II-B.Tech-CSE-AI-Semester 4,2020-2024

21MAT212 - Mathematics for Intelligent Systems

END-SEMESTER PROJECT

SUPPORT VECTOR MACHINES FOR BINARY CLASSIFICATION

AMRITA VISHWA VIDYAPEETHAM,ETTIMADAI, COIMBATORE.

Submitted by

G.Krishna Rayalu (CB.EN.U4AIE20016)

Jangala Gouthami (CB.EN.U4AIE20023)

N.Manish Rama Gopal (CB.EN.U4AIE20037)

D.Rohith(CB.EN.U4AIE20060)

Submitted on : 05-07-2022

INDEX

Introduction:

In this project we will be formulating SVM algorithm as an objective function with some constraints and we will be applying the concept of Lagrangian function.

Binary Classification:

Binary classification refers to those classification tasks that have two class labels.

Examples include:

- Email spam detection (spam or not).
- Churn prediction (churn or not).
- Conversion prediction (buy or not).

Typically, binary classification tasks involve one class that is the normal state and another class that is the abnormal state.

For example, “not spam” is the normal state and “spam” is the abnormal state. Another example is “cancer not detected” is the normal state of a task that involves a medical test and “cancer detected” is the abnormal state.

Support Vector Machine:

What is Support Vector Machine

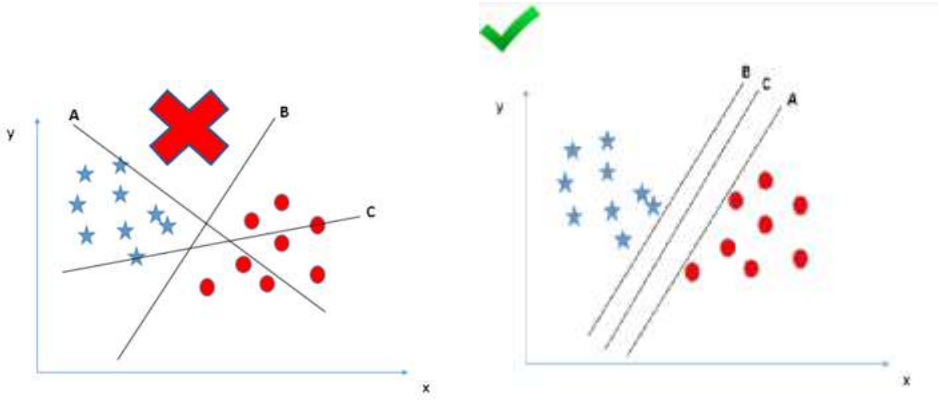
Support vector machine is a supervised machine learning algorithm which is used for both classification or regression challenges. In this project we will be focusing on classification challenges.

In SVM algorithm we plot each data point in n-dimensional space (n-number of features in dataset) with the value of each feature being a particular coordinate.

The classification is then done by finding a hyperplane that differentiates the two classes.

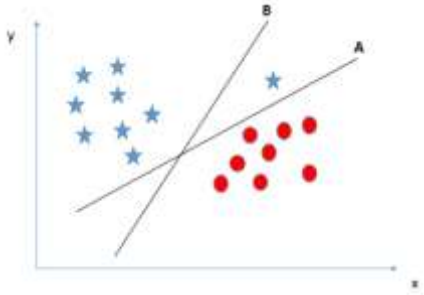
Working of SVM:

1. Identification of a hyperplane:



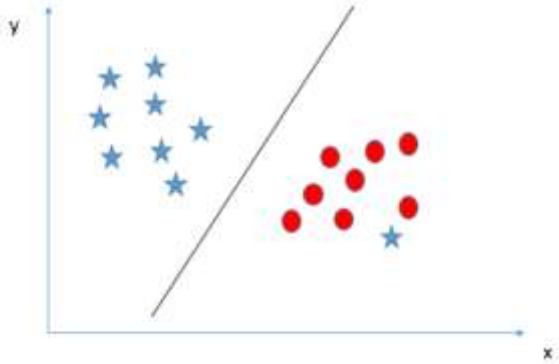
Hyperplane is selected such that the distance is maximum between nearest data point of either class. This distance is called as Margin. (Maximal Margin Hyperplane is selected)

2. Identification of Right Hyperplane:



The right hyperplane must be selected; even though a respective hyperplane 'B' has higher margin, 'A' hyperplane must be selected as it classifies properly.

3. SVM is robust to outliers:



Hence here we have the requirement for different constraints that is the two types of SVM classification techniques:
When the data is linearly separable, and we don't want to have any misclassifications, we use SVM with a hard margin. However, when a linear boundary is not feasible, or we want to allow some misclassifications in the hope of achieving better generality, we can opt for a soft margin for our classifier.

In this project we will be focusing only on the Linear Kernel technique.

Formulation:

Formulating The 'Hard Margin' and the 'Soft Margin' techniques:

We will be formulating the hyperplane equation from the basic line equation that is:

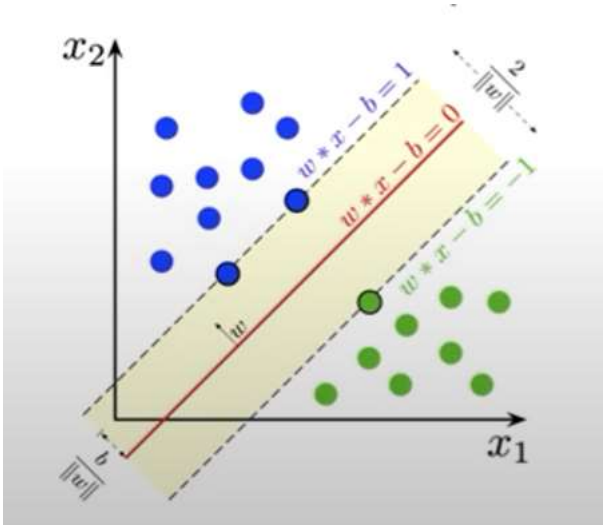
$$ax + by = c$$

Where (a,b) are the coefficients and 'c' is a constant.

So the distance of a point (m,n) to the line will be:

$$\frac{|a(m) + b(n) - c|}{\sqrt{a^2 + b^2}}$$

Now we will be applying this concept to find the equation for a hyperplane of 'N' dimension.



From the above plot we can see that the separation plane is clearly classifying the datapoints, as we can see the hyperplane (in this case a plane) can be represented as a equation of a line that is

$$wx - b = 0$$

Where 'w' is a coefficient of variable 'x' and 'b' is a constant.

When taken to higher dimension Hyperplane comes into existence, which can be represented as an equation as follows:

$$\bar{w}.\bar{x} - b = 0$$

Where '$\bar{w}$' represents a weighted vector; the coefficient of vector '$\bar{x}$'

Where '$\bar{x}$' represents $x_1, x_2, x_3 \ldots . x_k$

Similarly, we can formulate the equation of the support vectors, as the lines are parallel to the hyperplane, they have the same coefficients only the constant that they are being equated to get changed:

$$\bar{w}.\bar{x} - b = c$$

This is the generic equation for the first class.

$$\bar{w}.\bar{x} - b = -c$$

This is the generic equation of the second class.

We can now assume a value for 'c' which can be acting as a feature for classification. (The value of 'c' doesn't affect the generic equation because variables are not affected, only notations are changed which have no impact)

Let (m,n) be a point closest to the opposition class, the distance of that point to the hyperplane $\bar{w}.\bar{x} - b = 0$ can be formulated as:

$$\frac{|w1(m) + w2(n) - b|}{\sqrt{{w_1}^2 + {w_2}^2}}$$

As (m,n) is the closest point to the opposition class the support vector definitely passes through the point so the point can substitute them in the equation of support vector:

Hence our distance becomes:

$$\frac{1}{\sqrt{{w_1}^2 + {w_2}^2}} = \frac{1}{\|w\|_2}$$

The distance from both of the support vectors will be:

$$\frac{2}{\sqrt{{w_1}^2 + {w_2}^2}} = \frac{2}{\|w\|_2}$$

Now formulating this into an objective function with constraints:

We take the two types of SVM:

Hard Margin formulation (No Penalty):

Our objective is to maximise the margin hence It can be formulated as:

$$\text{Max}_{w,b}\left[\frac{2}{\|w\|_2}\right]$$

but this can also be achieved if minimise the inverse of it that is:

$$\text{min}_{w,b}\frac{1}{2}\|w\|_2^2$$

now we can write the constraints based on 'xi' an i^{th} datapoint associated to y^i (Here y^i can represents feature '1' or '-1')

The Constraint:

$w^T.x_i - b \geq 1 \; if \; y_i = 1$ if datapoint belongs to '1' class

$w^T.x_i - b \leq -1 \; if \; y_i = -1$ if datapoint belongs to '-1' class

These are the above two conditions that will give us the constraint, on combining them:

$$(w^T.x_i - b)y_i \geq 1$$

For whatever the above two conditions when multiplied with 'y_i' will always be greater than 1.

