DEPT. OF CEN

AMRITA VISHWA VIDHYAPEETHAM

(Amrita School of Engineering)

July 2, 2022

REPORT

Soft Margin SVM

Submitted by

Rachure Charith Sai (CB.EN. U4AIE20055)

Sajja Bala Karthikeya (CB.EN. U4AIE20065)

Menta Sai Akshay (CB.EN. U4AIE20040)

Penaka Vishnu Reddy (CB.EN. U4AIE20048)

Batch 2020-2024

Under the Supervision of

Prof. (Dr.) Sowmya V.

Abstract

This report contains soft margin SVM using linear and nonlinear kernels. And also contains the computation and formulation for different types of linear and non -linear SVM's and usage linear kernel and RBF (Radial Basis Function) Kernel for the classification of the non-linearly separable data (. i.e., Real world datasets).

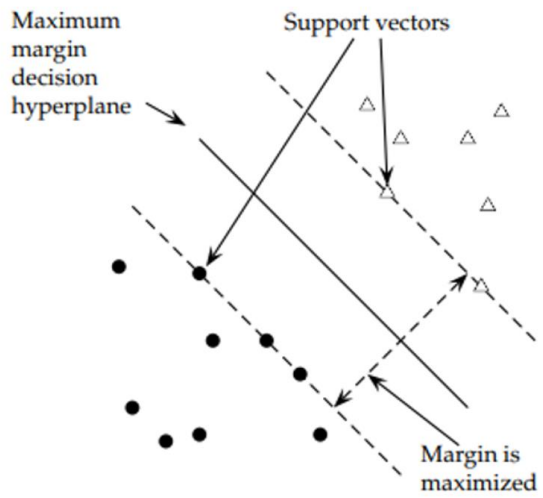
This also contains the computation in MATLAB to observe the working of linear and nonlinear soft margin SVMs. It contains the formulation of soft margin SVM as a constrained optimization problems in Primal, dual and Matrix forms. This also contains the implementation of SVM over breast cancer dataset (Real world data).

INTRODUCTION

Support Vector Machines (SVM)

The support vector machines (SVM) is a relatively recent addition to the toolbox of the AI practitioner. Its development followed in the reverse order of neural networks (NNs) development. SVMs evolved from the sound theory to the implementation and experiments, while the NNs followed a more heuristic path, from applications and extensive experimentation to the theory.

SVM constructs a hyperplane that separates two classes (can be extended for multiclass). A good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin/maximum margin), since in general the larger the margin the generation error of the classifier.



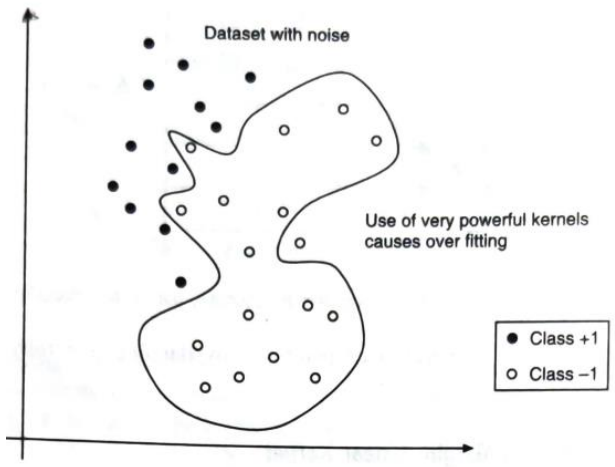
The two planes parallel to the classifier and which pass through one or more points in the data set are called bounding planes. The distance between these bounding planes is called the margin and SVM learning means finding a central hyperplane that maximizes this margin. The points falling on the bounding planes are called support vectors.

Linear SVM vs. Non-Linear SVM

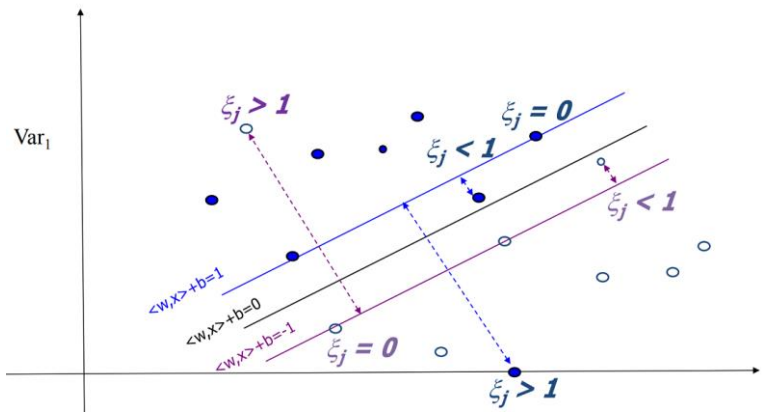
Linear SVM	Non-Linear SVM
It can be easily separated with a linear line	It cannot be easily separated with a linear line
Data is classified with the help of a hyperplane	We use Kernels to make non-separable data into separable data
Data can be easily classified by drawing a straight line.	We map data into high-dimensional space to classify.

Soft Margin SVM

SVMs belong to the class of supervised learning algorithms in which the learning machine is given a set of examples (or inputs) with the associated labels (or output values). SVM constructs a hyperplane that separates two-class (can be extended to multi-class problems). While doing so, the SVM algorithm tries to achieve maximum separation between the classes.



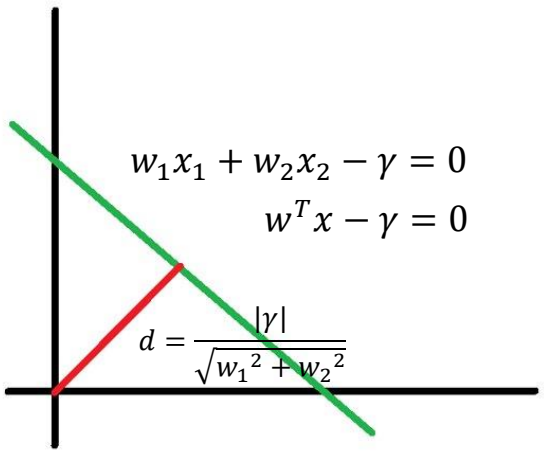
In the above fig., where it is not possible to find a separating plane that completely separates data points in region A_+ and A_-. So, the only possibility way is to find a maximally separating plane with a small number of points that fall on the wrong side. The deviation of such a point from the respective bounding plane is called error. While training, we must try for the maximum margin between the bounding planes and the minimum number of points that contributes to error. So, we introduce a control parameter C through which we control the weight given to these two contradicting requirements.



If all the points are not linearly separable, we allow training error or in other words allow points to lie between the bounding hyperplane and beyond. When a point lies either between the bounding hyperplanes or beyond, $w^T x_i - \gamma \leq -1$ we add a positive quantity ξ_i to the left of the inequality to satisfy the constraint $w^T x_i - \gamma \leq +1$. It now becomes $w^T x_i - \gamma + \xi_i \geq +1$. Similarly, we subtract a positive quantity ξ_i from the left of inequality for negative points that falls between bounding hyperplanes and into the region of positive points. The equation for such a point is $w^T x_i - \gamma - \xi_i \leq -1$.

ALGORITHMS

Deriving Objective Function



W - Matrix consisting weights of all equations

X - Matrix inputs/variable matrix

γ - bias/constant term

d - Represents Perpendicular distance between origin and hyperplane

$$d_U = \frac{|-\gamma-1|}{\sqrt{{w_1}^2+{w_2}^2}}\ (The\ perpendicular\ distance\ of\ the\ bounding\ hyper\ plane$$

$w_1x_1 + w_2x_2 - \gamma = 1\ from\ the\ origin)$

$$d_L = \frac{|-\gamma+1|}{\sqrt{{w_1}^2+{w_2}^2}}(The\ perpendicular\ distance\ of\ the\ bounding\ hyper\ plane$$

$w_1x_1 + w_2x_2 - \gamma = -1\ from\ the\ origin)$

$$Margin = d_U - d_L = \frac{|-\gamma-1+\gamma-1|}{\sqrt{{w_1}^2+{w_2}^2}} = \frac{2}{\sqrt{{w_1}^2+{w_2}^2}}\ (distance\ between\ the\ bounding\ hyper\ planes)$$

Our final point is to find w and γ that maximises this distance,

$$Maximize\frac{2}{\sqrt{{w_1}^2+{w_2}^2}} \equiv Minimize\frac{\sqrt{{w_1}^2+{w_2}^2}}{2} \equiv Minimize\ \frac{{w_1}^2+{w_2}^2}{2}$$

$$Minimize\ \frac{1}{2}w^Tw\ \equiv Minimize\ \frac{1}{2}{||w||}^2$$

$$Subject\ toD[Aw\ -\gamma e] \geq e$$

L1-SVM with Soft Margin: Linear Kernel

$$\min_{w,\gamma,\xi}\ \frac{1}{2}w^Tw + C\sum_{i=1}^{m}\xi_i$$

$$s.t\quad d_i[w^Tx_i - \gamma] + \xi_i \geq 1, \qquad 1 \leq i \leq m$$

$$\xi_i \geq 0, \qquad 1 \leq i \leq m$$

C is scalar value that control the weightage and e^T is sum of non-negative errors $\sum_{i=1}^{m}\xi_i$. Minimisation of the above equation with respect to w and ξ causes maximum separation between bounding planes with minimum number of points crossing their respective bounding planes.

The Lagrangian is

$$L(w,\gamma,\xi,u,\mu) = \frac{1}{2}w^Tw + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}u_i[d_i(w^Tx_i-\gamma)+\xi_i-1] - \sum_{i=1}^{m}\mu_i\xi_i$$

$$= \frac{1}{2}w^Tw + \sum_{i=1}^{m}(C-u_i-\mu_i)\xi_i - \left(\sum_{i=1}^{m}u_id_i{x_i}^T\right)w - \left(\sum_{i=1}^{m}u_id_i\right)\gamma + \sum_{i=1}^{m}u_i$$

Where u, μ are the Lagrange multipliers. The Wolfe of the problem is

$$\max L(w,\gamma,\xi,u,\mu)$$

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

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

$$\frac{\partial L}{\partial \xi_i} = 0, \qquad 1 \leq i \leq m$$

$$u \geq 0$$

$$\mu \geq 0$$

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

$$w^T - \sum_{i=1}^{m}u_id_i{x_i}^T = 0$$

or equivalently,

$$w = \sum_{i=1}^{m}u_id_i\,x_i$$

The Constarint $\frac{\partial L}{\partial \gamma} = 0$ *implies*

$$\sum_{i=1}^{m} u_i d_i = 0$$

The Constarint $\frac{\partial L}{\partial \xi_i} = 0$ *implies*

$$C - u_i - \mu_i = 0, \qquad 1 \le i \le m$$

Note that $u > 0, m > 0,$ *and we have*

$$0 \le u_i \le C$$

(Substituting these results into $L(w, \gamma, \xi, u, \mu)$

$$L(w, \gamma, \xi, u, \mu) = \frac{1}{2} w^T w + \sum_{i=1}^{m} 0 \times \xi_i - w^T w - 0 \times \gamma + \sum_{i=1}^{m} u_i$$

$$= -\frac{1}{2} w^T w + \sum_{i=1}^{m} u_i$$

$$= \sum_{i=1}^{m} u_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} d_i d_j {x_i}^T x_i u_i u_j$$

