Lab: AI & ML Lab (23A31403)

College: Andhra Engineering College

13. Apply Support Vector algorithm for classification.

Math

8.4 SUPPORT VECTOR MACHINES

We have seen in Fig. 8.4 that there could be theoretically infinite decision boundaries if the classes are linearly separable. Perceptron selects one of them. However, if we want to select a decision boundary that is evenly centred between the two class boundaries, then support vector machine (SVM) is the best choice to obtain the optimal solution. Consider Fig. 8.6.

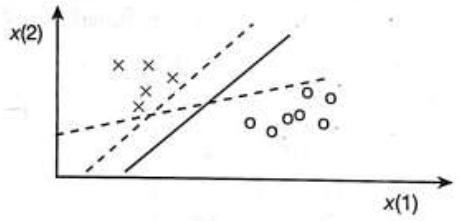

FIG. 8.6 Bad and good decision boundaries

There are two classes in a two-dimensional space, specified by $x(1)$ and $x(2)$. There are 5 Xs from the negative class and 7 circles from the positive class. These two classes are linearly separable.

There are two decision boundaries indicated by broken lines. Both of them are bad; one of them is very close to the negative class and the other one is unable to exploit the margin of separation between the two classes. The third decision boundary shown by a solid line appears to be good as it evenly splits the margin of separation between the two classes.

The SVM classifier formulates an optimization problem to maximize the margin between the two classes; the decision boundary will be evenly positioned between the two classes. Some of its important features are as follows:

- It considers a two-class problem with class label $y_i \in \{-1, +1\}$ for x_i in the training set for $i = 1, 2, \ldots, n$; if $y_i = -1$, then x_i is from the negative class and if $y_i = +1$, then x_i is from the positive class.

8.4.1 Linearly Non-Separable Case

Consider the data shown in Fig. 8.8.

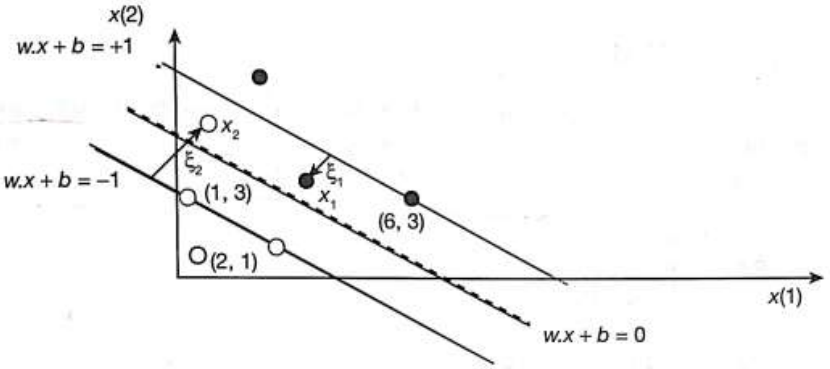

FIG. 8.8 An example to illustrate the margin violators (for colour figure, please see Colour Plate 2)

There are three points from the positive class; each of these is indicated by a blackened circle. There are four data points from the negative class; each of these is depicted using a whitened circle. There are two violators; they are labelled x_1 and x_2.

Data point x_1 is from the positive class and it violates the constraint given by $w^t x_1 + b \geq 1$ as it falls to the left of the positive support line. However, it is still on the correct side of the decision boundary. Similarly, data point x_2 is from the negative class and it violates the constraint $w^t x_2 + b \leq 1$. It clearly violates both the support line and the decision boundary.

The extent of violation is captured by values ξ_1 and ξ_2, respectively, for the two violators x_1 and x_2. Even though x_1 is a violator, it is still correctly classified as belonging to the positive class. However, x_2 is classified incorrectly as a member of the positive class. So, the violators need to be handled properly. Violations need to be tolerated to some extent.

This forms the basis for the so-called *soft-margin formulation* of the SVM given by

$$\textit{minimize } \frac{1}{2} w^t w + C \sum_{i=1}^{n} \xi_i$$

$$\textit{subject to } 1 - \xi_i - y_i(w^t x_i + b) \leq 0,\ i = 1, \ldots, n$$

8.4.3 Kernel Trick

We are transforming a vector x in the input space to a higher dimensional space ($\phi(x)$) and look for a possible linear separation in the new space that is called the feature space. The most important observation is that we do not need to map ϕ explicitly. In theory, ϕ could map points into an infinite dimensional space. What we need is the inner product in the feature space. If x_i and x_j are two data points in the input space, then our computations will need terms of the form $\phi(x_i)^t\phi(x_j)$.