Hence the final objective function is:

$$\min_{w,b} \frac{1}{2}\|w\|_2^2$$

Subject to:

$$y_i(w^T.x_i - b) \geq 1 \qquad i=1,2,\ldots..,k$$

Hence the final Lagrangian equation can be written by using the gradient property:

$$L(w,b,u)=\frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{k(number\ of\ datapoints)} u_i(y_i(w^T x_i - b) - 1)$$

The matrix form for this objective function will be:

$$\min_{w,b} \frac{1}{2}w^T w$$

Subject to:

$$Y(X.w - eb) - e \geq 0$$

Where 'e' is a column vector of ones, of dimension $k \times 1$

Lagrangian function Matrix form:

$$L(w,b,u)= \frac{1}{2}w^T w - u^T(Y(X.w - eb) - e)$$

Lagrangian Dual:

The above problem can be solved using Lagrangian dual:

$$\max_{u>=0}[\min_{w,d} L(w,b,u)]$$

$$L(w,b,u)=\frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{k(number\ of\ datapoints)} u_i(y_i(w^T x_i - b) - 1)$$

For a given 'u' we find the values of w,b such that the lagrangian function is minimised.

We know that at minimum,

$$\frac{\partial L}{\partial b} L(w,b,u) = 0 \ and \ \frac{\partial L}{\partial w} = 0$$

On solving the above two equations we get

$$\sum_{i=1}^{k} u_i y_i = 0$$

And

$$W= \sum_{i=1}^{k} u_i y_i x_i = 0$$

Substituting $w=\sum_{i=1}^{k} u_i y_i x_i = 0$ into the Lagrangian equation:

$$L(w,b,u)= \frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{k} ui(y_i(w.x_i - b) - 1)$$

$$\frac{1}{2}(w^T w) - \sum_{i=1}^{k} ui(y_i(w^T x_i - b) - 1)$$

$$\frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k} u_i u_j y_i y_j\,(x_i^T x_j) - \sum_{i=1}^{k} u_i y_i (w^T x_i) + b\sum_{i=1}^{k} u_i y_i + \sum_{i=1}^{k} u_i$$

Since $\sum_{i=1}^{k} u_i y_i = 0$

$$L(w,b,u)=$$

$$\frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k} u_i u_j y_i y_j\left(x_i^T x_j\right) - \sum_{i=1}^{k}\sum_{j=1}^{k} u_i u_j y_i y_j\left(x_i^T x_j\right) + \sum_{i=1}^{k} u_i$$

Now Lagrangian is a function of 'u' alone and we denote the function as

$$L_D(u)=\sum_{i=1}^{k} u_i - \frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k} u_i u_j y_i y_j (x_i^T x_j)$$

We know that 'u' must satisfy the condition $\sum_{i=1}^{k} u_i y_i = 0$ .The optimization problem now reduces to:

$$\text{Max } L_D(u)=\sum_{i=1}^{k} u_i - \frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k} u_i u_j y_i y_j (x_i^T x_j)$$

Subject to

$$\sum_{i=1}^{k} u_i y_i = 0, u_i \geq 0, i = 1,2,3..k$$

Till now we have formulated for hard margin SVM that is it can be well applied only for separable data, if we get a dataset of non-separable data we can then use Soft Margin SVM which adds a penalty to each datapoint that is wrongly classified and brings out a better classification:

Soft Margin SVM:

If all the points are not linearly separable, we will be adding error or I other words we allow points to lie between bounding hyperplanes and beyond .When a point, say (m,n) in A+ lies either between the bounding hyperplanes or beyond the class it should be in, that is in class A- then we add an error to the inequality:

$$(w^T.x_i - b) + \varepsilon_i \geq 1$$

And for a point in A- which should actually lie in A+ we substract a surplus variable to the inequality:

$$(w^T.x_i - b) - \varepsilon_i \leq -1$$

Hence the final objective function:

$$\text{Min}_{\text{w,b}} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{k}\varepsilon_i$$

Subject to:

$$y_i(w^T.x_i - b) + \varepsilon_i - 1 \geq 0 \quad ; 1 \leq i \leq k$$

$$\varepsilon_i \geq 0 \quad ; 1 \leq i \leq k$$

Now we will be using the above maximised objective function as a constraint so the final Lagrangian function will be:

L(w,b,u,μ) =

$$\frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{k}\varepsilon_i - \sum_{i=1}^{k}u_i\left[y_i(w^T x_i - b) + \varepsilon_i - 1\right] - \sum_{i=1}^{k}\mu_i\varepsilon_i$$

The matrix form for this objective function will be:

$$min_{c,w,\varepsilon}\ \frac{1}{2}w^T w + Ce^T\varepsilon$$

Subject to:

$$Y(X.w - eb) + \varepsilon - e \geq 0$$

$$\varepsilon \geq 0$$

Where ‘e’ is a column vector of ones, of dimension $k \times 1$

Lagrangian function Matrix form:

L(w,b,u,μ)= $\frac{1}{2}w^T w + Ce^T\varepsilon - u^T(Y(X.w - eb) + \varepsilon - e) - \mu^T\varepsilon$

Lagrangian Dual:

With respect to Lagrangian multipliers u,μ dual problem is:

$$Max_{u,\mu}\, L(w, b, \varepsilon, u, \mu)$$

Subject to:

$$\frac{\partial L}{\partial w} = 0$$

$$\frac{\partial L}{\partial b} = 0$$

$$\frac{\partial L}{\partial \varepsilon_i} = 0\ 1 \le i \le k$$

$$u \ge 0$$

$$\mu \ge 0$$

The constraint $\frac{\partial L}{\partial w} = 0$ implies

$$w = \sum_{i=1}^{k} u_i y_i x_i$$

The constraint $\frac{\partial L}{\partial b} = 0$ implies

$$\sum_{i=1}^{k} u_i y_i = 0$$

The constraints $\frac{\partial L}{\partial \varepsilon_i} = 0$ implies

$$C - u_i - \mu_i = 0; 1 \le i \le k$$

Note that u>0 and k>0

We have:$0 \le u_i \le C$

Substituting these results into $L(w, b, \varepsilon, u, \mu)$,

$$L(w,b,\varepsilon,u,\mu)=\frac{1}{2}w^Tw+\sum_{i=1}^{k}0\times\varepsilon_i-w^Tw-0\times b+\sum_{i=1}^{k}u_i$$

$$=-\frac{1}{2}w^Tw+\sum_{i=1}^{k}u_i$$

$$=\sum_{i=1}^{k}u_i-\frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k}y_iy_jx_i^Tx_ju_iu_j$$

The final dual problem is:

$$\mathrm{Max_u}L_D(\mathrm{u})=\sum_{i=1}^{k}u_i-\frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k}y_iy_jx_i^Tx_ju_iu_j$$

Subject to:

$$\sum_{i=1}^{k}u_iy_i=0\,,0\le u_i\le C\ and\ 1\le i\le k$$

L2 Norm Linear SVM:

In L2 norm SVM, the sum of square of error variable are minimised along with reciprocal of the square of the margin between the bounding planes. The formulation of the problem is given by

$$\min_{w,d,\mathcal{E}}\frac{1}{2}w^Tw+\frac{c}{2}\sum_{i=1}^{k}\mathcal{E}_i^2$$

$$y_i(w^Tx_i-b)+\mathcal{E}_i-1\ge 0;1\le i\le k$$

$$\mathcal{E}_i\ge 0;1\le i\le k$$

The Lagrangian is

$$L(w,b,\mathcal{E},\mathrm{u})=\frac{1}{2}w^Tw+\frac{c}{2}\sum_{i=1}^{k}\mathcal{E}_{\mathrm{i}}^2-\sum_{i=1}^{k}u_i[y_i(w^Tx_i-b)+\mathcal{E}_{\mathrm{i}}-1]$$

$$=\frac{1}{2}w^Tw+\frac{c}{2}\sum_{i=1}^{k}\mathcal{E}_{\mathrm{i}}^2-(\sum_{i=1}^{k}u_iy_ix_i^T)\,w-b(\sum_{i=1}^{k}u_iy_i)-\sum_{i=1}^{k}u_i\mathcal{E}_{\mathrm{i}}+\sum_{i=1}^{k}u_i$$

