "linear",
                ranges = list(cost = c(0.001, 0.01, 0.1, 0.5, 1, 10, 100)))
summary(tune.svm)

# the tune function also stores the best model obtained
# cost = 0.5 is the best model

best.mod = tune.svm$best.model
best.mod

```

Figure 3.7.15: Tuning and cross validation

The tune function uses cross validation to check for various values of the cost or any other list of values for parameters. The best model can be stored using best.model and can be used to predict on test data.

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.5

- best performance: 0.01071429
```

Figure 3.7.16: Summary of tuning

The sampling method is mentioned in the summary along with the best parameter of cost which has the minimum error (which in this case is the cost with a value of 0.5).

```
- Detailed performance results:
   cost      error dispersion
1 1e-03 0.50000000 0.18365772
2 1e-02 0.02500000 0.03388155
3 1e-01 0.01428571 0.02497164
4 5e-01 0.01071429 0.01725164
5 1e+00 0.01071429 0.01725164
6 1e+01 0.01071429 0.01725164
7 1e+02 0.01428571 0.01844278
```

Figure 3.7.17: Detailed Performance results

This gives a detailed overview of the error and dispersion for each value of cost. 1e-03 means 0.001, 5e-01 is 0.5 and so on. The best performance is for a model that has the least error and dispersion.

```
56  # predicting test classes using the best model and analyzing the table
57  # best.model is the one with cost = 0.5
58
59  ypred = predict(best.mod, test.data)
60  table(predicted = ypred, truth = test.data$class)
61  confusionMatrix(ypred, test.data$class)
62
63  plot(best.mod, test.data)
64
```

Figure 3.7.18 Confusion Matrix

The table function can be used to get the confusion matrix to get the True Positives & Negatives and the False Positives & Negatives. Using the confusionMatrix function further displays the Sensitivity, Specificity, Accuracy and other parameters to help determine the model performance.

```
         truth
predicted dolphin whale
  dolphin      62     2
  whale         0    56
```

Figure 3.7.19: Truth Table

The diagonals represent the true positive and true negatives. The above table shows that two whales have been wrongly classified as dolphins whereas 62 dolphins and 56 whales have been classified correctly.

```
Confusion Matrix and Statistics

          Reference
Prediction dolphin whale
   dolphin      62     2
   whale         0    56

               Accuracy : 0.9833
                 95% CI : (0.9411, 0.998)
    No Information Rate : 0.5167
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9666
 Mcnemar's Test P-Value : 0.4795

            Sensitivity : 1.0000
            Specificity : 0.9655
         Pos Pred Value : 0.9688
         Neg Pred Value : 1.0000
             Prevalence : 0.5167
         Detection Rate : 0.5167
   Detection Prevalence : 0.5333
      Balanced Accuracy : 0.9828

       'Positive' Class : dolphin
```

Figure 3.2.20: Detailed Confusion Matrix and Statistics

The accuracy is about 98% and the value of sensitivity and specificity are high as well. The positive class is the dolphin class.