Assignment

B ANIRUDH SRINIVASAN
COE17B019

1. Support Vector Machine (SVM)

1.1 Introduction

SVM is a method for the classification of both linear and nonlinear data. In a nutshell, an SVM is an algorithm that works as follows. It uses a nonlinear mapping to transform the original training data into a higher dimension. Within this new dimension, it searches for the linear optimal separating hyperplane (i.e., a "decision boundary" separating the tuples of one class from another). With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane. The SVM finds this hyperplane using support vectors ("essential" training tuples) and margins (defined by the support vectors). Although the training time of even the fastest SVMs can be extremely slow, they are highly accurate, owing to their ability to model complex nonlinear decision boundaries. They are much less prone to overfitting than other methods. The support vectors found also provide a compact description of the learned model. SVMs can be used for numeric prediction as well as classification. They have been applied to a number of areas, including handwritten digit recognition, object recognition, and speaker identification, as well as benchmark time-series prediction tests.

Intuition and Math behind SVM

For linearly separable data, a hyper plane divides the hyperspace in such a way that, points to one side of hyperplane is class 1 and the points on the other side of the hyperplane is class 2.

In mathematical sense, if

$$w^t x + b = 0 \tag{1}$$

is the equation of the hyperplane, then, $w^t x$ + b ¿ 0, all x satisfying this equation, will be of one class and the points satisfying the eq
$w^t x$ + b ¡ 0
will be another class.

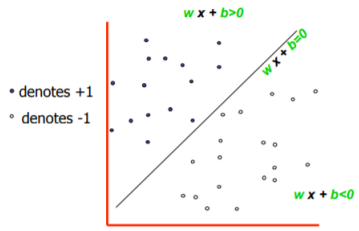

Figure 1: Illustration of how the hyperplane separates points.

We can easily separate the datapoints with many hyperplanes, For example,

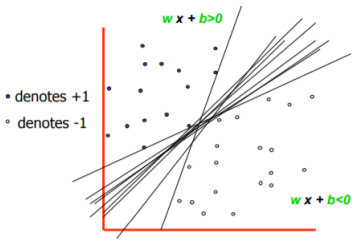

Figure 2: This problem has infinitely many solutions.

But only one of those hyerplane classify the data optimally, and we use svm to idenitify them.

The optimal plane is the one in which the distance between the plane and the class points are maximum. Moreover, the minimum distance between points in class A and the hyperplane and points in class B and the hyperplane must be same, i.e. if the minimum distance of points class A with the hyperplane is greater than that of minimum distance of points in classe B with the hyper plane, then the hyperplane wouldnt be optimum as we tend to classify points more towards class A than class B. For example, consider this one dimensional data set,

Table 1: Example dataset

class	x
class A	1
class A	2
class A	3
class B	5
class B	6
class B	7

The optimum plane (or point) in this case would be x = 4, which is equally spaced between both the classes.

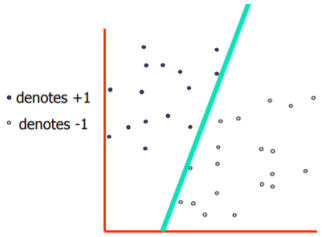

Figure 3: Optimal hyper plane

Support vectors are those planes which run parallel to the decision boundary and pass through the points of both the classes, which have minimum distance with the hyperplane.

The equation of the support vectors is,

$$w^t x + b = \pm 1 \tag{2}$$

The optimal decision boundary is the one in which the margin (distance between the support vectors) is maximum.

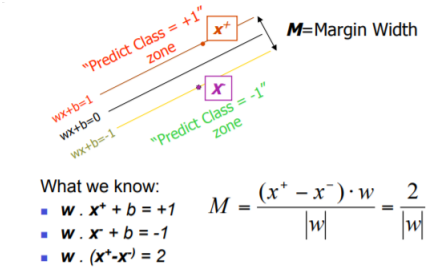

Figure 4: Optimal hyper conditions

From above figure, we can tell that,

$$w^t x + b \geqslant 1 if x \in classA \tag{3}$$
$$w^t x + b \geqslant -1 if x \in classB \tag{4}$$

We can create a label such that, points in class B have label -1 and point in class A has label 1.