Note that in Lagrangian,the Lagrangian multipliers corresponding to the constraint $\varepsilon_i\ge 0;1\le i\le k$ have not been considered. This is because ε_i becomes a function of u_i alone and all u_i are unbounded as derived in the following:

$$\max_{u,\mu}\boldsymbol{L(w,b,\xi,u,\mu)}$$

$$\frac{\partial L}{\partial \boldsymbol{w}} = 0$$

$$\frac{\partial L}{\partial b} = 0$$

$$\frac{\partial L}{\partial \xi_i} = 0$$

$$1 \le i \le k \boldsymbol{u} \ge \boldsymbol{0}$$

$$The\ constraint\ \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{w}} = 0\ implies:$$

$$\boldsymbol{w}^T - \sum_{i=1}^{k} u_i y_i \boldsymbol{x}_i^T = 0$$

$$\boldsymbol{w} = \sum_{i=1}^{k} u_i y_i \boldsymbol{x}_i$$

$$The\ constraint\ \frac{\partial \boldsymbol{L}}{\partial b} = 0\ implies:$$

$$\sum_{i=1}^{k} u_i y_i = 0$$

$$The\ constraint\ \frac{\partial \boldsymbol{L}}{\partial \boldsymbol{\xi_i}} = 0\ imply:$$

$$\boldsymbol{C}\xi_i - u_i = 0, \qquad 1 \le i \le k.$$

$$Note\ that\ \boldsymbol{u} \ge \boldsymbol{0},$$

$$Substituting\ these\ results\ into\ \boldsymbol{L}(\boldsymbol{w}, \boldsymbol{b}, \boldsymbol{\xi}, \boldsymbol{u}), L(w, b, \xi, u)$$

$$= \frac{1}{2} w^T w + \frac{1}{2C} \sum_{i=1}^{k} u_i^2 - w^T w - 0 * b - \frac{1}{C} \sum_{i=1}^{k} u_i^2 + \sum_{i=1}^{k} u_i$$

$$= -\frac{1}{2} w^T w + \sum_{i=1}^{k} u_i - \frac{1}{2C} \sum_{i=1}^{k} u_i^2 = \sum_{i=1}^{k} u_i - \frac{1}{2} \sum_{i=1}^{k} \sum_{j=1}^{k} y_i y_j x_i^T x_j u_i u_j - \frac{1}{2C} \sum_{i=1}^{k} u_i^2$$

$$To\ summarise,\ the\ dual\ problem\ is$$

$$\max_{\boldsymbol{u}} \boldsymbol{L}(\boldsymbol{u}) = \sum_{i=1}^{k} u_i - \frac{1}{2} \sum_{i=1}^{k} \sum_{j=1}^{k} y_i y_j u_i u_j \left(\boldsymbol{x}_i^T \boldsymbol{x}_j + \frac{\delta_{ij}}{\boldsymbol{C}}\right)$$

$$subject\ to$$

$$\sum_{i=1}^{k} y_i u_i = 0$$

$$u_i \ge 0 \quad 1 \le i \le k$$

$$The\ standard\ form\ in\ matrix\ format\ is$$

$$\min_{u} L(u) = \frac{1}{2} u^T Y \left(AA^T + \frac{I}{C}\right) Yu - e^T u$$

$$subject\ to$$

$$y^T u = 0$$

$$\min_{u} L(u) = \frac{1}{2} u^T Q u - e^T u$$

$$subject\ to$$

$$y^T u = 0$$

$$u \geq 0$$

$$In\ matlab\ notation,\ Q\ can\ be\ computed\ as\ Q \quad = (A * A^T + I/C).* (y * y^T)$$

Formulation of 'b':

In order to compute 'b', we use the lagrangian multiplier which lies between 0 and C, In other words the point must lie on the hyperplane. Hence the equation for such point can be written as,

$$w^T x - b = y_i$$

From the above equation 'b' can be written as

$$b = w^T x_i - y_i$$

But there can be more than one point which lie on the hyperplane, hence we compute 'd' for all such points and find the average. So, the above equation can be written as,

$$b = \sum_{j\ \epsilon\ sindex} u_j y_j x_j^T x_i - y_i$$

In the above equation 'sindex' represents the indices of all support vectors.

The above equation for 'γ' can be written as,

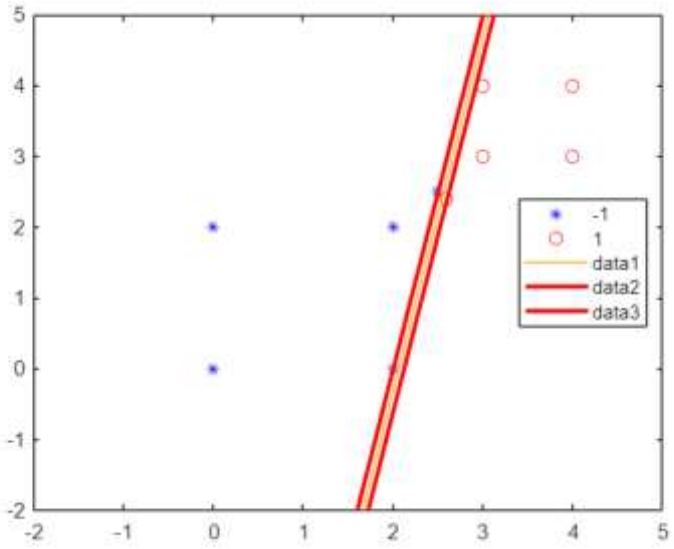
$$b = \frac{1}{|svmindex|} [\sum_{i\ \epsilon\ svmindex} (\sum_{j\ \epsilon\ sindex} u_j d_j x_j^T x_i) - y_i]$$

In the above equation 'svmindex' represents the indices of support vectors which lie on hyperplane.

Implementation of SVM in MATLAB

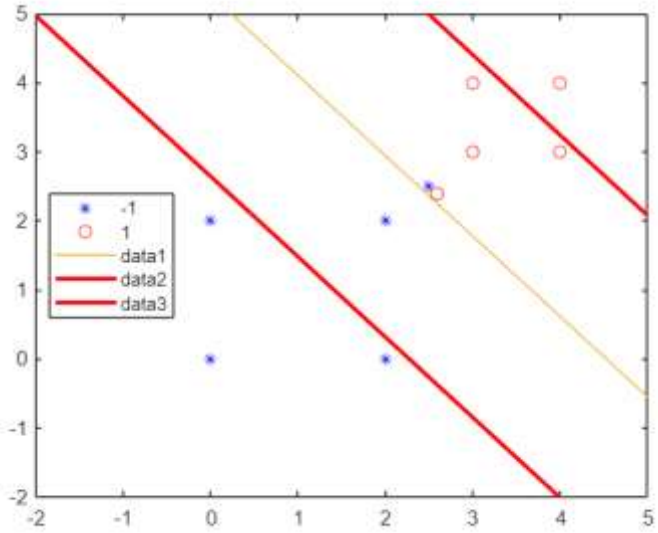
Solving objective functions using CVX and plotting hyperplane.

Hard margin:

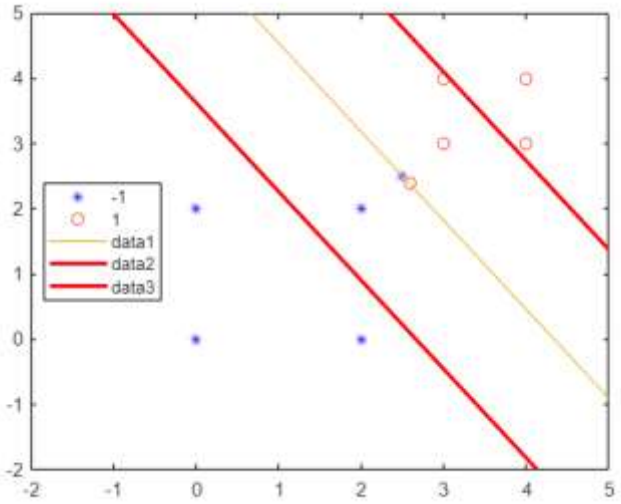


In the above plot the orange line represents the hyperplane and the two red lines represent the bounding plane. There are total three points lying on the bounding plane. These points are the support vectors.