The Dual Problem is

$$\max \; L_D(\boldsymbol{u}) = \sum_{i=1}^{m} u_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} d_i d_j \boldsymbol{x}_i^T \boldsymbol{x}_j u_i u_j$$

Subject to

$$\sum_{i=1}^{m} d_i u_i = 0$$

$$0 \le u_i \le \boldsymbol{C} \qquad \boldsymbol{1 \le i \le m}$$

or

$$\min \; L_D(\boldsymbol{u}) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} d_i d_j \boldsymbol{x}_i^T \boldsymbol{x}_j u_i u_j - \sum_{i-1}^{m} u_i$$

Subject to

$$\sum_{i=1}^{m} d_i u_i = 0, \qquad 0 \le u_i \le \boldsymbol{C}, \quad \boldsymbol{0 \le i \le m}$$

In matrix form it can be written as follows

$$\min \; L_D(\boldsymbol{u}) = \frac{1}{2} \boldsymbol{u}^T \boldsymbol{Q} \boldsymbol{u} - \boldsymbol{e}^T \boldsymbol{u}$$

$$\boldsymbol{dTu} = \boldsymbol{0}, \; \boldsymbol{0 \le u \le C_e}$$

Here $\boldsymbol{Q} = \boldsymbol{DAA^T D}$ and A is data matrix. Each data is a row vector in matrix A Note that in matrix, Q can be computed as

$$\boldsymbol{Q} = (\boldsymbol{A} * \boldsymbol{A^T}).* (\boldsymbol{d} * \boldsymbol{d^T})$$

$$Weight, \; \boldsymbol{W} = \boldsymbol{A'} * (\boldsymbol{u}.* \boldsymbol{d})$$

The main difference between L-Norm soft margin and hard margin SVM Classifier is that in case of the soft margin, the Lagrangian multiplier $u_i s$ are bounded. That is,

$$0 \le u_i \le \boldsymbol{C}; \quad i = 1,2, \; \ldots, \; m$$

KKT (complementary) conditions have the following three cases for u_i:

1. $u_i = 0$. Then $\xi_i = 1$ (corresponding data points do not require positive ξ_i). Thus $\boldsymbol{x}_i$ (ith data point in the training set) is correctly classified.
2. $0 < u_i < C.\ \ Then\ d_i(\boldsymbol{w}^T\boldsymbol{x}_i - \gamma) + \xi_i = 1\ and\ \xi_i = 0.$ Then $\boldsymbol{x}_i$ is a support vector. Such support vectors with $0 < u_i < C$ are called unbounded support vectors. These are called unbounded support vectors.
3. $u_i = C\ then\ d_i(\boldsymbol{w}^T\boldsymbol{x}_i - \gamma) + \xi_i = 1\ and \xi_i > 0$. Then $\boldsymbol{x}_i$ is support vector. Such support vectors with $u_i = C$ are called 'bounded support vectors'. These are the points falling outside the bounding plane of the respective classes to which the points belong. If $0 \leq \xi_i < 1$, then $\boldsymbol{x}_i$ is correctly classified, and if $\xi_i \geq 1$, it is misclassified.

How to compute γ (gamma) once we obtain u (Lagrangian Multiplier) using Quadratic Programming?

For computing γ, one of those points whose Lagrangian multiplier falls between 0 and C can used, that is one of those points which falls on the hyper planes. For these points $\boldsymbol{w}^T\boldsymbol{x}_i - \gamma = d_i$.

And therefore

$\gamma = \boldsymbol{w}^T\boldsymbol{x}_i - d_i = \sum_{j \in svindex} u_j d_j \boldsymbol{x}_j^T \boldsymbol{x}_i - d_i$. It is customary to get γ from all such points and find average. Here svindex correspond to the set of indices of support vectors (include those points falling on the hyperplanes.

Let svmindex correspond to the set of indices of support vectors that falls on the hyperplanes. Then

$$\gamma = \frac{1}{|svmindex|}\left[\sum_{j \in svindex}\left(\sum_{j \in svindex} u_j d_j \boldsymbol{x}_j^T \boldsymbol{x}_i\right) - d_i\right]$$

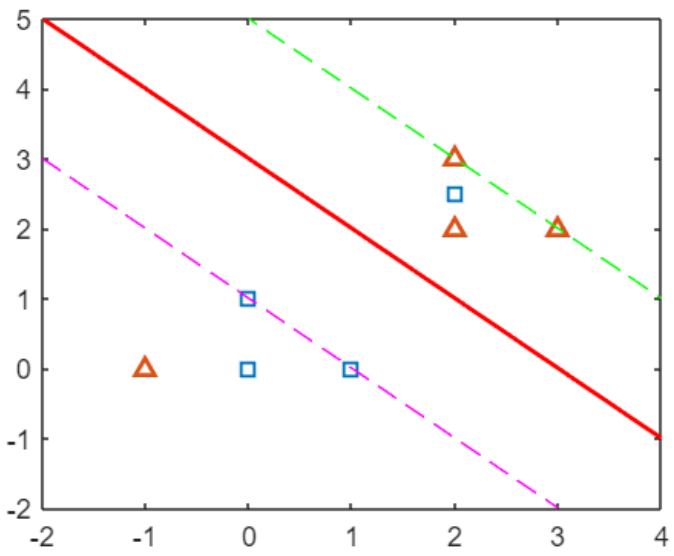
$$\gamma = \frac{1}{|svmindex|}\left[\sum_{j \in svindex}\left(\sum_{j \in svindex} d_i u_j \underbrace{d_i d_j \boldsymbol{x}_j^T \boldsymbol{x}_i}_{\substack{elements\ of \\ \boldsymbol{Q}\ Matrix}}\right) - d_i\right] \quad (since\ d_i d_i = 1)$$

Implementation in MATLAB

```
X=[1 0 0 2 2 2 3 -1;0 1 0 2.5 2 3 2 0]'; % training data or input data
d=[-1 -1 -1 -1 1 1 1 1]'; % target
e=ones(length(d),1);
C = 1;

cvx_begin quiet
variable w(2);
variable gama;
variable psii(8); % psii as a column vector
minimize (0.5*(w'*w)+C*sum(psii));
subject to
d.*(X*w-gama*e)>=1-psii;
psii>=0;
cvx_end
w1 = w(1)
w2 = w(2)
gama
plot(X(1:4,1), X(1:4,2),'s', LineWidth=1.5);% plot-1 points
hold on
plot(X(5:8,1), X(5:8,2),'^', LineWidth=1.5);% plot+1 points
hold on
x1=-2:5;
x2=-(w(1)/w(2))*x1+(gama/w(2));
plot(x1,x2,LineWidth=1.5,Color='r'); % draw the classifier line
hold on
x2=-(w(1)/w(2))*x1+((-1+gama)/w(2));
plot(x1,x2,'m--') % draw the lower bounding line
hold on
x2=-(w(1)/w(2))*x1+((1+gama)/w(2));
plot(x1,x2, 'g--') % draw the upper bounding line
axis([-2 4 -2 5])
```

```
w1 = 0.5000
w2 = 0.5000
gama = 1.5000
```

output



L2 Norm Linear SVM

In L2 Norm SVM, the sum of squares of error (slack) variable are minimized along with reciprocal of the square of the margin between the bounding planes. The formulation of problem is given by

$$\min_{w,\gamma,\xi} \frac{1}{2} w^T w + \frac{C}{2} \sum_{i=1}^{m} \xi_1^2$$

$$subject\ to \qquad d_i(w^T x_i - \gamma) + \xi_i - 1 \geq 0,$$

$$1 \leq i \leq m, \quad \xi_i \geq 0, \ 1 \leq i \leq m$$

The Lagrangian is

$$\boldsymbol{L}(\boldsymbol{w},\gamma,\xi,\boldsymbol{u}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \frac{\boldsymbol{C}}{2}\sum_{i=1}^{m}\xi_1^2 - \sum_{i=1}^{m} u_i\,[d_i(\boldsymbol{w}^T\boldsymbol{x}_i - \gamma) + \xi_i - 1]$$

$$= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \frac{\boldsymbol{C}}{2}\sum_{i=1}^{m}\xi_1^2 - \left(\sum_{i=1}^{m} u_i\, d_i \boldsymbol{x}_i^T\right)\boldsymbol{w} - \left(\sum_{i=1}^{m} u_i\, d_i\right)\gamma - \sum_{i=1}^{m} u_i\,\xi_i + \sum_{i} u_i$$

Where u are the Lagrange multipliers.

Note that in Lagrangian, The Lagrangian multipliers corresponding to the constraint $\xi_i \geq 0, \ 1 \leq i \leq m$ have not been considered. This is because ξ_i becomes a function of u_i alone and all u_is are unbounded (on positive side) as derived in the following:

The Wolfe dual of the problem is

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

Subject to

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

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

$$\frac{\partial L}{\partial \xi_i} = 0 \qquad 1 \leq i \leq m$$

$$\boldsymbol{u} \geq \boldsymbol{0}$$

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

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

Or equivalently

$$\boldsymbol{w} = \sum_{i=1}^{m} u_i d_i \boldsymbol{x}_i$$

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

$$\sum_{i=1}^{m} u_i d_i = 0$$

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

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

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

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

$$\boldsymbol{L}(\boldsymbol{w}, \boldsymbol{\gamma}, \boldsymbol{\xi}, \boldsymbol{u}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \frac{1}{2C}\sum_{i=1}^{m} u_i^2 - \boldsymbol{w}^T\boldsymbol{w} - \boldsymbol{0} * \gamma - \frac{1}{C}\sum_{i=1}^{m} u_i^2 + \sum_{i=1}^{m} u_i$$

$$= -\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \sum_{i=1}^{m} u_i - \frac{1}{2C}\sum_{i=1}^{m} u_i^2$$

$$= \sum_{i=1}^{m} u_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{i=1}^{m} d_i d_j \boldsymbol{x}_i^T \boldsymbol{x}_j u_i u_j - \frac{1}{2C}\sum_{i=1}^{m} u_i^2$$

multiplying numerator and denominator with $u_i u_j d_i d_j$

$$= \sum_{i=1}^{m} u_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{i=1}^{m} d_i d_j u_i u_j \, (\boldsymbol{x}_i^T \boldsymbol{x}_j + \frac{\delta_{ij}}{\boldsymbol{C}})$$

$$where\ \delta_{ij} = \frac{ui^2}{u_i u_j d_i d_j}$$

To summarize, the dual problem is

$$\max_{\boldsymbol{u}} \boldsymbol{L}(\boldsymbol{u}) = \sum_{i=1}^{m} u_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{i=1}^{m} d_i d_j \boldsymbol{x}_i^T \boldsymbol{x}_j u_i u_j \, (\boldsymbol{x}_i^T \boldsymbol{x}_j + \frac{\delta_{ij}}{\boldsymbol{C}})$$

Subject to

$$\sum_{i=1}^{m} d_i u_i = 0$$

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

The standard form in matrix format is

$$\min_{\boldsymbol{u}} \boldsymbol{L}(\boldsymbol{u}) = \frac{1}{2}\boldsymbol{u}^T \boldsymbol{D}\left(\boldsymbol{A}\boldsymbol{A}^T + \frac{\boldsymbol{I}}{\boldsymbol{C}}\right)\boldsymbol{D}\boldsymbol{u} - \boldsymbol{e}^T\boldsymbol{u}$$

Subject to

$$\boldsymbol{d}^T\boldsymbol{u} = 0, u \ge 0$$

Or

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

Subject to

$$\boldsymbol{d}^T\boldsymbol{u} = 0, u \geq 0$$

Q can be computed as $\boldsymbol{Q} = (\boldsymbol{A} * \boldsymbol{A}^T + \boldsymbol{I}/\boldsymbol{C}).*(\boldsymbol{d} * \boldsymbol{d}^T)$

How to compute γ ?