Therefore, the above equation becomes,

$$w^t x_i + b \geqslant 1 if y_i \in classA \tag{5}$$
$$w^t x_i + b \geqslant -1 if y_i \in classB \tag{6}$$

We can combine them into one equation,

$$y_i(w^t x_i + b) \geqslant 1 \tag{7}$$

Moreover, we know that margin width,

$$M = \frac{2}{|w|} \tag{8}$$

Maximising M, is same as minimising —w—.

Thus, we minimise,

$$|w| \tag{9}$$

$$and \forall (y_i, x_i) : y_i(w^t x_i + b \geqslant 1) \tag{10}$$

This minimisation problem can be solved by associating Lagrange multiplier with each constraint of this problem.

Cases of SVM

There are three cases of SVM. They are:

- **Hard Margin**

  Here, we try to find a decision boundary that classifies all training points correctly.

  Extremely useful in the case of linearly seperable, but performs poorly in case of non linearly separable and is susceptible to noises.

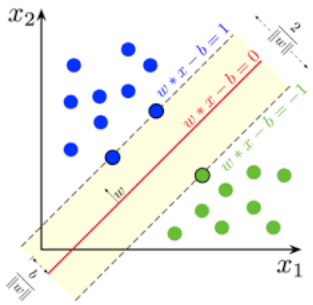

Figure 5: Example of Hard Margin.

- **Soft Margin**

  Here, we try to find a decision boundary that classifies most of the points correctly. We allow some points to be misclassified. The amount of error allowed is set by us.

  This is useful to classify noisy data and some non-linearly separable data.

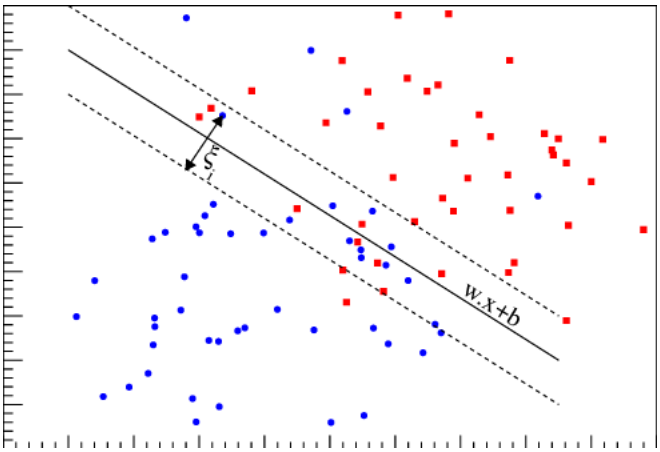

Figure 6: Example of Soft Margin.

- **Kernal**
  Here, we transform the points to a higher dimension (where they are linearly separable) and find a linear hyperplane that classifies the points accurately in that dimension and we plot the contours of that hyperplane on our lower dimension to get a decision boundary.

  This is extremely useful to classify non-linearly seperable data.

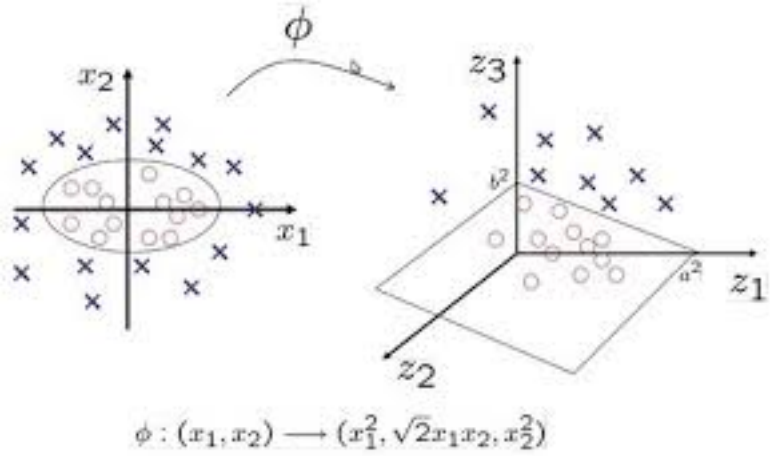

Figure 7: Example of the use of kernal function to transform the input to higher dimension, where they are linearly separable.