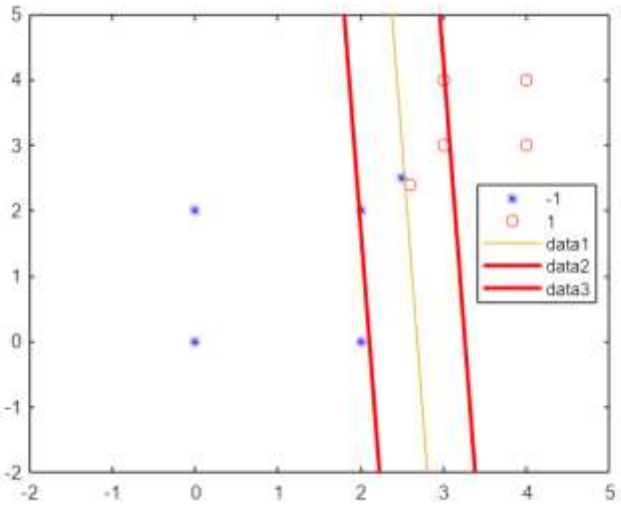
Soft margin:



For c = 0.5



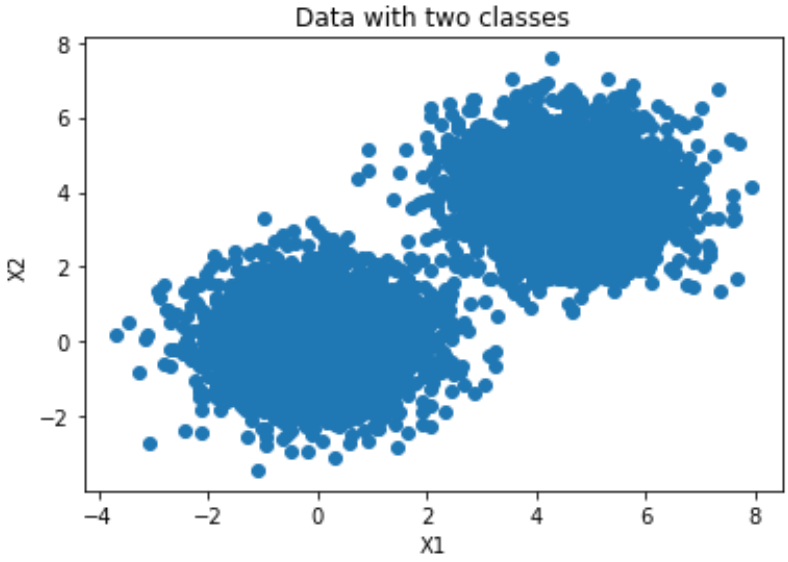
For c = 1

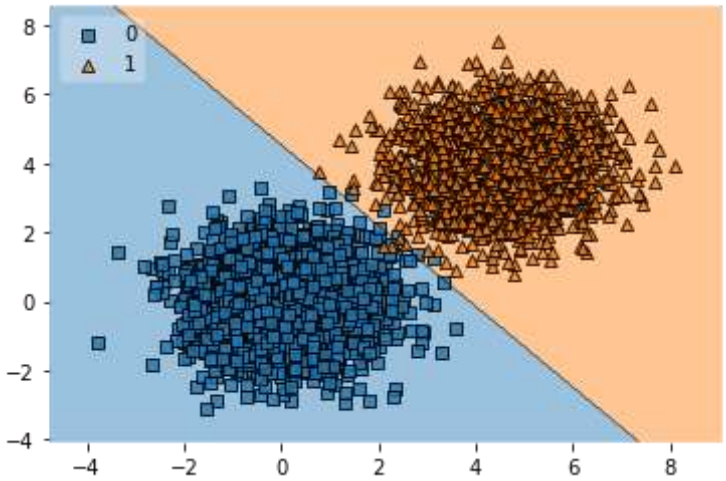


For c = 10

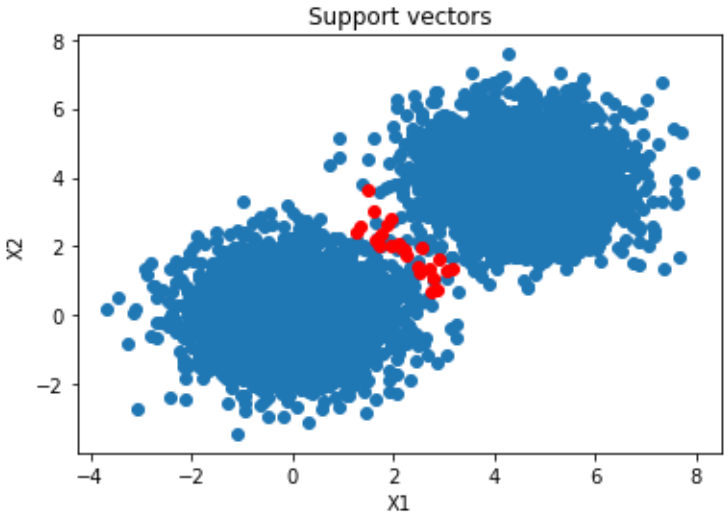
We can observe that as we change the control parameter there is a change in hyperplane and bounding line. Initially there is a misclassification when c = 1 and 0.5 but later it got corrected when c = 10. There are three points which are on bounding line, which means there are three support vectors.

SVM on randomly generated data in python:





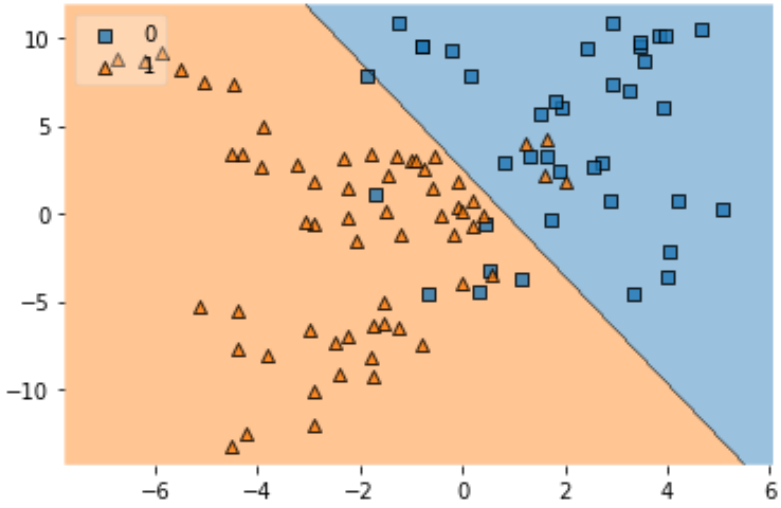
Hyperplane representation for the data



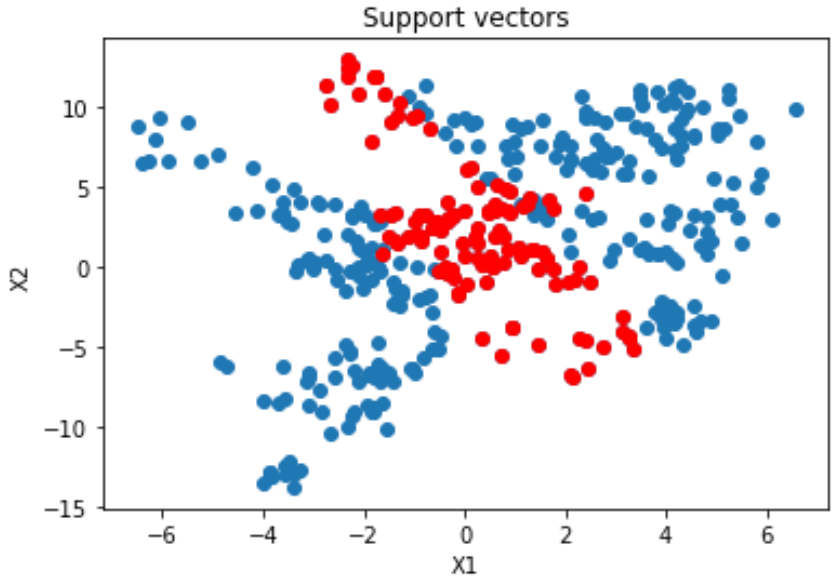
Support vectors representation for data

SVM on real life data in python:

We apply SVM linear classifier on real life bill authentication data. The data has two classes ‘0’ and ‘1’.



Hyperplane representation for the data



Support vector representation for the data.