Many similarity functions can be viewed as inner products. If vectors $\phi(x_i)$ and $\phi(x_j)$ are unit norm vectors, then $cos\theta = \phi(x_i)^t\phi(x_j)$, where θ is the angle between the two vectors $\phi(x_i)$ and $\phi(x_j)$. We use the **kernel function**, $K(x_i, x_j)$, that maps a pair of vectors $x_i \in \Re^l$ and $x_j \in \Re^l$ to a real number. Specifically,

$$K(x_i, x_j) = \phi(x_i)^t\phi(x_j)$$

We know that $w = \sum \alpha_i y_i \phi(x_i)$ if we want to use the vectors of the form $\phi(x_i)$ explicitly. However, in order to classify a pattern x, we need to compute

$$w^t\phi(x) + b = \sum_i \alpha_i y_i \phi(x_i)^t\phi(x) + b = \sum_i \alpha_i y_i K(x_i, x) + b$$

Even b can be computed as follows. Let (x_p, y_p) be a training data pair. So, we have $w^t\phi(x_p) + b = y_p$. So, $b = y_p - w^t\phi(x_p) = y_p - \sum_i \alpha_i y_i K(x_i, x_p)$. So, $w^t\phi(x) + b = \sum_i \alpha_i y_i K(x_i, x) + y_p - \sum_i \alpha_i y_i K(x_i, x_p)$. So, all the computations can be made using the kernel function and the α_is. However, we need to have a suitable kernel function that satisfies $K(x_i, x_j) = \phi(x_i)^t\phi(x_j)$. Some popularly used kernel functions are:

- Polynomial of degree p: $K(x_i, x) = (x_i^t x)^p$
- Polynomial kernel up to degree p: $K(x_i, x) = (1 + x_i^t x)^p$.
- Gaussian kernel: $K(x_i, x) = e^{-\frac{||x_i - x||^2}{\sigma^2}}$.
- Sigmoidal kernel: $K(x_i, x) = tanh(ax_i^t x + b)$

Code

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn import svm

xBlue = np.array([0.3,0.5,1,1.4,1.7,2])
yBlue = np.array([1,4.5,2.3,1.9,8.9,4.1])

xRed = np.array([3.3,3.5,4,4.4,5.7,6])
yRed = np.array([7,1.5,6.3,1.9,2.9,7.1])

X =
np.array([[0.3,1],[0.5,4.5],[1,2.3],[1.4,1.9],[1.7,8.9],[2,4.1],[3.3,7],[3.5,1.5],[4,6.3],[4.4,1.9],[5
.7,2.9],[6,7.1]])
y = np.array([0,0,0,0,0,0,1,1,1,1,1,1]) # 0: blue class, 1: red class

plt.plot(xBlue, yBlue, 'ro', color='blue')
plt.plot(xRed, yRed, 'ro', color='red')
plt.plot(2.5,4.5,'ro',color='green')

#
#       Important parameters for SVC: gamma and C
#               gamma -> defines how far the influence of a single training example reaches
```

```
#                                   Low value: influence reaches far      High value:
influence reaches close
#
#                C -> trades off hyperplane surface simplicity + training examples
missclassifications
#                                   Low value: simple/smooth hyperplane surface
#                                   High value: all training examples classified correctly but
complex surface
classifier = svm.SVC()
classifier.fit(X,y)

print( classifier.predict([[2.5,4.5]]))

plt.axis([-0.5,10,-0.5,10])

plt.show()
```

Output

```
/tmp/ipykernel_3837/477375651.py:14: UserWarning: color is redundantly defined by the 'color
he keyword argument will take precedence.
  plt.plot(xBlue, yBlue, 'ro', color='blue')
/tmp/ipykernel_3837/477375651.py:15: UserWarning: color is redundantly defined by the 'color
he keyword argument will take precedence.
  plt.plot(xRed, yRed, 'ro', color='red')
/tmp/ipykernel_3837/477375651.py:16: UserWarning: color is redundantly defined by the 'color
he keyword argument will take precedence.
  plt.plot(2.5,4.5,'ro',color='green')
[0]
```