1.2 Classifying using SVM with example

Example Dataset

Table 2: Example dataset

class	x	y
class 1 (w1)	2	2
class 1 (w1)	-1	2
class 1 (w1)	1	3
class 1 (w1)	-1	-1
class 1 (w1)	0.5	0.5
class 2 (w2)	-1	-3
class 2 (w2)	0	-1
class 2 (w2)	1	-2
class 2 (w2)	-1	-2
class 2 (w2)	0	-2

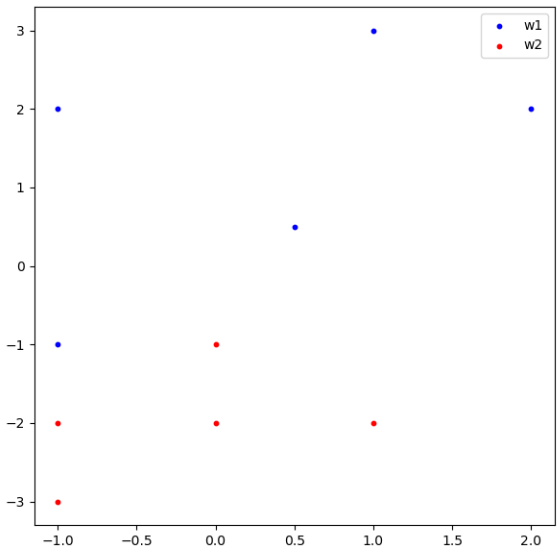

Figure 8: Visualisation of the dataset

Case 1 : Hard Margin

Step 1: Decide the initial weights of the plane.

The Equation of the plane that separates the classes is :

$$w^t x + b = 0 \tag{11}$$

where,

- w is the normal vector of the hyperplane that separates the data. It is represented as a matrix. Hence w^t means transpose of w matrix.
- x is a variable vector.
- b is the bias.

The z matrix would be a matrix where, $z_i = 1$ if x_i in class 1
z_i = -1 if x_i in class 2

Step 2: Finding the Lagrange Multipliers

We need to find the lagrange multipliers for our hyperplane, which inturn would help us to identify the equation of the hyperplane.

$$L(w, b, \alpha) = \frac{||w||^2}{2} - \sum_{k=1}^{n} \alpha_k (1 - z_k(w^t x_k + b)) \tag{12}$$

where,

- w^t is the transpose of weight matrix, and $||w||$ is the magnitude of the normal vector of the hyper plane.
- b is the bias.
- α is a matrix of all the lagrange multipliers. There are totally n lagrange multiplier, where n is the number of datapoints.
- z_k is the class value of a data point x_k. Note that z_k takes either 1 or -1 and not 1 or 0.

After finding few relation between α and weight matrix and bias, the above equation reduces to,

$$L(\alpha) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k,j}^{n} \alpha_k \alpha_j z_k z_j x_j^t x_k \tag{13}$$

We should, find the minimum value of L(α), which satisfy the following constraint,

(a) $\sum_{k=1}^{n} z_k \alpha_k = 0$
(b) $\alpha_k \geqslant 0, k = 1, 2, ..., n$

For our example, we got,

Step 3: Finding Weight Vector and Bias

The weight vector is given by,

$$w = \sum_{k=1}^{n} \alpha_i z_i x_i \tag{14}$$

We know that,
$z_i(w^t x_i + b) = 1$
is the equation of the support planes.

And these support planes would pass through the corner points(that are to the other class end) of each class.

Table 3: *αvalues*

α	Value
α_1	1.26183502 x 10^{-1}
α_2	1.03087027 x 10^{-14}
α_3	2.58278642 x 10^{-15}
α_4	3.45312250 x 10^{0}
α_5	4.21384434 x 10^{1}
α_6	1.64516838 x 10^{-15}
α_7	3.01092476 x 10^{0}
α_8	1.82009356 x 10^{-14}
α_9	9.89765678 x 10^{1}
α_10	8.76371458 x 10^{-15}

Therefore, we need to find the points.

α_i would be negligible for all non - other points/ points through which the support vector wont pass through.