Multiclass classification:

Problems which involve classifying the data into one of the three or more classes or labels present in the dataset is termed as multiclass classification. In this section we boil down the multiclass classification problem to a binary classification problem.

There are two ways to reduce a multiclass classification problem to binary classification.

- One-vs-One approach

- One-vs-Rest approach

One-vs-One:

Consider the given data has m classes. In one vs one approach $\frac{m*(m-1)}{2}$ classifiers are built. So if the data has 3 classes A,B and C; three classifiers are constructed (A-B,B-C, A-C). For a given data point from the dataset, the classifier classifies the point based on voting strategy.

One-vs-All:

In this approach binary classifiers are constructed for every class present in data. So, if there are m classes m binary classifiers are created. When a datapoint is given, all the m classifiers provide a decision value to the datapoint. The classifier which has the maximum decision value is assigned as the label to the data point.

When to choose OvO and when to choose OvA?

Algorithms like SVM scale very poorly for the size of training dataset. Hence, for these algorithms we use OvO as it is faster and much easier to train many classifiers on small training sets. For algorithms which solve problems of binary classification we use OvA approach.

Conclusion:

Through this project we have learnt that SVM is a great technique for binary classification as it is advantageous in many ways that is it is faster, it can be performed on both small and large datasets, irrelevant features can be handled with ease. We were able to apply it on real life datasets which gave scientifically important results.

Code Appendix:

```
%Solving hard margin svm using CVX

clc;
clear all;
close all;

A = [0 0;2 0; 2 2;0 2;2.5 2.5;2.6 2.4; 3 3;4 3;4 4;3 4]; % data points
nc = 5; % amount of data to be assigned to each class
d = [-1*ones(nc,1);ones(nc,1)]; % class labels
D = diag(d); % diagonal matrix of class labels
n = size(A,2);
m = size(A,1);

e = ones(m,1);

cvx_begin quiet
 variables w(n) g
 dual variable u
 minimize (0.5*w'*w) % objective function
 subject to % constraints
 u:D*(A*w-g) >= e;
cvx_end
w
```

```
w = 2×1
   16.6667
   -3.3333
```

```
u %lagrangian multipliers
```

```
u = 10×1
    0.0000
    4.4444
    0.0000
    0.0000
  140.0002
  144.4446
    0.0000
    0.0000
    0.0000
    0.0000
```

```
idx = [find(u>0.1)] % finding indices of u>0
```

```
idx = 3×1
     2
     5
     6
```

```
%respective active constraints
syms a b c
for i = 1:length(idx)
    k = idx(i);
    if u(k) ~=0
        d(k)*(A(k,1)*a + A(k,2)*b - c) >=1
    end
end
```

ans = $1 \leq c - 2a$

ans =

$$1 \leq c - \frac{5b}{2} - \frac{5a}{2}$$

ans =

$$1 \leq \frac{13a}{5} + \frac{12b}{5} - c$$

```
% testing
z = sign(A*w-g) % predicted class labels
```

```
z = 10×1
    -1
    -1
    -1
    -1
    -1
     1
     1
     1
     1
     1
```

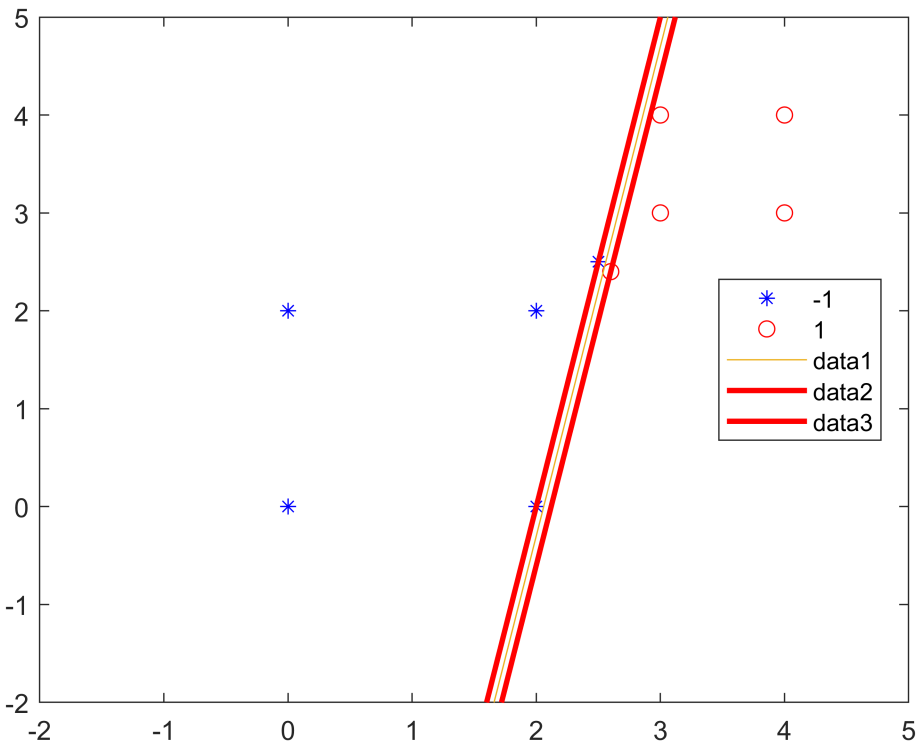
```
r = sum(d==z) % finding errors in predection
```

```
r = 10
```

```
Acc = (r/m)*100 % calculating accuracy
```

```
Acc = 100
```

```
%visulaisation
figure;
gscatter(A(:,1),A(:,2),d,'br','*o');hold on
% linear classifier
x1 = -2:5;
y1 = (g - w(1)*x1)/w(2);
plot(x1,y1);hold on;

% boundary lines
x3 = (1 + g - w(1)*x1)/w(2);
plot(x1,x3,'r','linewidth',2);hold on;

x4 = (-1 + g - w(1)*x1)/w(2);
plot(x1,x4,'r','linewidth',2);
axis([-2 5 -2 5])
hold off
```




```
%Solving soft margin svm using CVX
clc;
clear all;
close all;

A = [0 0;2 0; 2 2;0 2;2.5 2.5;2.6 2.4; 3 3;4 3;4 4;3 4]; % 10 data points
nc = 5; % no of data per class
d = [-1*ones(nc,1);ones(nc,1)]; % class labels
D = diag(d); % diagonal matrix
n = size(A,2);
m = size(A,1);

e = ones(m,1);
%change values of 'c' and check hyperplane plot for each value.
c = 10; % control parameter
cvx_begin quiet
 variables w(n) g Psi(m)
 dual variable u
 minimize ((0.5*w'*w)+(0.5*c*sum(Psi.^2))) % objective function
 subject to % constraints
 u:D*(A*w-g)+Psi >= e;
 Psi >= 0;
cvx_end
w
```

```
w = 2×1
    1.7345
```

```
    0.1056
```

```
u %lagrangian multipliers
```

```
u = 10×1
    0.0000
    0.0000
    0.4223
    0.0000
    9.6229
    8.7481
    1.1765
    0.0000
    0.0000
    0.1207
```

```
idx = [find(u>0.1)] % finding indices for u>0
```

```
idx = 5×1
     3
     5
     6
     7
    10
```

```
%respective active constraints
syms a b c p
for i = 1:length(idx)
    k = idx(i);
    if u(k) ~=0
        d(k)*(A(k,1)*a + A(k,2)*b - c + p) >=1
    end
end
```

```
ans = 1 ≤ c − 2 b − 2 a − p
ans =
```

$$1 \le c - \frac{5\,b}{2} - \frac{5\,a}{2} - p$$

```
ans =
```

$$1 \le \frac{13\,a}{5} + \frac{12\,b}{5} - c + p$$

```
ans = 1 ≤ 3 a + 3 b − c + p
ans = 1 ≤ 3 a + 4 b − c + p
```

```
%testing
z = sign(A*w-g) % predicted class labels
```

```
z = 10×1
    -1
    -1
    -1
    -1
    -1
     1
     1
     1
```

```
     1
     1
```