According to the KKT complementarity theorem, at optimal point:

$$u_i(d_i(\boldsymbol{w}^T\boldsymbol{x}_i - \gamma) + \boldsymbol{\xi}_i - 1) = 0$$

$$So\ either\ u_i = 0\ or\ (d_i(\boldsymbol{w}^T\boldsymbol{x}_i - \gamma) + \boldsymbol{\xi}_i - 1) = 0$$

$$that\ is,\ if\ u_i > 0,\ then\ (d_i(\boldsymbol{w}^T\boldsymbol{x}_i - \gamma) + \boldsymbol{\xi}_i - 1) = 0$$

Substituting the optimal values of $\boldsymbol{w}\ and \boldsymbol{\xi}_i$, we obtain

$$d_i\left(\sum_{j=1}^{m} u_j d_j x_j^T x_i + \frac{d_i}{C}u_i - \gamma\right) - 1 = 0; \quad [because\ d_i d_i = +1]$$

$$d_i\left(\sum_{j=1}^{m} u_j d_j \left(x_j^T x_i + \frac{\delta_{ij}}{C}\right) - \gamma\right) - 1 = 0$$

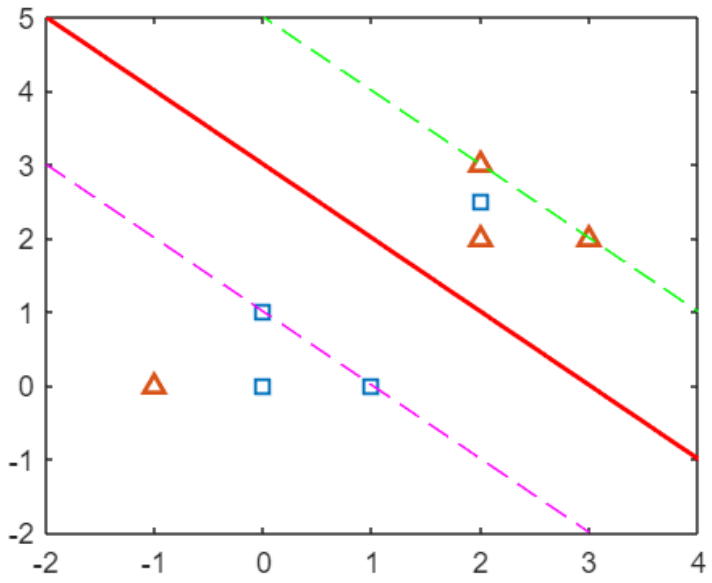
$$\sum_{j=1}^{m} u_j d_j \left(x_j^T x_i + \frac{\delta_{ij}}{C}\right) - \gamma = \frac{1}{d_i} = d_i$$

$$\gamma = \sum_{j=1}^{m} u_j d_j \left(x_j^T x_i + \frac{\delta_{ij}}{C}\right) - d_i$$

Implementation in MATLAB

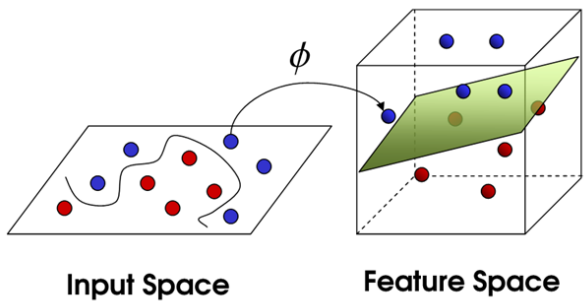
```
X=[1 0 0 2 2 2 3 -1;0 1 0 2.5 2 3 2 0]';
d=[-1 -1 -1 -1 1 1 1 1]';
e=ones(length(d),1);
C = 1;
cvx_begin quiet
variable w(2);
variable gama;
variable psii(8); % psii as a column vector
minimize (0.5*(w'*w)+0.5*C*sum(psii)^2);
subject to
d.*(X*w-gama*e)>=1-psii;
psii>=0;
cvx_end
w1 = w(1)
w2 = w(2)
gama
plot(X(1:4,1), X(1:4,2),'s', LineWidth=1.5);% plot-1 points
hold on
plot(X(5:8,1), X(5:8,2),'^', LineWidth=1.5);% plot+1 points
hold on
x1=-2:5;
x2=-(w(1)/w(2))*x1+(gama/w(2));
plot(x1,x2,LineWidth=1.5,Color='r'); % draw the classifier line
hold on
x2=-(w(1)/w(2))*x1+((-1+gama)/w(2));
plot(x1,x2,'m--') % draw the lower bounding line
hold on
x2=-(w(1)/w(2))*x1+((1+gama)/w(2));
plot(x1,x2, 'g--') % draw the upper bounding line
axis([-2 4 -2 5])
```

output



Non-Linear SVM

Linear SVM is undoubtedly better to classify data if it is trained by linearly separable data. Linear SVM also can be used for non-linearly separable data, provided that the number of such instances is less. However, in real-life applications, the number of data overlapping is so high that soft margin SVM cannot cope to yield an accurate classifier. As an alternative to this there is a need to compute a decision boundary, which is not linear (i.e., not a hyperplane but rather a hypersurface).



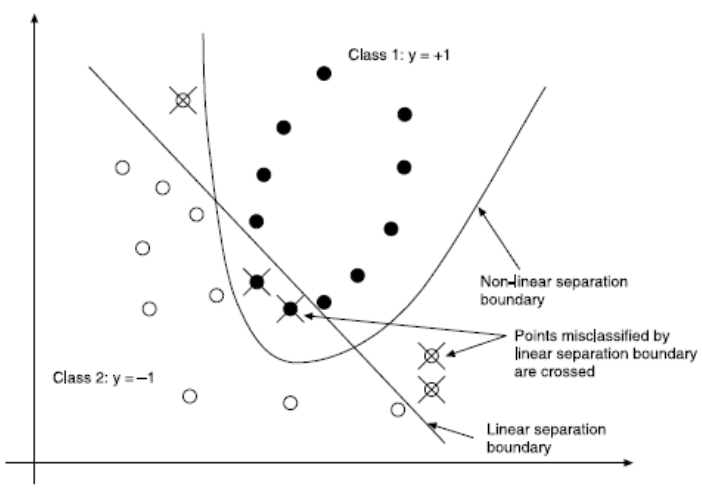
A hyperplane is expressed as

$$\boldsymbol{Linear}: w_1x_1 + w_2x_2 + w_3x_3 + c = 0$$

$$\boldsymbol{Non-Linear}: w_1x_1^2 + w_2x_2^2 + w_3x_1x_2 + w_4x_3^2 + w_5x_1x_3 + c = 0$$

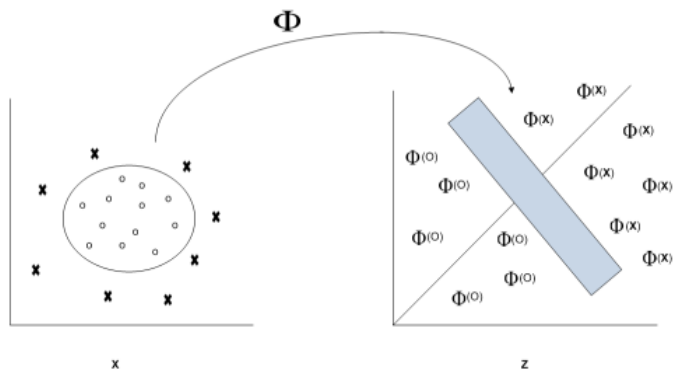
The task, therefore, takes a turn to find nonlinear decision boundaries, that is, nonlinear hypersurface in input space comprising linearly not separable data. This task can be achieved by extending the formulation of linear SVM, which we have already learned. We can do it in two major steps

1) Transform the original(non-linear) input data into a higher-dimensional space (as a linear representation of data). It is feasible because SVM's performance is decided by the number of support vectors, not by the dimension of data.
2) Search for the linear decision boundaries to separate the transformed higher-dimensional data.

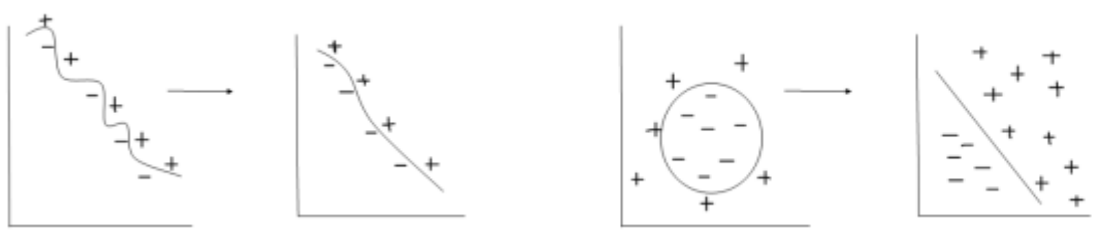


Kernel trick

From the idea that training data that are not linearly separable, can be transformed into a higher dimensional feature space such that in higher dimensional transformed space a hyperplane can be decide to separate the transformed data and hence original data.



- This kernel function K (X_i, X_j) physically implies the similarity in the transformed space (i.e., nonlinear similarity measure) using the original attribute X_i, X_j.In other words, K, the similarity function computes a similarity of both whether data in original attribute space or transformed attribute space.
- There are different kernel functions that follow different parameters. Those parameters are called magic parameters and are to be decided prior. In general, polynomial kernels result in large dot products. Gaussian RBF produces more support vectors than other kernels.



Polynomial Kernel RBF Kernel

- Merce's Theorem: A Kernel function K can be expressed as K (X, Y) = $\phi(X)^T . \phi(Y)$,if and only if , for any function g(x) such that $\int g(x)^2 . dx$ is finite then

$$\int K(X,Y)g(x)g(y)dxdy \geq 0$$

RBF Kernel

The radial basis function kernel is given by,

$$K(x,y) = \exp(-\sigma\|x-y\|^2)$$

where, σ is a positive parameter controlling the radius.

Expanding $\exp(-\sigma\|x-y\|^2)$, we obtain

$$\exp(-\sigma\|x-y\|^2) = \exp(-\sigma\|x\|^2)\exp(-\sigma\|y\|^2)\exp(2\sigma x^T y)$$

Since, $\exp(2\sigma x^T y) = 1 + 2\sigma x^T y + \frac{(2\sigma)^2}{2!}(x^T y)^2 + \frac{(2\sigma)^3}{3!}(x^T y)^3 + \cdots.,$

$$\exp(2\sigma x^T y) \textit{ is a infinite summation of polynomials.}$$

Soft Margin SVM of Non-Linear Kernel (L1 Norm with Bias Term)

The formulation can be written as

$$\min_{w,\gamma,\xi} \frac{1}{2} w^T w + C\sum_{i=1}^{m} \xi_i$$

Subject to

$$d_i(w^T\phi(x_i) - \gamma) + \xi_i - 1 \geq 0$$

$$1 \leq \mathrm{i} \leq m$$

$$\xi_i \geq 0$$

$$1 \leq \mathrm{i} \leq m$$

The Lagrangian is

$$L(w,\gamma,\xi,u,\mu) = \frac{1}{2}w^T w + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}u_i[d_i(w^T\phi(x_i)-\gamma)+\xi_i-1] - \sum_{i=1}^{m}\mu_i\xi_i$$

$$= \frac{1}{2}w^T w + \sum_{i=1}^{m}(C-u_i-\mu_i)\xi_i - \left(\sum_{i=1}^{m}u_i d_i \phi(x_i^T)\right)w - \left(\sum_{i=1}^{m}u_i d_i\right)\gamma + \sum_{i=1}^{m}u_i$$

Where u, μ are the Lagrange multipliers.