Therefore, with this we can locate the corner point and take avgerage to find avgerage b value.

In our example we get,

w = [-2.0002976 2.00039283] and b = 1.0001476102615467.

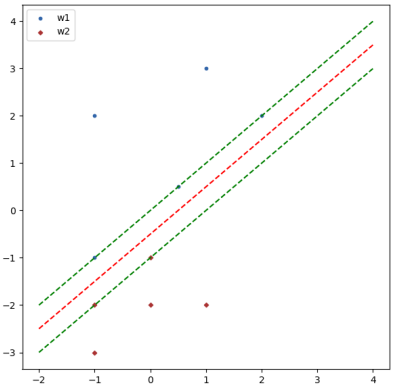

Figure 9: The decision boundary for the example dataset. The red line is the actual decision boundary with green line as the support vectors.

Case 2 : Soft Margin

The steps are same as that of Hard Margin, we only change the equation to allow error.

$$w^t x + b = 1 - \varepsilon \tag{15}$$

$$L(w, b, \varepsilon, \alpha, \gamma) = \frac{||w||^2}{2} + C * \sum_{k=1}^{n} \varepsilon_k - \sum_{k=1}^{n} \alpha_k (z_k (w^t x_k + b) - 1 + \varepsilon_k) - \sum_{i=k}^{n} \gamma_k \varepsilon_k \tag{16}$$

where,

- ε is the error in our classifier.
- $\alpha and \gamma$ are two lagrange multiplier.
- C is a regularisation parameter, which tells us how much error to be present. If C = 0, we get hard margin, and if C is a large value, then we allow a lot of miscalculation as a the error term in $eq(31)$ increases and dominates the equation.

This may seem difficult to solve, but they give the same equation,but differ in constraint.

$$L(\alpha) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k,j}^{n} \alpha_k \alpha_j z_k z_j x_j^t x_k \tag{17}$$

We should, find the minimum value of L(α), which satisfy the following constraint,

(a) $\sum_{k=1}^{n} z_k \alpha_k = 0$
(b) $C \geqslant \alpha_k \geqslant 0, k = 1, 2, ..., n$

The weight vector is given by,

$$w = \sum_{k=1}^{n} \alpha_i z_i x_i \tag{18}$$

The error or slack is giver by,

$$\varepsilon_i = \frac{\alpha_i}{C} \tag{19}$$

Similarly, corresponding α of non corner point will be negligible. With this we find the corner points.

Same way we substitute the corner points in eq(32) to find bias .

Case 3 : Kernal method

Here we transform the input data to a higher dimension using kernal functions.

$$y = \Phi(x) \tag{20}$$

where,

- y is the transformed data.
- x is the input data points.
- $\phi(.)$ is a transformation or kernal function.

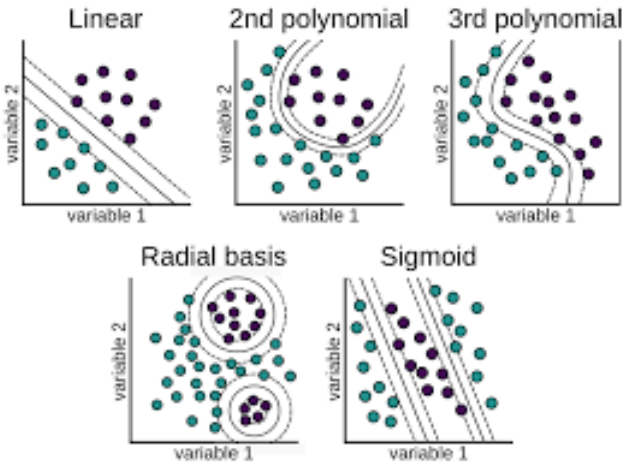

Figure 10: Different kernal functions

After transforming the input data, follow hard margin algorithm to get the equation of plane in the higher dimension.

To make the computation at higher dimension easier, we use a special property of kernal functions. i.e. the dot product of the function is conserved throughout the dimensions.

The contours of that plane on the input dimension would be our decision tree.