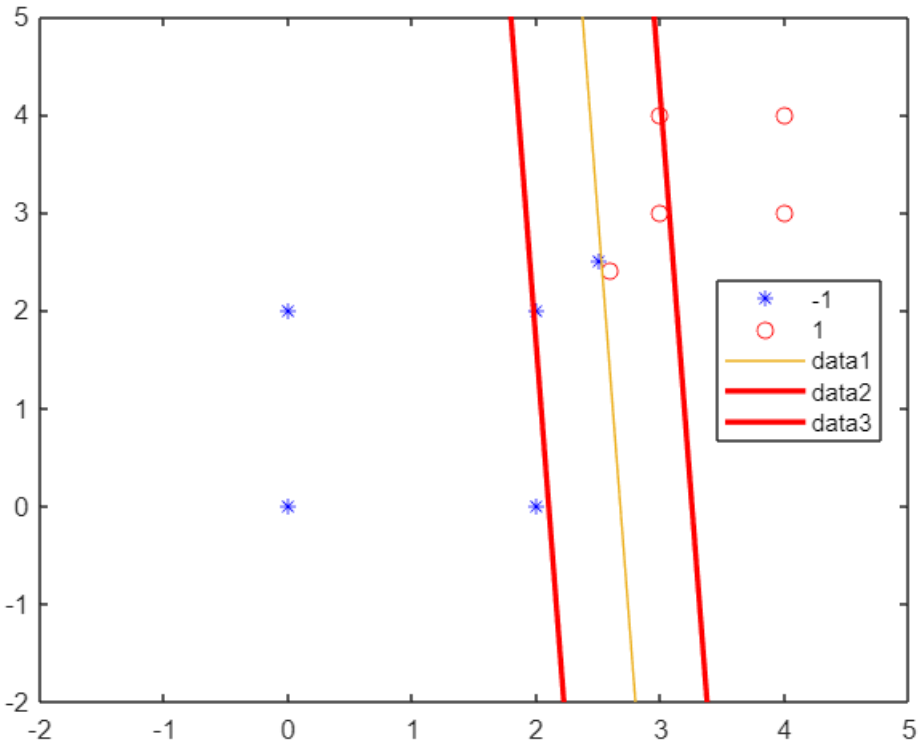
```
r = sum(d==z) % finding errors in predection
```

```
r = 10
```

```
Acc = (r/m)*100 % calculating accuracy
```

```
Acc = 100
```

```
%visualization
figure;
gscatter(A(:,1),A(:,2),d,'br','*o');hold on
% linear classifier
x1 = -2:5;
y1 = (g - w(1)*x1)/w(2);
plot(x1,y1);hold on;

%bounding lines
x3 = (1 + g - w(1)*x1)/w(2);
plot(x1,x3,'r','linewidth',2);hold on;
x4 = (-1 + g - w(1)*x1)/w(2);
plot(x1,x4,'r','linewidth',2);
axis([-2 5 -2 5])
hold off
```



```
%computation of gamma
clc;
clear all;
close all;

A = [0 0;2 0; 2 2;0 2;2.5 2.5;2.6 2.4; 3 3;4 3;4 4;3 4]; % 10 data points
nc = 5; % no of data per class
D = [-1*ones(nc,1);ones(nc,1)]; % class labels
%D = diag(d); % diagonal matrix

c = 10
```

```
c = 10
```

```
epsilon = 1e-3
```

```
epsilon = 1.0000e-03
```

```
Q = (A*A').*(D*D')
```

```
Q = 10×10
         0         0         0         0         0         0         0         0 ···
         0    4.0000    4.0000         0    5.0000   -5.2000   -6.0000   -8.0000
         0    4.0000    8.0000    4.0000   10.0000  -10.0000  -12.0000  -14.0000
         0         0    4.0000    4.0000    5.0000   -4.8000   -6.0000   -6.0000
         0    5.0000   10.0000    5.0000   12.5000  -12.5000  -15.0000  -17.5000
         0   -5.2000  -10.0000   -4.8000  -12.5000   12.5200   15.0000   17.6000
         0   -6.0000  -12.0000   -6.0000  -15.0000   15.0000   18.0000   21.0000
         0   -8.0000  -14.0000   -6.0000  -17.5000   17.6000   21.0000   25.0000
         0   -8.0000  -16.0000   -8.0000  -20.0000   20.0000   24.0000   28.0000
         0   -6.0000  -14.0000   -8.0000  -17.5000   17.4000   21.0000   24.0000
```

```
[m,n] = size(Q);

g = -ones(m,1);
LB = zeros(m,1);
UB = c*ones(m,1);
dt = D';
bequ = 0;
u = quadprog(Q,g,[],[],dt,bequ,LB,UB); %finding solution for 'u' using QP
```

```
Minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
```

```
u
```

```
u = 10×1
    0.0000
    0.0001
    0.9999
    0.0000
   10.0000
   10.0000
```

```
    0.9999
    0.0000
    0.0000
    0.0000
```

```
svind = find(u>epsilon) % points with u>0
```

```
svind = 4×1
     3
     5
     6
     7
```

```
nsv = length(svind);
svmind = find((u>epsilon)&(u<(c-epsilon))) % support vectors on bounding plane
```

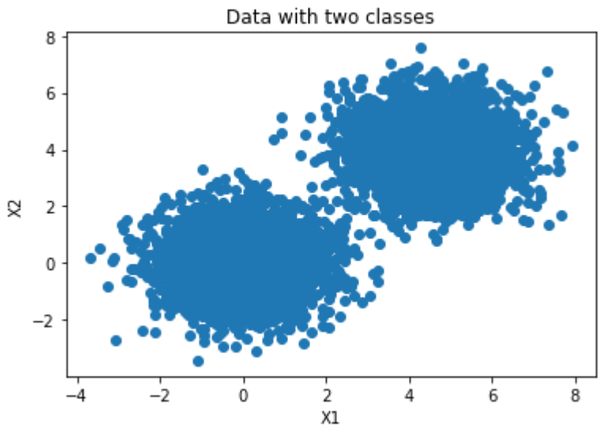
```
svmind = 2×1
     3
     7
```

```
length(svmind);

gamma = sum((Q(svmind,svind)*u(svind).*D(svmind))-D(svmind))/length(svmind) %final gamma value
```

```
gamma = 5.0003
```

In [1]:
```
# Imports
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.metrics import plot_confusion_matrix
from mlxtend.plotting import plot_decision_regions
import pandas as pd
import seaborn as sns
```

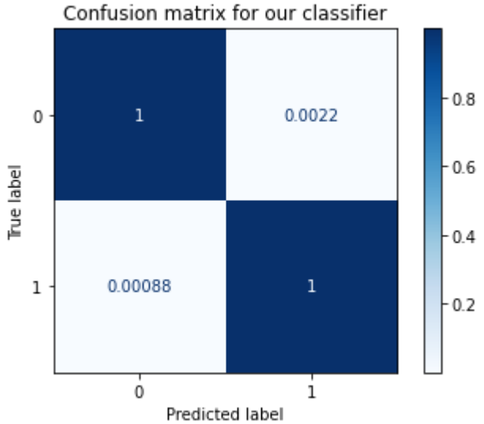
In [49]:
```
# Configuration options
blobs_random_seed = 42
centers = [(0,0), (4.5,4)]
cluster_std = 1
frac_test_split = 0.45
num_features_for_samples = 2
num_samples_total = 10000

# Generate data
inputs, targets = make_blobs(n_samples = num_samples_total, centers = centers, n_features = num_features_for_samples, cluster_std = cluster_std)
X_train, X_test, y_train, y_test = train_test_split(inputs, targets, test_size=frac_test_split, random_state=blobs_random_seed)

# Generate scatter plot for training data
plt.scatter(X_train[:,0], X_train[:,1])
plt.title('Data with two classes')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```



In [14]:
```
# Initialize SVM classifier
clf = svm.SVC(kernel='linear')

# Fit data
clf = clf.fit(X_train, y_train)

# Predict the test set
predictions = clf.predict(X_test)
```

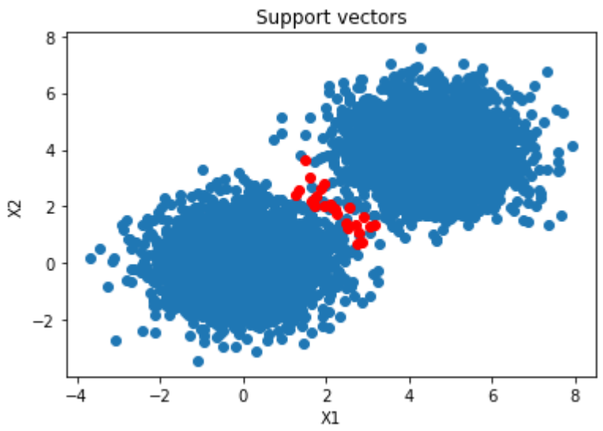
```
# Generate confusion matrix
matrix = plot_confusion_matrix(clf, X_test, y_test,
                               cmap=plt.cm.Blues,
                               normalize='true')
plt.title('Confusion matrix for our classifier')
plt.show(matrix)
plt.show()
```