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

Subject to

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

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

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

$$1 \leq \mathrm{i} \leq m$$

$$\mu \geq 0$$

$$u \geq 0$$

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

$$w^T - \sum_{i=1}^{m}u_i d_i \phi(x_i^T) = 0$$

Or equivalently,

$$w = \sum_{i=1}^{m}u_i d_i \phi(x_i)$$

The constarint $\frac{\partial L}{\partial \gamma} = 0$ *implies*

$$\sum_{i=1}^{m}u_i d_i = 0$$

The constarint $\frac{\partial L}{\partial \xi_i} = 0$ *imples*

$$C - u_i - \mu_i = 0$$

$$1 \leq \mathrm{i} \leq m$$

Note that $u \geq 0, \mu \geq 0,$ *and we have*

$$0 \leq u_i \leq C$$

Substituting these results into $L(w,.\gamma,\xi,u,\mu)$

$$L(w,.\gamma,\xi,u,\mu) = \frac{1}{2}w^T w + \sum_{i=1}^{m}0\times\xi_i - w^T w - 0\times\gamma + \sum_{i=1}^{m}u_i$$

$$= -\frac{1}{2}w^T w + \sum_{i=1}^{m} u_i$$

$$= \sum_{i=1}^{m} u_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} d_i d_j \emptyset(x_i^T)\emptyset(x_j)u_i u_j$$

The dual problem is

$$\max_u L(u) = \sum_{i=1}^{m} u_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} d_i d_j \phi(x_i^T)\phi(x_j)u_i u_j$$

Subject to

$$\sum_{i=1}^{m} d_i u_i = 0$$

$$1 \le \mathrm{i} \le m$$

$$0 \le u_i \le C$$

The above dual problem can be converted into a minimization problem

by changing the sign of objective function.

Then in Matrix form, formulation become

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

$$Q = K.*(d * d^T)$$

$$d^T u = 0, 0 \le u \le Ce$$

$$Here\ \ Q = DKD\ and\ K\ is\ Kernel\ matrix$$

Soft Margin SVM of Non-Linear Kernel (L2 Norm with Bias Term)

For linear SVM, the dual formulation is

$$\min_u L(u) = \frac{1}{2}u^T D\left(AA^T + \frac{1}{C}\right)Du - e^T u$$

Subject to

$$d^T u = 0$$

$$u \ge 0$$

So for a non − linear kernel, the dual formulation is

$$\min_u L(u) = \frac{1}{2}u^T D\left(K + \frac{1}{C}\right)Du - e^T u$$

Subject to

$$d^T u = 0$$

$$u \ge 0$$

Or

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

$$Q = \left(k + \frac{I}{C}\right).*(d * d^T)$$

Subject to

$$d^T u = 0$$

$$u \geq 0$$

Soft Margin SVM of Non-Linear Kernels (L1 Norm without Bias Term)

if no bias term is assumed, then the decision function is given by

$$\boldsymbol{w}^T \phi(\boldsymbol{X}_i) \; = \; 0$$

The bounding planes are then $\boldsymbol{w}^T \phi(\boldsymbol{X}_i) \; = +1$ for positive samples and $\boldsymbol{w}^T \phi(\boldsymbol{X}_i) \; = -1$ for negative samples.

The SVM formulation is therefore written as

$$\underset{w,\gamma,\xi}{min} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{m} \boldsymbol{\xi}_i$$

subject to

$$d_i(\boldsymbol{w}^T \phi(\boldsymbol{X}_i)) + \boldsymbol{\xi}_i - 1 \geq 0, \;\; 1 \leq i \leq m$$

$$\boldsymbol{\xi}_i \geq 0, \;\; 1 \leq i \leq m$$

The Lagrangian is

$$L(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{u}, \boldsymbol{\mu}) \; = \; \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \boldsymbol{C}\sum_{i=1}^{m} \boldsymbol{\xi}_i - \sum_{i=1}^{m} u_i \left[d_i\left(\boldsymbol{w}^T \phi(\boldsymbol{X}_i)\right) + \boldsymbol{\xi}_i - 1\right] - \sum_{i=1}^{m} \boldsymbol{\mu}_i \boldsymbol{\xi}_i$$

$$= \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \sum_{i=1}^{m} (C - u_i - \mu_i)\boldsymbol{\xi}_i - \left(\sum_{i=1}^{m} u_i d_i \phi(\boldsymbol{X}_i^T)\right)\boldsymbol{w} + \sum_{i=1}^{m} u_i$$

where $\boldsymbol{u}$, μ *are the Lagrange multipliers, the Wolfe dual problem is*

$$\underset{u,\mu}{max} \boldsymbol{L}(\boldsymbol{w}, \boldsymbol{\xi}, \boldsymbol{u}, \boldsymbol{\mu})$$

subject to

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

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{\xi}_i} = 0 \qquad 1 \leq i \leq m$$

$$\boldsymbol{u} \geq 0$$

$$\boldsymbol{\mu} \geq 0$$

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

$$\boldsymbol{w}^T - \sum_{i=1}^{m} u_i d_i \phi(\boldsymbol{X}_i^T) = 0$$

or equivalently,

$$w = \sum_{i=1}^{m} u_i d_i \phi(\boldsymbol{X}_i)$$

The constraint $\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{\xi}_i} = 0$ *imply*

$$\boldsymbol{C} - u_i - \mu_i = 0, \;\; 1 \leq i \leq m.$$

Note that $\boldsymbol{u} \geq 0$, $\boldsymbol{\mu} \geq 0$, *and we have*: $\;\; 0 \leq u_i \leq \boldsymbol{C}$

Substituting these results into $L(w, \xi, u, \mu)$:

$$\boldsymbol{L(w,\xi,u,\mu)} = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \sum_{i=1}^{m} 0 \times \boldsymbol{\xi}_i - \boldsymbol{w}^T\boldsymbol{w} + \sum_{j=1}^{m} \boldsymbol{u}_i$$

$$= -\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \sum_{i=1}^{m} \boldsymbol{u}_i = \sum_{i=1}^{m} \boldsymbol{u}_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \boldsymbol{d}_i\boldsymbol{d}_j\boldsymbol{\phi}(\boldsymbol{X}_i^T)\boldsymbol{\phi}(\boldsymbol{X}_j)\boldsymbol{u}_i\boldsymbol{u}_j$$

The dual of the problem is:

$$\max_{\boldsymbol{u}} \boldsymbol{L(u)} = \sum_{i=1}^{m} \boldsymbol{u}_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \boldsymbol{d}_i\boldsymbol{d}_j\boldsymbol{\phi}(\boldsymbol{X}_i^T)\boldsymbol{\phi}(\boldsymbol{X}_j)\boldsymbol{u}_i\boldsymbol{u}_j$$

subject to

$$0 \le u_i \le \boldsymbol{C},\ 1 \le i \le m$$

This is equivalent to the following matrix formulation.

$$Minimise\ \boldsymbol{L_D(u)} = \frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} - \boldsymbol{e}^T\boldsymbol{u}$$

$$\boldsymbol{0 \le u \le Ce}$$

$$Here\ \boldsymbol{Q = DKD}\ and\ \boldsymbol{K}\ is\ the\ kernel\ matrix.$$

$$\boldsymbol{Q}\ can\ be\ computed\ as\ \boldsymbol{Q = K}.\ *(\boldsymbol{d}\ *\boldsymbol{d}^T).$$

Soft Margin SVM Non-Linear Kernel (L2 Norm without Bias Term)

for a non-linear kernel, the dual formulation is

$$\min_{\boldsymbol{u}} \boldsymbol{L(u)} = \frac{1}{2}\boldsymbol{u}^T\boldsymbol{D}\left(\boldsymbol{K} + \frac{\boldsymbol{I}}{\boldsymbol{C}}\right)\boldsymbol{D}\boldsymbol{u} - \boldsymbol{e}^T\boldsymbol{u}$$

subject to

$$\boldsymbol{u} \ge 0$$

or

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

subject to

$$\boldsymbol{u} \ge 0$$

Where,

$$Q = \left(K + \frac{I}{C}\right).*(d * d^T)$$

Example for Non-Linear SVM

Let A=$\begin{bmatrix}1\\2\\5\\6\end{bmatrix}$ $d = \begin{bmatrix}-1\\-1\\1\\-1\end{bmatrix}$ $K(x_i, x_j) = ({x_i}^T x_j + 1)^2\ and\ C = 50$

***case* 1**: ***L*1 *norm*, *with bias term* γ**

The SVM formulation is given by

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

$$d^T u = 0, 0 \le u \le Ce$$

Here $Q = DKD$ *and* K *is the kernel matrix*

$Q\ is\ computed\ as\ Q = K.*\ (d * d^T)$

$$Q = \begin{bmatrix} 4 & 9 & -36 & 49 \\ 9 & 25 & -121 & 169 \\ -36 & -121 & 676 & -961 \\ 49 & 169 & -961 & 1369 \end{bmatrix}$$

$on\ solving, we\ obtain$

$$u_1 = 0, u_2 = 2.5, u_3 = 7.333333, u_4 = 4.833333$$

$$f(x) = sign\left(\sum_i^4\ d_i\ u_i\ K(x_i\ , x) - \gamma\right) = sign\left(\sum_i^4\ d_i\ u_i\ ({x_i}^T\ x + 1)^2 - \gamma\right)$$

$$= sign((-1)(2.5)(2x+1)^2 + \ (1)(7.33333)(5x+1)^2 + (-1)(4.833333)(6x+1)^2 - \gamma)$$

$$= sign(-0.666667x^2 + 5.333333x - \gamma)$$

$Bias\ \gamma\ is\ determined\ from\ the\ requirment\ that\ at\ the\ SV\ points\ 2{,}5\ and\ 6, the\ outputs\ must\ be$

$-1{,}1\ and\ -1\ respectively. Hence\ \gamma = 9, resulting\ in\ the\ decision\ function$

Case-2: Working without a bias term b solution is obtained as

$u_1 = 0, u_2 = 25, u_3 = 43.333333, u_4 = 27.333333$

$$f(x)\ =\ sign\left(\sum_{i=1}^4\ d_i u_i (x_i x + 1)^2\right)$$

$$=\ sign((-1)(25)(2x+1)^2 + (1)(43.333333)(5x+1)^2 + (-1)(27.333333)(6x+1)^2 - \gamma)$$

$$=\ sign(-0.666667x^2 + 5.333333x - 9)$$

Thus the non-linear (quadratic)decision function and consequently the indicator function in these two particular cases are equal.