1.3 Pseudo-Code SVM

Hard Margin

```
Begin
Initialize all weights to 0;
for each training tuple X in D {
z_i = classlabel(X)
}
n = len(X)
L(α) = Σ_{k=1}^{n} α_k − (1/2) Σ_{k,j}^{n} α_k α_j z_k z_j x_j^t x_k
α = α : L(α) is maximum,
        α_i ⩾ 0, and
        Σ_{i=1}^{n} α_i z_i = 0

Weights = Σ_{i=1}^{n} z_i α_i x_i

for i in len(α){
if(alpha_i ⩾ 0.001)
  b = \dfrac{1}{z_i} − w^t x_i
  add b to the set bias
}
bias = bias.mean()
```

Soft Margin SVM

```
Begin
Initialize all weights to 0;
for each training tuple x in X {
z_i = classlabel(x)
}
initialise the desired value of c.
n = len(X)
L(α) = Σ_{k=1}^{n} α_k − (1/2) Σ_{k,j}^{n} α_k α_j z_k z_j x_j^t x_k
α = α : L(α) is maximum,
        c ⩾ alpha_i ⩾ 0, and
        Σ_{i=1}^{n} α_i z_i = 0

Weights = Σ_{i=1}^{n} z_i α_i x_i
for i in len(X){
 ε_i = \dfrac{α_i}{c}
}
for i in len(X){
if(alpha_i ⩾ 0.001)
  b = \dfrac{1−ε_i}{z_i} − w^t x_i
  add b to the set bias
}
bias = bias.mean()
```

Kernal trick SVM

```
Begin
Initialize all weights to 0;
for each training tuple x in X {
z_i = classlabel(x)
}
Y = ψ(X) //transforming the input into higher dimensions.

weight, bias = HardMargin(Y)
```

1.4 Additional Notes

More about Kernal functions

Consider the following example. A 3-D input vector X = (x1, x2, x3) is mapped into a 6-D space, Z, using the mappings ϕ_1(X) = x1, ϕ_2(X) = x2, ϕ_3(X) = x3, ϕ_4(X) = $(x1)^2$, ϕ_5(X) = x1.x2, and ϕ_6(X) = x1.x3. A decision hyperplane in the new space is

$$d(Z) = W^t Z + b \tag{21}$$

where, W and Z are vectors. This is linear. We solve for W and b and then substitute back so that the linear decision hyperplane in the new (Z) space corresponds to a nonlinear second-order polynomial in the original 3-D input space:

But there are some problems. First, how do we choose the nonlinear mapping to a higher dimensional space? Second, the computation involved will be costly. Refer to

Given the test tuple, we have to compute its dot product with every one of the support vectors. In training, we have to compute a similar dot product several times in order to find the MMH. This is especially expensive. Hence, the dot product computation required is very heavy and costly.

Luckily, we can use another math trick. It so happens that in solving the quadratic optimization problem of the linear SVM (i.e., when searching for a linear SVM in the new higher dimensional space), the training tuples appear only in the form of dot products, ϕ(Xi)· ϕ(Xj), where ϕ(X) is simply the nonlinear mapping function applied to transform the training tuples. Instead of computing the dot product on the transformed data tuples, it turns out that it is mathematically equivalent to instead apply a kernel function, K(Xi, Xj), to the original input data. That is,

$$K(Xi, Xj) = \phi(Xi).\phi(Xj) \tag{22}$$

In other words, everywhere that ϕ(Xi)· ϕ(Xj) appears in the training algorithm, we can replace it with K(Xi ,Xj). In this way, all calculations are made in the original input space, which is of potentially much lower dimensionality! We can safely avoid the mapping—it turns out that we don't even have to know what the mapping is! We will talk more later about what kinds of functions can be used as kernel functions for this problem.

After applying this trick, we can then proceed to find a maximal separating hyperplane. kernel functions that could be used to replace the dot product scenario just described,

$$Polynomialkernelofdegreeh : K(Xi, Xj) = (Xi\Delta Xj + 1)^h \tag{23}$$

$$Gaussianradialbasisfunctionkernel : K(Xi, Xj) = e^{-\frac{||kXiXjk||^2}{2\sigma^2}} \tag{24}$$

$$Sigmoidkernel : K(Xi, Xj) = tanh(\kappa XiXj - \delta) \tag{25}$$