```
C:\Users\Manish Nadella\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

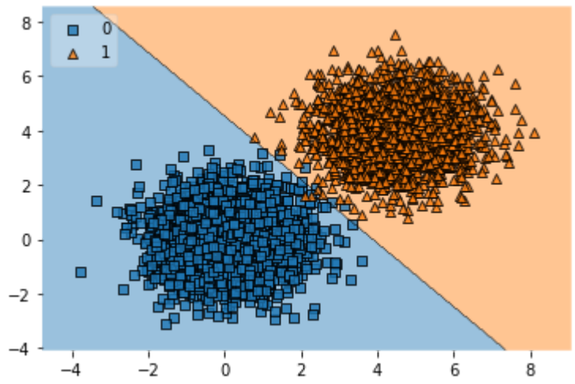


In [50]:

```
# Get support vectors
support_vectors = clf.support_vectors_

# Visualize support vectors
plt.scatter(X_train[:,0], X_train[:,1])
plt.scatter(support_vectors[:,0], support_vectors[:,1], color='red')
plt.title('Support vectors')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```



In [16]:
```
# Plot decision boundary
plot_decision_regions(X_test, y_test, clf=clf, legend=2)
plt.show()
```

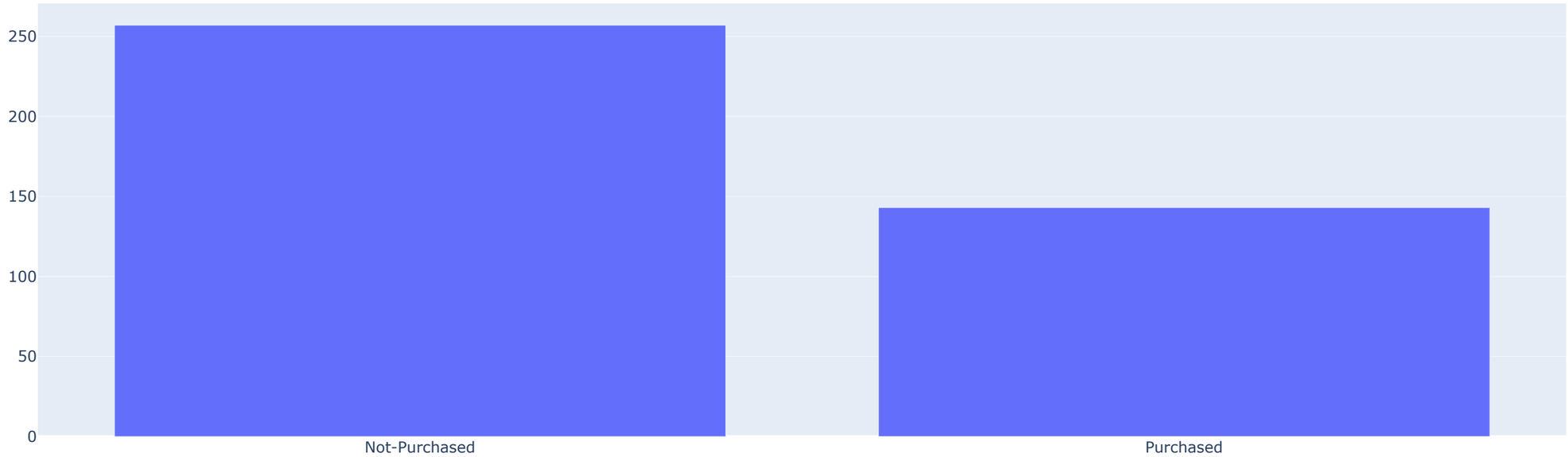


In [17]:
```
dataset = pd.read_csv('C:/Users/Manish Nadella/Downloads/customer_purchases.csv')

# split the data into inputs and outputs
X = dataset.iloc[:, [0,1]].values
y = dataset.iloc[:, 2].values

print(dataset.Purchased)
```

```
0      0
1      0
2      0
3      0
4      0
      ..
395    1
396    1
397    1
398    0
399    1
Name: Purchased, Length: 400, dtype: int64
```

In [18]:
```
import matplotlib.pyplot as plt
import chart_studio.plotly as py
import plotly.graph_objects as go
import plotly.offline as pyoff
import pandas as pd

# importing the dats set
data = pd.read_csv('C:/Users/Manish Nadella/Downloads/customer_purchases.csv')

# counting the total output data from purchased column
target_balance = data['Purchased'].value_counts().reset_index()

# dividing the output classes into two sections
target_class = go.Bar(
    name = 'Target Balance',
    x = ['Not-Purchased', 'Purchased'],
```

```
    y = target_balance['Purchased']
)

# ploting the output classes
fig = go.Figure(target_class)
pyoff.iplot(fig)
```



In [20]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.25, random_state=0)
```

In [31]:
```
from sklearn.preprocessing import StandardScaler

# scalling the input data
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
```

In [42]:
```
from sklearn.svm import SVC

# kernel to be set linear as it is binary class
classifier = SVC(kernel='linear')
```

```
# traininf the model
classifier.fit(X_train, y_train)
```

Out[42]:

```
SVC(kernel='linear')
```

In [43]:

```
y_pred = classifier.predict(X_test)
```

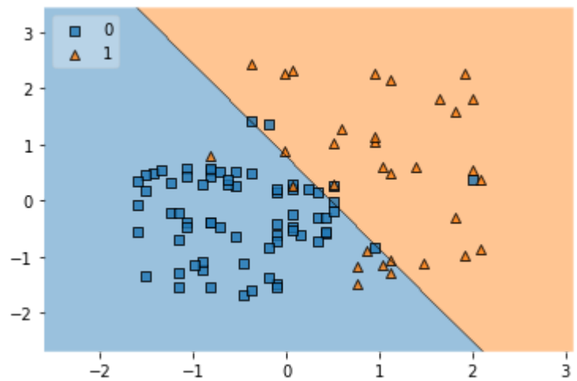
In [44]:

```
from sklearn.metrics import accuracy_score

# printing the accuracy of the model
print(accuracy_score(y_test, y_pred))
```

```
0.88
```

In [45]:

```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X_test, np.array(y_test), clf=classifier, legend=2)
plt.show()
```

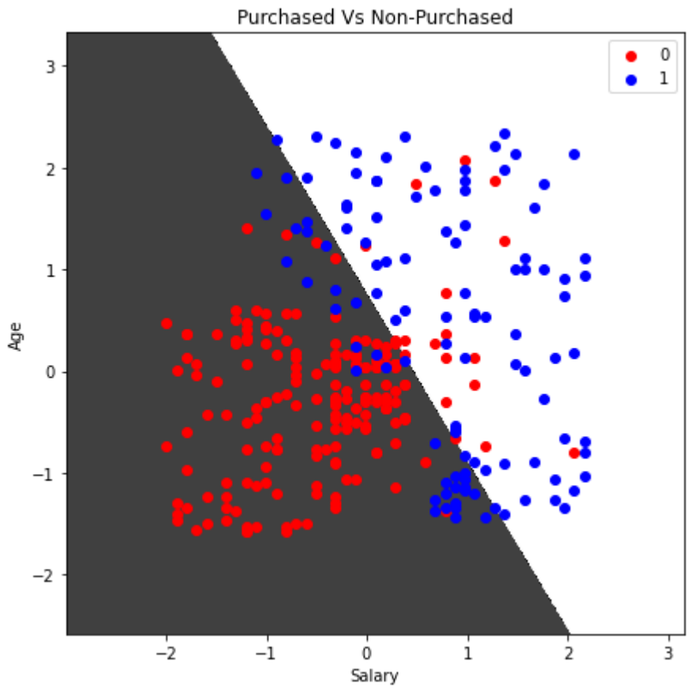


In [46]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# plotting the fgiure
plt.figure(figsize = (7,7))

# assigning the input values
X_set, y_set = X_train, y_train

# ploting the linear graph
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01), np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = ListedColormap(('black', 'white')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

# ploting scattered graph for the values
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'blue'))(i), label = j)
```

```
# labeling the graph
plt.title('Purchased Vs Non-Purchased')
plt.xlabel('Salary')
plt.ylabel('Age')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-ar
gument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-ar
gument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```