L2 Norm with and without Bias Term - Non Linear SVM

objective function in Matrix form

$$\min_u L_D(u) = \frac{1}{2} u^T\ D\left(K + \frac{1}{C}\right) D\ u - e^T u$$

(or)

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

subject to

$$d^T u = 0, \qquad u \geq 0$$

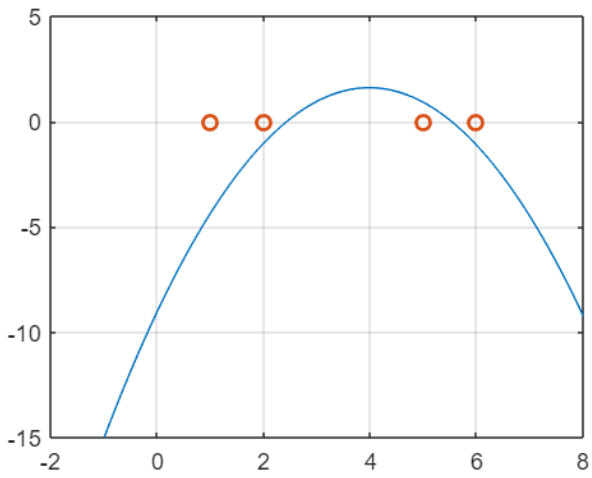
$$Q = \left(K + \frac{1}{C}\right).*(d * d^T); \quad \text{where } K \text{ is kernel matrix}$$

Implementation in MATLAB with Bias

```
A=[1 2 5 6]';    % 1-D point
d=[-1 -1 1 -1]';
e = ones(length(A),1);
k = (A*A' +1).^2; % kernel matrix
C=50; % weight
Q = (k + 1/C).*(d*d');
cvx_begin quiet
variable u(4)
minimize (0.5*u'*Q*u-e'*u)
% 4*1 * 4*4 * 1*4
subject to
d'*u==0;
u>=0;
cvx_end


u1 = round(u(1));
u2 = u(2);
u3 = u(3);
```

```
u4 = u(4);
x = -1:0.01:8;
y = -0.67*x.^2+5.33*x-9;
plot(x,y)
hold on
plot(A,zeros(4,1),'o', LineWidth=1.5)
grid on
```

Output



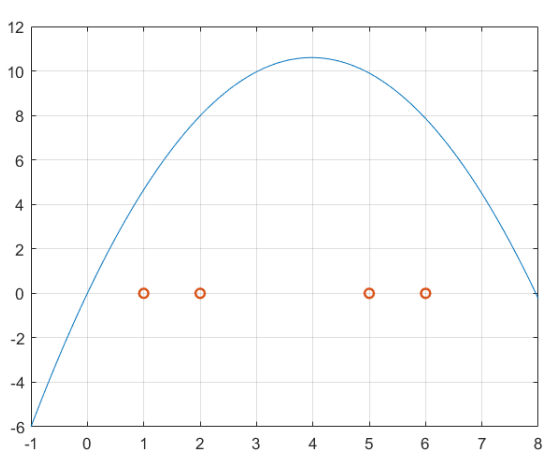
Implementation in MATLAB without Bias

```
A=[1 2 5 6]';    % 1-D point
d=[-1 -1 1 -1]';
e = ones(length(A),1);
k = (A*A' +1).^2; % kernel matrix
C=50; % weight
Q = (k + 1/C).*(d*d');


cvx_begin quiet
variable u(4)
minimize (0.5*u'*Q*u-e'*u)
% 4*1 * 4*4 * 1*4
subject to
d'*u==0;
u>=0;
cvx_end


u1 = round(u(1));
u2 = u(2);
u3 = u(3);
u4 = u(4);
```

$$f(x) = \text{sign}\left[\sum_{i=1}^{4} d_i u_i \left(x_i^T x + 1\right)^2\right]$$
$$= \text{sign}[(-1)(2.5)(2x+1)^2 + (1)(7.3333)(5x+1)^2 + (-1)(4.8333)(6x+1)^2]$$
$$= \text{sign}[-0.666667x^2 + 5.333333x]$$

```
x = -1:0.01:8;
y = -0.67*x.^2+5.33*x;
plot(x,y)
hold on
plot(A,zeros(4,1),'o', LineWidth=1.5)
grid on
```

Output



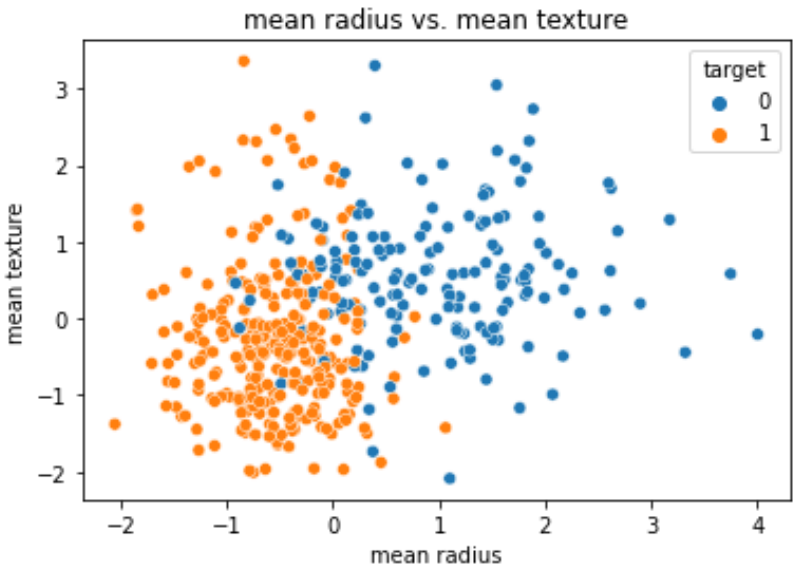
Implementation of Linear and Non-Linear Kernel over Breast Cancer Dataset

Dataset

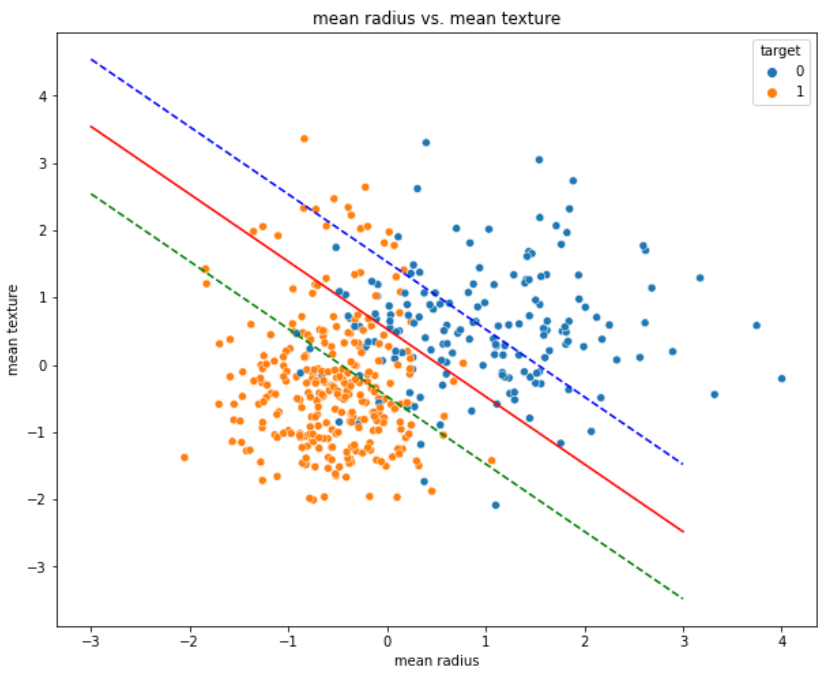
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0

For simplicity we take first two features of dataset and apply SVM over them.

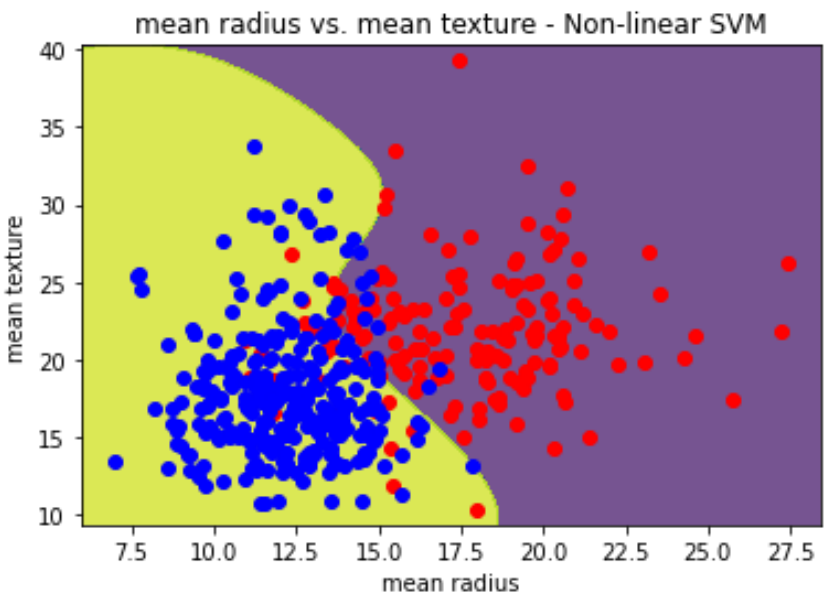
Plot for mean radius and mean texture



Applying Linear Kernel



Applying Non-Linear Kernel



Inference

First objective we found a maximum-margin classifier, in terms of hyper-plane(the vectors w and b) that separates the positive and negative instances. The data set is noisy (with some overlap in positive and negative samples), there will be some error in classifying them with the hyper-plane.

And we minimized the errors in the classification along with maximizing the margin and the problem becomes a soft-margin SVM problem. And Latter slack variables per training point is introduced to include the classification errors (miss-classified data points in the training set) in the objective. Since, it is easier to solve the optimization problem in the dual rather than the primal space. Hence the optimization is often solved in the dual space by converting the minimization to a maximization (keeping in mind the weak/strong duality theorem and the complementary slackness conditions), by constructing the Lagrangian and then using the KKT conditions for a saddle point.

Since our data set is not linearly separable, the kernel trick is used to conceptually map the data points to some higher dimensions only by computing the kernel matrix. We implemented the RBF kernel function to classify. Finally the dual and primal variables are used in conjunction to predict the class of a new data point.

References

1. *Machine Learning with SVM and other kernel methods - chapter 4.*
2. *Note on writing constraints with soft margin from first principle and SVM classifier training as LP problem.*

APPENDIX

Soft Margin SVM On Breast Cancer Data

Imports

In [1]:
```
import pandas as pd
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.svm import SVC
import plotly.graph_objects as go
import plotly.offline as pyoff
```

Breast Cancer Dataset

In [2]:
```
bcancer = load_breast_cancer()
df = pd.DataFrame(bcancer.data, columns=bcancer.feature_names)
df.head()
```

Out[2]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678

5 rows × 30 columns

In [3]:
```
df['target'] = bcancer.target
df.head()
```

Out[3]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902	0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758	0

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300	0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678	0

5 rows × 31 columns

In [4]:
```
bcancer.target_names
```

Out[4]:
```
array(['malignant', 'benign'], dtype='<U9')
```

In [5]:
```
df0 = df[df.target==0]
df0.head()
```

Out[5]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902	0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758	0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300	0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678	0

5 rows × 31 columns

In [6]:
```
df1 = df[df.target==1]
df1.head()
```

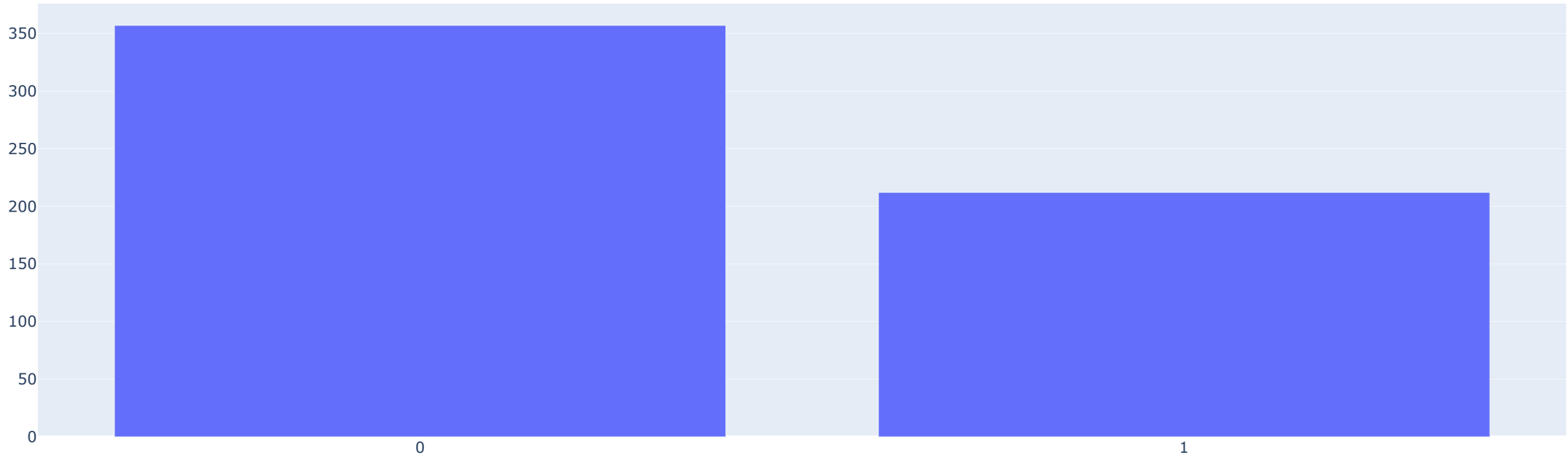
Out[6]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
19	13.540	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.047810	0.1885	0.05766	...	19.26	99.70	711.2	0.14400	0.17730	0.23900	0.12880	0.2977	0.07259	1
20	13.080	15.71	85.63	520.0	0.10750	0.12700	0.04568	0.031100	0.1967	0.06811	...	20.49	96.09	630.5	0.13120	0.27760	0.18900	0.07283	0.3184	0.08183	1
21	9.504	12.44	60.34	273.9	0.10240	0.06492	0.02956	0.020760	0.1815	0.06905	...	15.66	65.13	314.9	0.13240	0.11480	0.08867	0.06227	0.2450	0.07773	1
37	13.030	18.42	82.61	523.8	0.08983	0.03766	0.02562	0.029230	0.1467	0.05863	...	22.81	84.46	545.9	0.09701	0.04619	0.04833	0.05013	0.1987	0.06169	1
46	8.196	16.84	51.71	201.9	0.08600	0.05943	0.01588	0.005917	0.1769	0.06503	...	21.96	57.26	242.2	0.12970	0.13570	0.06880	0.02564	0.3105	0.07409	1

5 rows × 31 columns

In [7]:
```
target_balance = df['target'].value_counts().reset_index()
target_class = go.Bar(
    #name = 'Target Balance',
```

```
    x = ['0', '1'],
    y = target_balance['target']
)

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



In [8]:
```
bcancer.feature_names
```

Out[8]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

In [9]:
```
X = df.drop(['target'], axis='columns')
y = df.target
```

Training Dataset

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234)
```

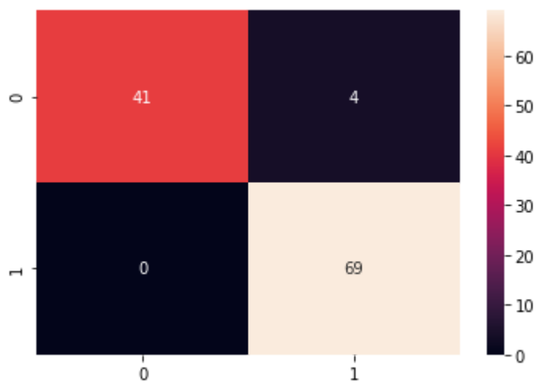
In [11]:
```
from sklearn.preprocessing import StandardScaler

sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
```

In [12]:
```
svc_model = SVC(kernel='linear', random_state=32)
svc_model.fit(X_train, y_train)
y_pred = svc_model.predict(X_test)

# importing accuracy score
from sklearn.metrics import accuracy_score

# printing the accuracy of the model
print(f'Accuracy of predicted data = {accuracy_score(y_test, y_pred)*100}')
```

```
Accuracy of predicted data = 96.49122807017544
```

In [13]:
```
# importing the required modules
import seaborn as sns
from sklearn.metrics import confusion_matrix

# passing actual and predicted values
cm = confusion_matrix(y_test, y_pred, labels=svc_model.classes_)

# true Write data values in each cell of the matrix
sns.heatmap(cm, annot=True)
```

Out[13]:
```
<AxesSubplot:>
```



In [14]:
```
# importing classification report
from sklearn.metrics import classification_report

# printing the report
print(classification_report(y_test, y_pred))
```

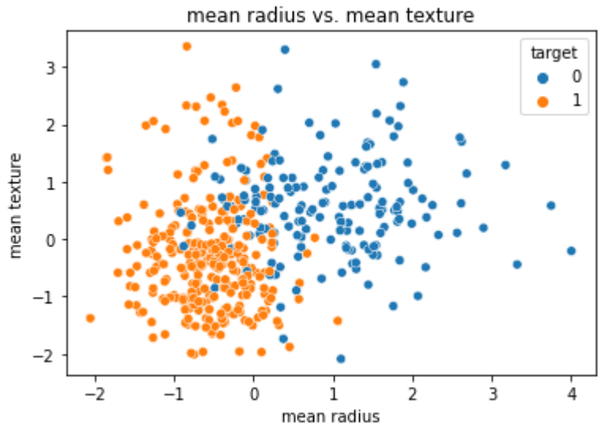
```
              precision    recall  f1-score   support

           0       1.00      0.91      0.95        45
           1       0.95      1.00      0.97        69

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

Plotting for mean radius and mean texture

In [15]:

```
sns.scatterplot(X_train[:, 0], X_train[:, 1], hue=y_train)
plt.xlabel('mean radius')
plt.ylabel('mean texture')
plt.title('mean radius vs. mean texture')
plt.show()
```

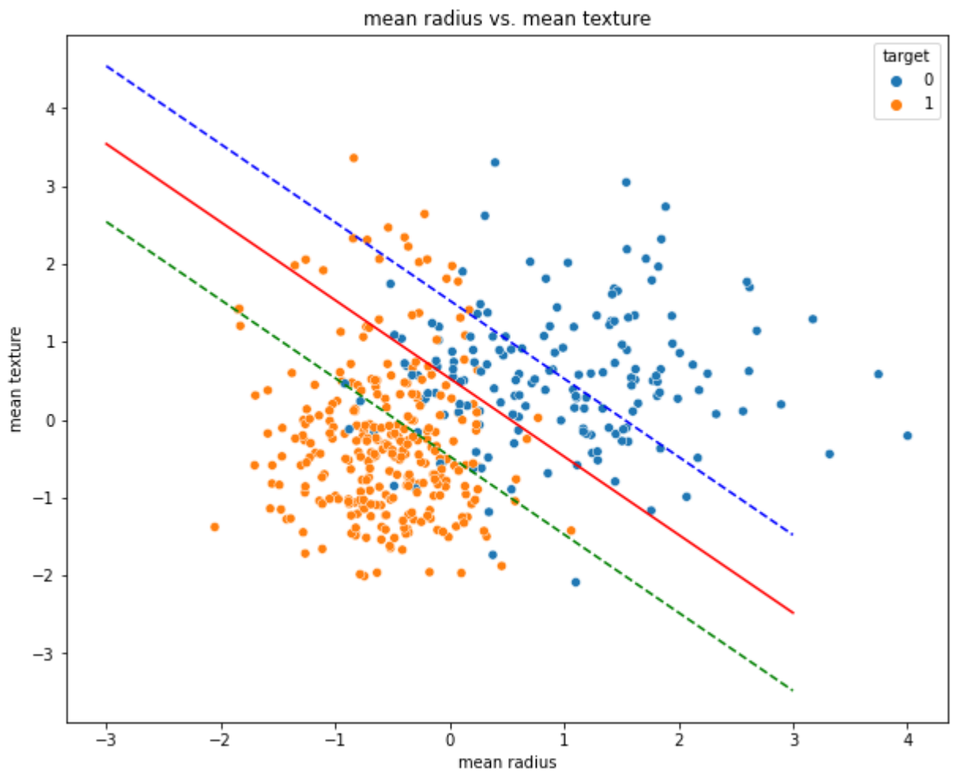
```
C:\Users\LENOVO\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:

Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
```



In [16]:

```
plt.figure(figsize=(10,8))
sns.scatterplot(X_train[:, 0], X_train[:, 1], hue=y_train)
w = svc_model.coef_[0]           # w consists of 2 elements
b = svc_model.intercept_[0]      # b consists of 1 element
x_points = np.linspace(-3, 3)    # generating x-points from -1 to 1
y_points = -(w[0] / w[1]) * x_points - b / w[1]  # getting corresponding y-points
y_points_below = (-(w[0] / w[1]) * x_points - b / w[1]) - 1
y_points_above = (-(w[0] / w[1]) * x_points - b / w[1]) + 1
# Plotting a red hyperplane
plt.plot(x_points, y_points, c='r', label=f'y ={w[0]}x1  + {w[1]}x2 {b}')
plt.plot(x_points, y_points_above, 'b--', label=f"1 ={w[0]}x1  + {w[1]}x2  {b}")
plt.plot(x_points, y_points_below, 'g--', label=f"-1 ={w[0]}x1  + {w[1]}x2  {b}")
plt.xlabel('mean radius')
plt.ylabel('mean texture')
plt.title('mean radius vs. mean texture')
plt.show()
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:

Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
```

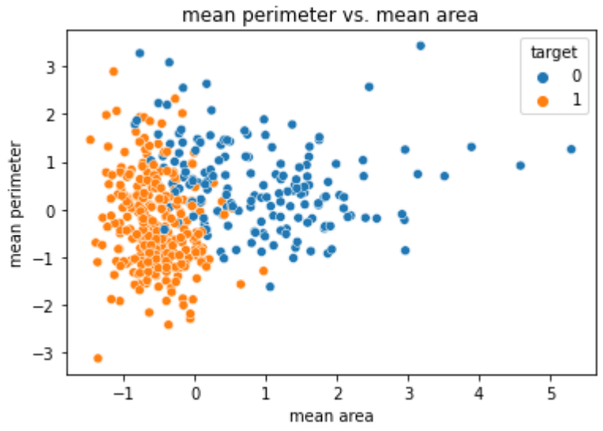


Mean area vs. Mean Perimeter

In [45]:

```
sns.scatterplot(X_train[:, 3], X_train[:, 4], hue=y_train)
plt.xlabel('mean area')
plt.ylabel('mean perimeter')
plt.title('mean perimeter vs. mean area')
plt.show()
```

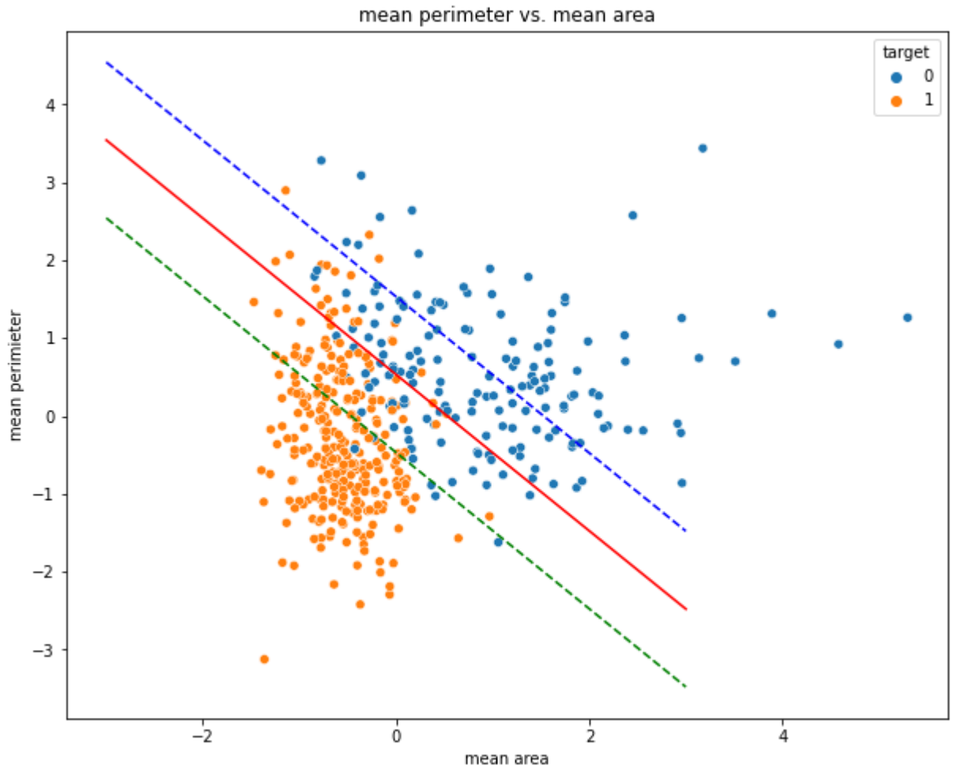
```
C:\Users\LENOVO\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:

Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
```



In [46]:

```
plt.figure(figsize=(10,8))
sns.scatterplot(X_train[:, 3], X_train[:, 4], hue=y_train)
w = svc_model.coef_[0]           # w consists of 2 elements
b = svc_model.intercept_[0]      # b consists of 1 element
x_points = np.linspace(-3, 3)    # generating x-points from -1 to 1
y_points = -(w[0] / w[1]) * x_points - b / w[1]  # getting corresponding y-points
y_points_below = (-(w[0] / w[1]) * x_points - b / w[1]) - 1
y_points_above = (-(w[0] / w[1]) * x_points - b / w[1]) + 1
# Plotting a red hyperplane
plt.plot(x_points, y_points, c='r', label=f'y ={w[0]}x1  + {w[1]}x2 {b}')
plt.plot(x_points, y_points_above, 'b--', label=f"1 ={w[0]}x1  + {w[1]}x2  {b}")
plt.plot(x_points, y_points_below, 'g--', label=f"-1 ={w[0]}x1  + {w[1]}x2  {b}")
plt.xlabel('mean area')
plt.ylabel('mean perimieter')
plt.title('mean perimeter vs. mean area')
plt.show()
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning:

Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
```



In [89]:
```
new_x = df.iloc[:, 0:2].values
new_y = df.iloc[:, -1].values

new_X_train, new_X_test, new_y_train, new_y_test = train_test_split(new_x, new_y, test_size=0.2, random_state=123)
```

In [81]:
```
classifier1 = SVC(kernel='rbf')

# traininf the model
classifier1.fit(new_X_train, new_y_train)
```

Out[81]:
```
SVC()
```

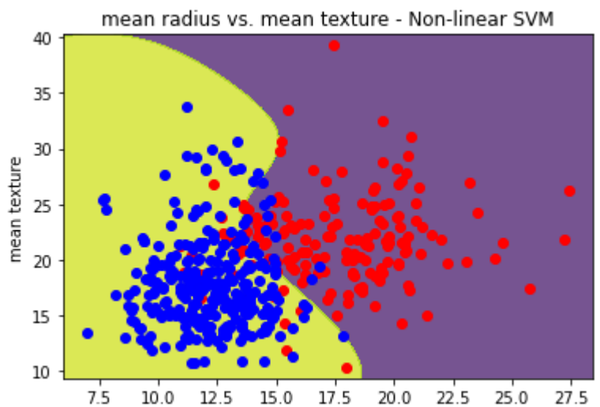
In [84]:
```
new_X_train[:,0].min()
from matplotlib.colors import ListedColormap
```

In [87]:
```
X_set, y_set = new_X_train, new_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, classifier1.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha = 0.75)
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)

plt.title('mean radius vs. mean texture - Non-linear SVM')
plt.xlabel('mean radius')
plt.ylabel('mean texture')
```

```
*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 2-D 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 2-D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```

Out[87]:
```
Text(0, 0.5, 'mean texture')
```



In [ ]:

One vs One, One vs Rest with SVM for multi-class classification

In multiclass classification problems, we need to deal with more than two classes which means the algorithm which we are using should be capable of working with multiple classes. There are various models available for this and some methods are also available that can make support vector machines capable of dealing with more than two classes. We call these methods heuristic methods. There are two types of heuristic methods:

1. One-vs-Rest (OvR) or One-vs-All(OvA)
2. One-vs-One (OvO)

In [11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [39]:
```
iris_data = datasets.load_iris()

X = iris_data.data[:, [0, 1]]
y = iris_data.target

iris_dataframe = pd.DataFrame(iris_data.data[:, [0, 1]],
                  columns=iris_data.feature_names[0:2])

print(iris_dataframe.head())

print('\n' + 'Unique Labels contained in this data are '
     + str(np.unique(y)))
```

```
   sepal length (cm)  sepal width (cm)
0                5.1               3.5
1                4.9               3.0
2                4.7               3.2
3                4.6               3.1
4                5.0               3.6

Unique Labels contained in this data are [0 1 2]
```

In [40]:
```
iris_data.target_names
```

Out[40]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

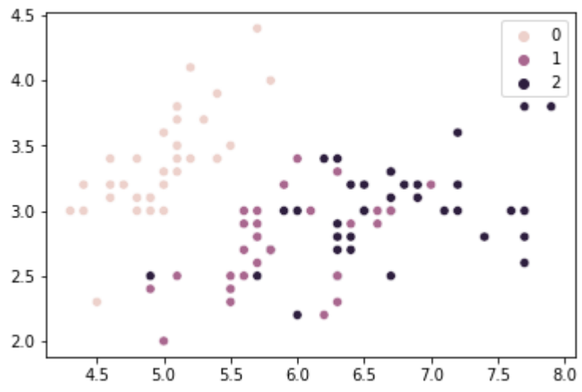
In [41]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
print('The training set contains {} samples and the test set contains {} samples'.format(X_train.shape[0], X_test.shape[0]))
```

```
The training set contains 105 samples and the test set contains 45 samples
```

In [42]:
```
import seaborn as sns
sns.scatterplot(X_train[:, 0], X_train[:, 1], hue=y_train)
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `d
ata`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
<AxesSubplot:>
```

Out[42]:



In [43]:

```
standard_scaler = StandardScaler()
#DataFlair
standard_scaler.fit(X_train)
X_train_standard = standard_scaler.transform(X_train)
X_test_standard = standard_scaler.transform(X_test)
print('The first five rows after standardisation look like this:\n')
print(pd.DataFrame(X_train_standard, columns=iris_dataframe.columns).head())
```

```
The first five rows after standardisation look like this:

   sepal length (cm)  sepal width (cm)
0          -1.023664         -2.378463
1           0.695175         -0.101903
2           0.924353          0.581065
3           0.122229         -1.923151
4           0.924353         -1.240183
```

In [44]:

```
SVM = SVC(kernel='rbf', random_state=0, gamma=.10, C=1.0)
SVM.fit(X_train_standard, y_train)
print('Accuracy of our SVM model on the training data is {:.2f} out of 1'.format(SVM.score(X_train_standard, y_train)))
print('Accuracy of our SVM model on the test data is {:.2f} out of 1'.format(SVM.score(X_test_standard, y_test)))
```

```
Accuracy of our SVM model on the training data is 0.81 out of 1
Accuracy of our SVM model on the test data is 0.80 out of 1
```

One Vs. Rest Plot

The procedure of conversion of the data can be understood using an example of iris data where we have three classes as follows:

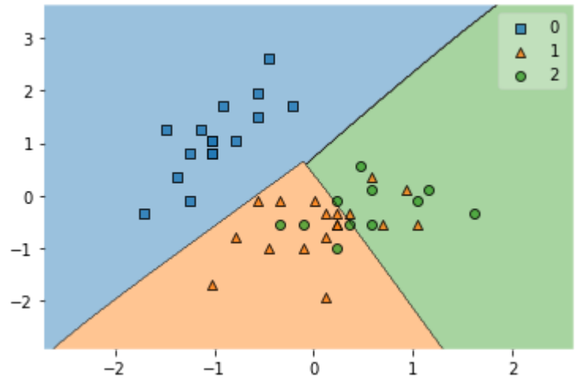
1. Setosa
2. Versicolor
3. Virginica

The converted data as binary classification data will look like the following:

1. Setosa vs [Versicolor, Virginica]
2. Versicolor vs [Setosa, Virginica]
3. Virginica vs [Setosa, Versicolor]

By looking at the conversion we can think that there is a requirement of three models but with the large datasets creating three models can be a tough and non-accurate approach to modeling. Here One-vs-Rest (OvR) or One-vs-All(OvA) comes to save us where binary classifiers can be trained to predict any class as positive and other classes as negative.

In [45]:
```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X_test_standard, np.array(y_test), clf=SVM)
plt.show()
```



One Vs. One Plot

This method can also be considered as an approach to making binary classification algorithms capable of working as multiclass classification algorithms.it is similar to the One-Vs-Rest method because it also works based on splitting the data but the splitting behavior of this method is different from the One-Vs-Rest method. This method includes the split of data for each class where each class has every other class as its opponent. We can again use the iris data set to understand the data split behavior of this method.

In the iris, we have the following classes:

1. Setosa
2. Versicolor
3. Virginica

Split iris data using the One-vs-One (OvO) methods will look like the following:

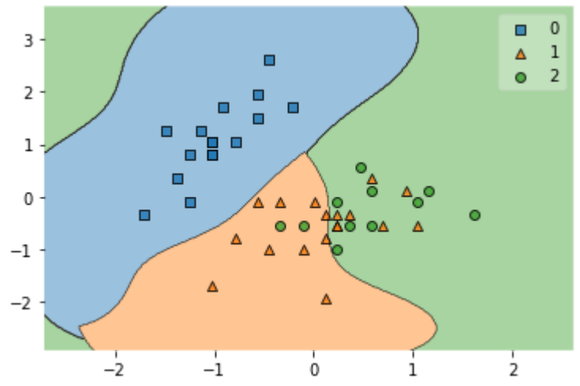
1. Setosa vs Versicolor
2. Setosa vs Virginica
3. Versicolor vs Virginica
4. Virginica vs Versicolor

Here we can see using this data we have binary classification data that includes one class with every other class. We can find out the number of data split using the following formula

Split of data = (number of classes X (number of classes – 1))/2

In [46]:
```
rbf = SVC(kernel='rbf', gamma=1, C=1, decision_function_shape='ovo').fit(X_train_standard, y_train)
```

In [47]:
```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X_test_standard, np.array(y_test), clf=rbf)
plt.show()
```



In [ ]:

GUI For Breast Cancer Classification Using Soft Margin SVM

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import streamlit as st
# import plotly.graph_objects as go
# import plotly.offline as pyoff
from matplotlib.colors import ListedColormap
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

st.write('<h1 style="text-align:center;color :white">Soft Margin SVM</h1>',
         unsafe_allow_html=True)
st.write('<h1 style="text-align:center;color :white">Team 06</h1>',
         unsafe_allow_html=True)


bcancer = load_breast_cancer()
df = pd.DataFrame(bcancer.data, columns=bcancer.feature_names)
df['target'] = bcancer.target
model = st.sidebar.radio('Choose Algorithm', ('Linear SVM', 'Non Linear SVM'))

if model == 'Linear SVM':

    X = df.drop(['target'], axis='columns')
    y = df.target

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=1234)

    sc_X = StandardScaler()
    X_train = sc_X.fit_transform(X_train)
    X_test = sc_X.fit_transform(X_test)

    features = ('mean radius', 'mean texture', 'mean perimeter',
                'mean area', 'mean smoothness', 'mean compactness')

    feature1 = st.sidebar.selectbox('Choose Feature 1', features)
    feature2 = st.sidebar.selectbox('Choose Feature 2', features, index=1)

    svc_model = SVC(kernel='linear', random_state=32)
    svc_model.fit(X_train, y_train)
    st.set_option('deprecation.showPyplotGlobalUse', False)
    plt.figure(figsize=(10, 8))
    sns.scatterplot(X_train[:, features.index(feature1)],
                    X_train[:, features.index(feature2)], hue=y_train)
    w = svc_model.coef_[0]           # w consists of 2 elements
    b = svc_model.intercept_[0]      # b consists of 1 element
    x_points = np.linspace(-3, 3)    # generating x-points from -1 to 1
    y_points = -(w[0] / w[1]) * x_points - b / \
        w[1]  # getting corresponding y-points
    y_points_below = (-(w[0] / w[1]) * x_points - b / w[1]) - 1
```

```
    y_points_above = (-(w[0] / w[1]) * x_points - b / w[1]) + 1
# Plotting a red hyperplane
    plt.plot(x_points, y_points, c='r', label=f'y ={w[0]}x1  + {w[1]}x2 {b}')
    plt.plot(x_points, y_points_above, 'b--',
             label=f"1 ={w[0]}x1  + {w[1]}x2  {b}")
    plt.plot(x_points, y_points_below, 'g--',
             label=f"-1 ={w[0]}x1  + {w[1]}x2  {b}")
    plt.xlabel(f'{feature1}')
    plt.ylabel(f'{feature2}')
    plt.title(f'{feature1} vs. {feature2}')
    plt.show()
    st.pyplot()
elif model == 'Non Linear SVM':
    features = ('mean radius', 'mean texture', 'mean perimeter',
                'mean area', 'mean smoothness', 'mean compactness')
    feature1 = st.sidebar.selectbox('Choose Feature 1', features)
    feature2 = st.sidebar.selectbox('Choose Feature 2', features, index=5)

    new_x = df.iloc[:, [features.index(
        feature1), features.index(feature2)]].values
    new_y = df.iloc[:, -1].values

    new_X_train, new_X_test, new_y_train, new_y_test = train_test_split(
        new_x, new_y, test_size=0.2, random_state=123)

    classifier1 = SVC(kernel='rbf')

    classifier1.fit(new_X_train, new_y_train)
    X_set, y_set = new_X_train, new_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, classifier1.predict(
        np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha=0.75)
    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)

    plt.xlabel(f'{feature1}')
    plt.ylabel(f'{feature2}')
    plt.title(f'{feature1} vs. {feature2}')
    plt.show()
    st.pyplot()
```

Output

Linear SVM Classifier

Choose Algorithm

- (•) Linear SVM
- ( ) Non Linear SVM

Choose Feature 1

mean radius

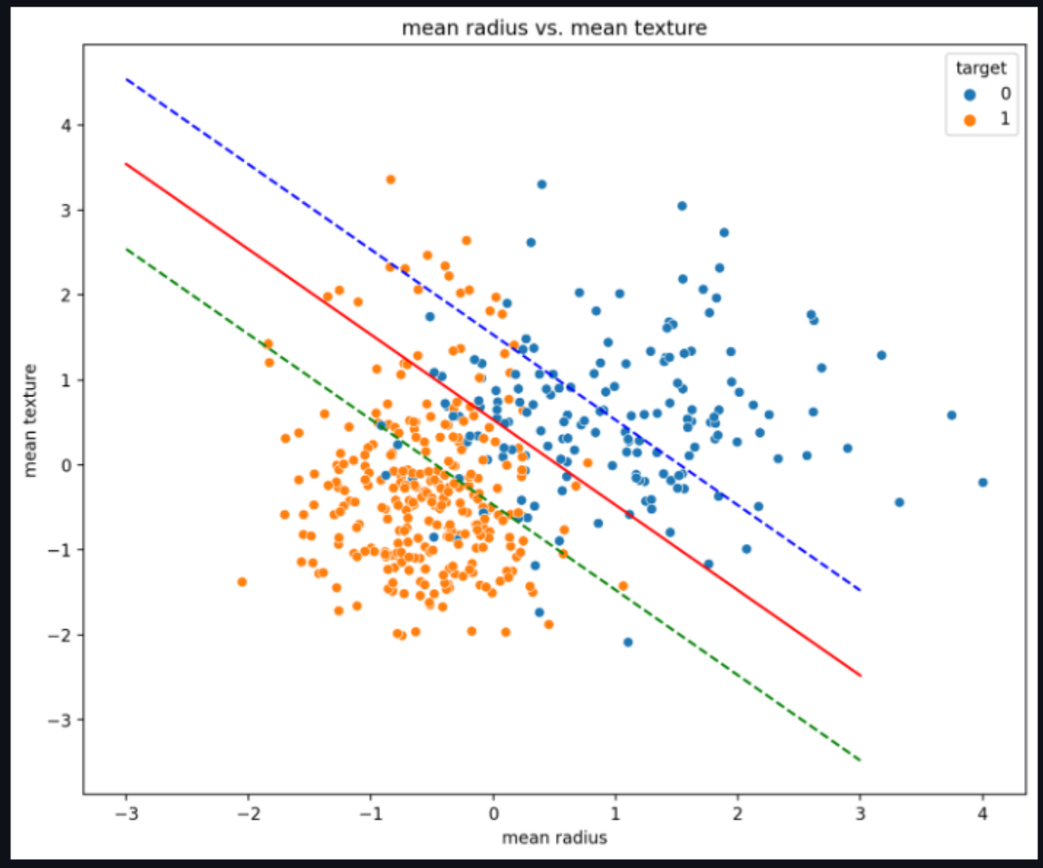
Choose Feature 2

mean texture

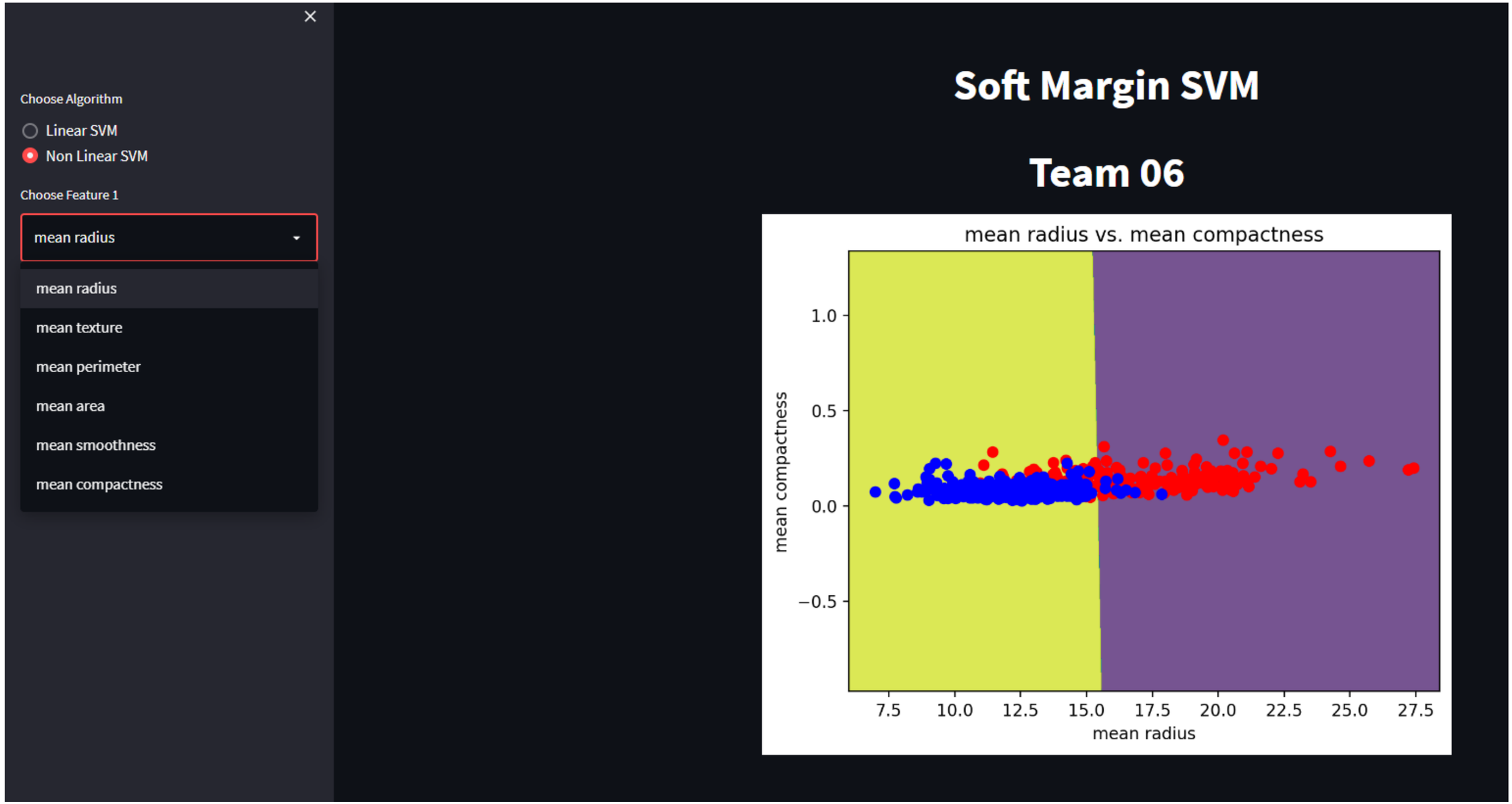
- mean radius
- mean texture
- mean perimeter
- mean area
- mean smoothness
- mean compactness

Soft Margin SVM

Team 06



Non-Linear SVM Classifier


Choose Algorithm
Linear SVM
Non Linear SVM
Choose Feature 1
mean radius
mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
Soft Margin SVM
Team 06
mean radius vs. mean compactness
mean compactness
1.0
0.5
0.0
−0.5
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
mean radius

Thank You