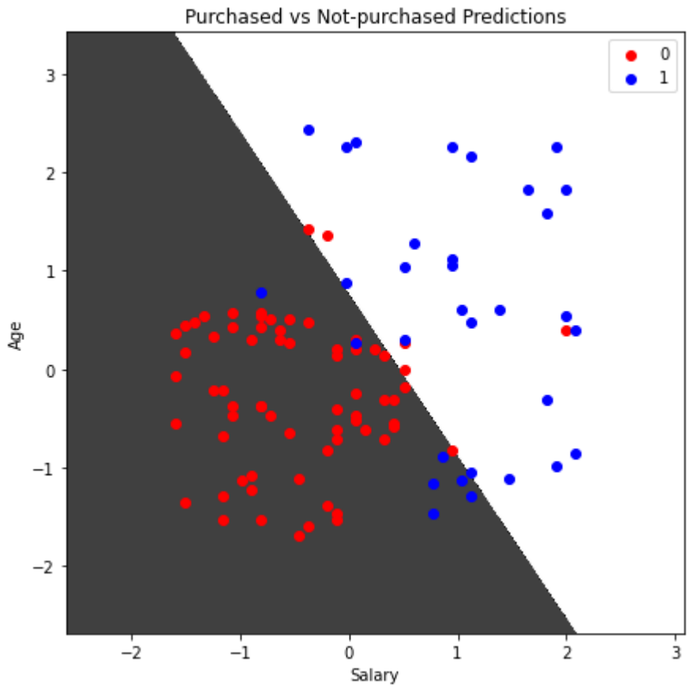


In [14]:

```
plt.figure(figsize = (7,7))

# assigning the testing dataset
X_set, y_set = X_test, y_test

# ploting the predicted graph
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),alpha = 0.75, cmap = ListedColormap(('black', 'white')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

# plorting scattred graph for the testing values
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],c = ListedColormap(('red', 'blue'))(i), label = j)

# labelling the graphe
plt.title('Purchased vs Not-purchased Predictions')
plt.xlabel('Salary')
```

```
plt.ylabel('Age')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-ar
gument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-ar
gument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



In [ ]:

In [56]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
%matplotlib inline
```

In [57]:
```
bankdata = pd.read_csv("C:/Users/Manish Nadella/Downloads/bill_authentication.csv")
```

In [58]:
```
X = bankdata.iloc[:, [0,1]].values
y = bankdata['Class']
```

In [59]:
```
print(y)
```

```
0      0
1      0
2      0
3      0
4      0
      ..
493    1
494    1
495    1
496    1
497    1
Name: Class, Length: 498, dtype: int64
```

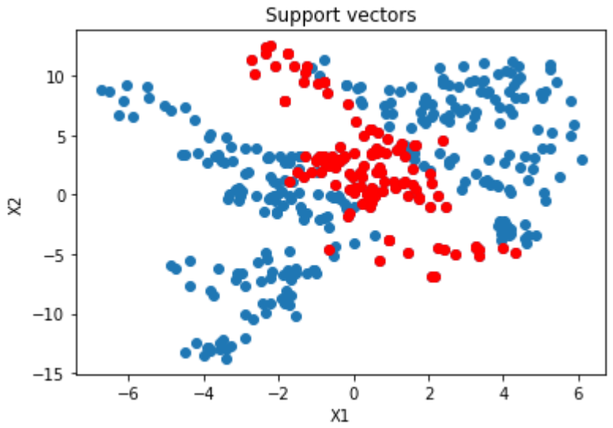
In [67]:
```
from sklearn.model_selection import train_test_split
X_train_data, X_test_data, y_train_data, y_test_data = train_test_split(X, y, test_size = 0.20)
```

In [68]:
```
from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train_data, y_train_data)
```

Out[68]:
```
SVC(kernel='linear')
```

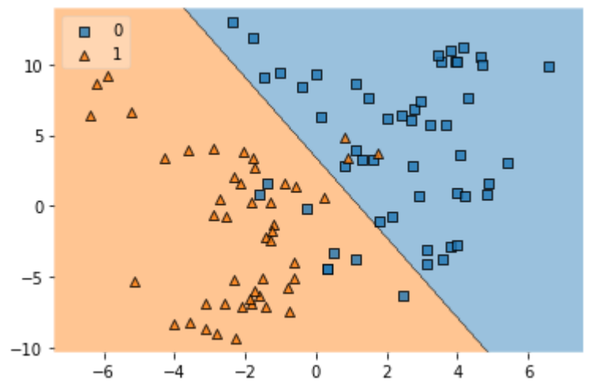
In [69]:
```
y_predct = svclassifier.predict(X_test_data)
```

In [70]:
```
support_vectors = svclassifier.support_vectors_

# Visualize support vectors
plt.scatter(X_train_data[:,0], X_train_data[:,1])
plt.scatter(support_vectors[:,0], support_vectors[:,1], color='red')
plt.title('Support vectors')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```



In [71]:

```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X_test_data, np.array(y_test_data), clf=svclassifier, legend=2)
plt.show()
```



In [72]:

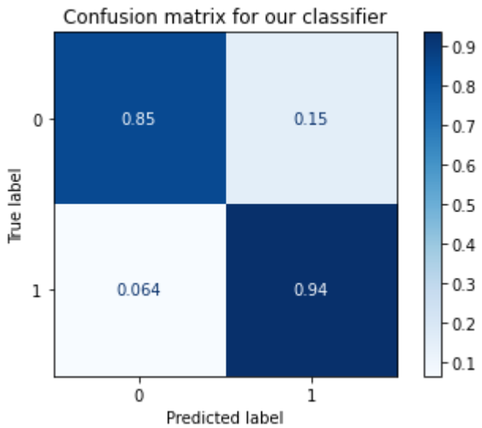
```
from sklearn.metrics import accuracy_score

# printing the accuracy of the model
print(accuracy_score(y_test_data, y_predct))
```

```
0.89
```

In [73]:

```
matrix = plot_confusion_matrix(svclassifier, X_test_data, y_test_data,
                               cmap=plt.cm.Blues,
                               normalize='true')
plt.title('Confusion matrix for our classifier')
plt.show(matrix)
plt.show()
```

```
C:\Users\Manish Nadella\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 an
d will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



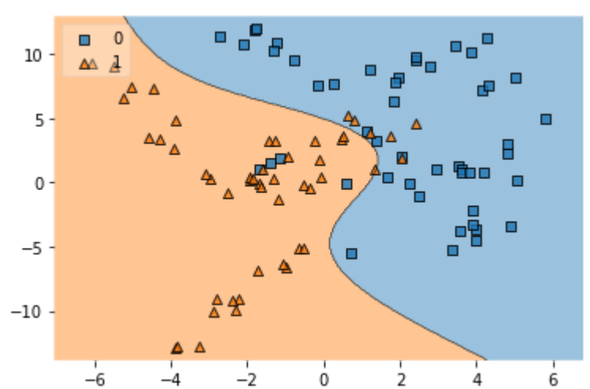
In [74]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

In [75]:
```
from sklearn.svm import SVC
svclassifier = SVC(kernel='rbf')
svclassifier.fit(X_train, y_train)
```

Out[75]:
```
SVC()
```

In [76]:
```
y_pred = svclassifier.predict(X_test)
```

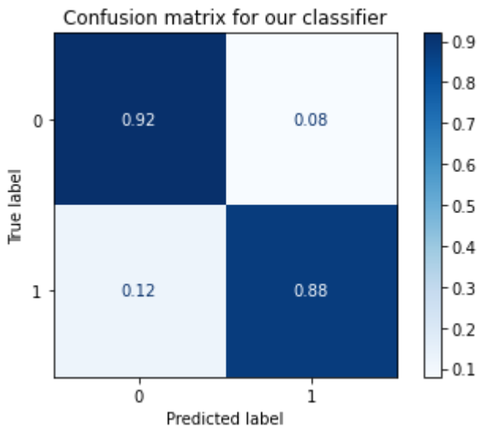
In [79]:
```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X_test, np.array(y_test), clf=svclassifier, legend=2)
plt.show()
```



In [80]:
```
matrix = plot_confusion_matrix(svclassifier, X_test, y_test,
                               cmap=plt.cm.Blues,
```

```
                                normalize='true')
plt.title('Confusion matrix for our classifier')
plt.show(matrix)
plt.show()
```

```
C:\Users\Manish Nadella\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 an
d will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



In [81]:

```
from sklearn.metrics import accuracy_score

# printing the accuracy of the model
print(accuracy_score(y_test, y_pred))
```

```
0.9
```

In [ ]: