SUPPORT VECTOR MACHINES

Outline

- Definition - Support Vector Machine
- Decision boundary
- Linearly separable dataset
- Maximal Margin classifier (linear) hard-margin classifier
- Support vector classifier (linear) soft-margin classifier
- Support vector machines (nonlinear boundary)

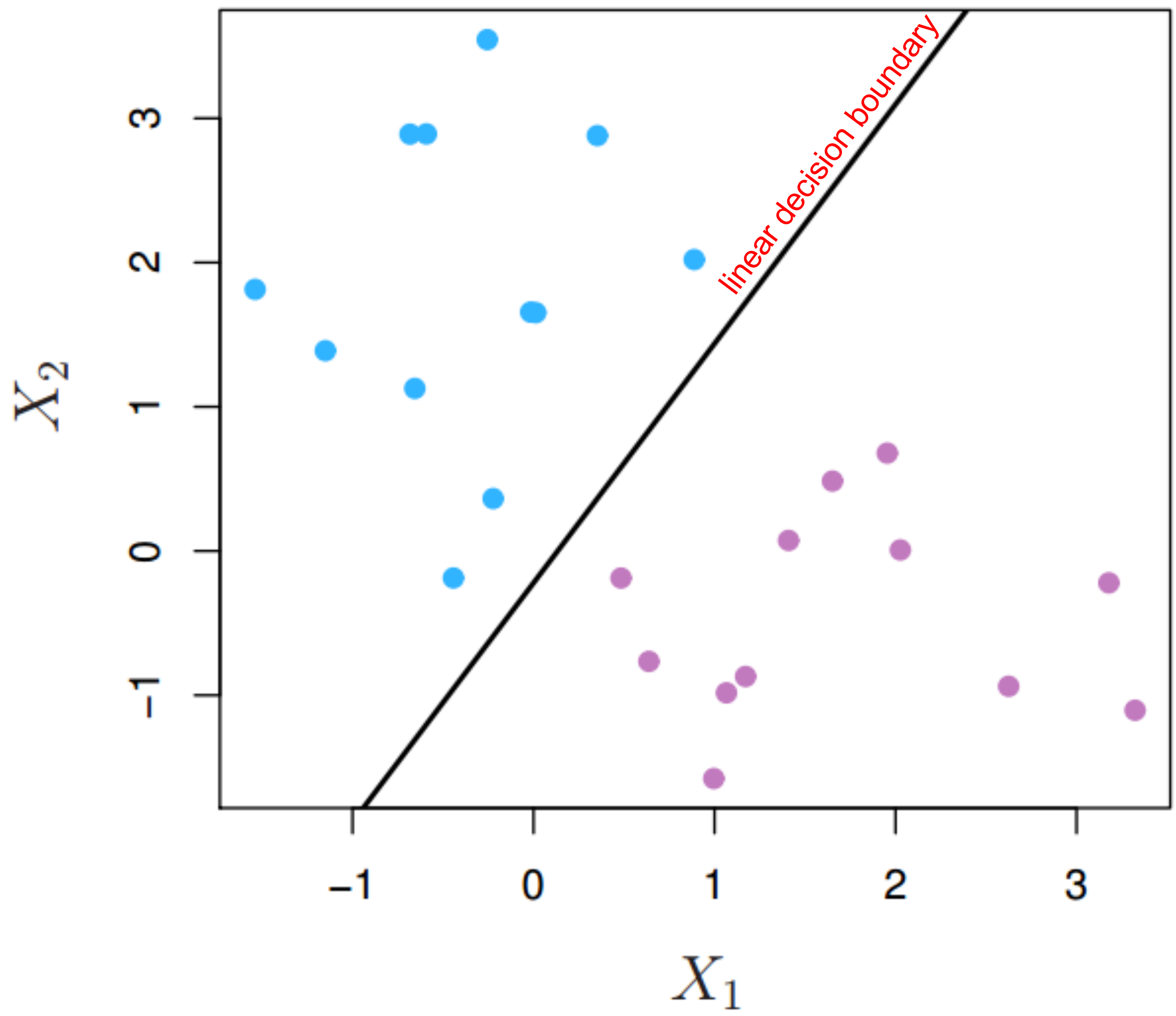
Support Vector Machine

- ML Method that classifies observations by using an optimization model to find a function that serves as the boundary between the categories of the response
- If the data is linearly separable the boundary is a line (or a plane)
- If the data is not linearly separable, a linear boundary that allows for misclassifications is used
- In the general case a non-linear boundary between the categories can be found by means of kernel functions

Linearly separable dataset

Y	X1	X2
0	2.5	1.5
0	1.7	0.6
1	2.3	1.1
0	0.8	2.5
1	1.1	0.9
1	1.5	1.9

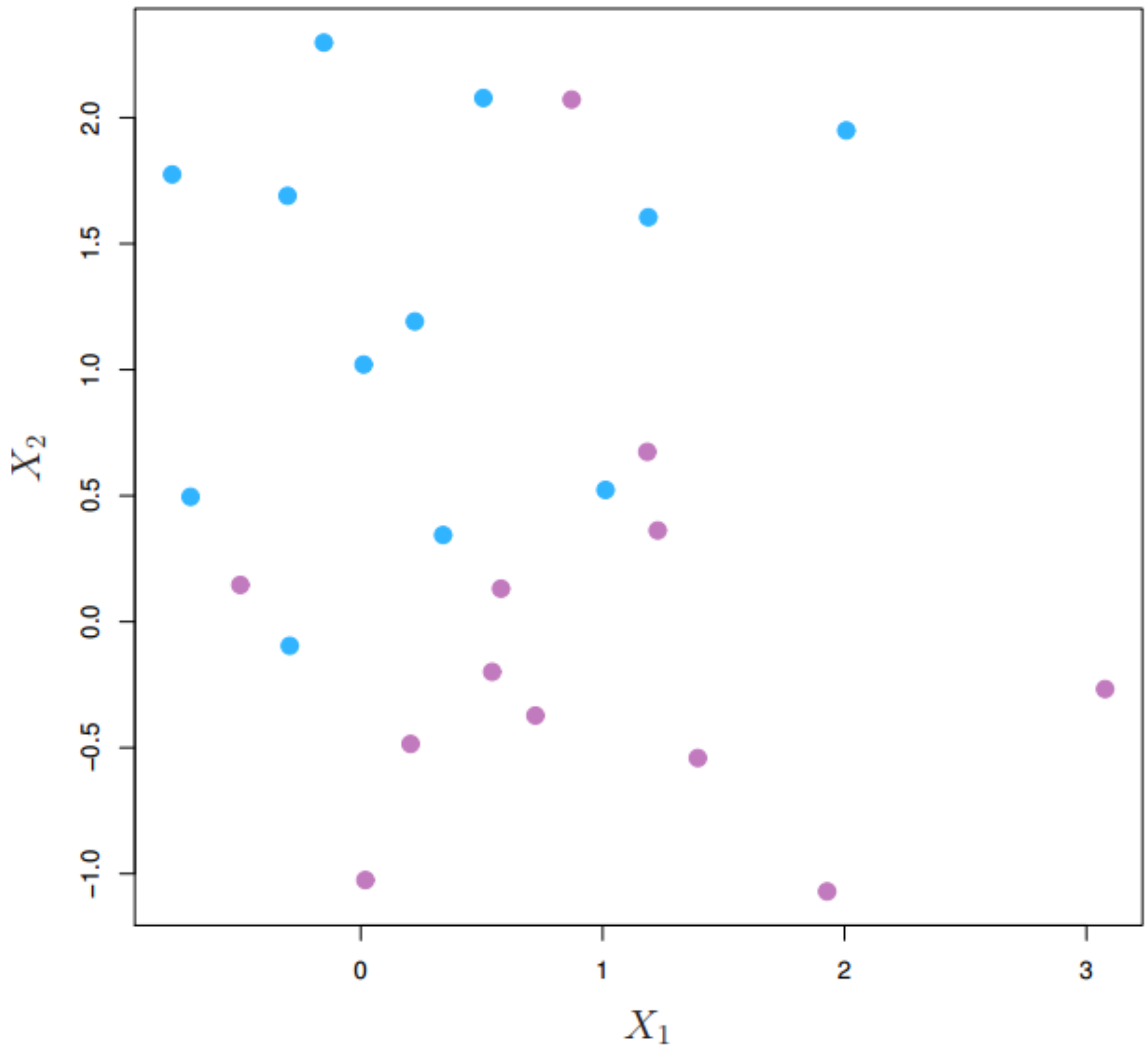
not actual values



Non-Linearly separable dataset

Y	X1	X2
0	2.5	1.5
0	1.7	0.6
1	2.3	1.1
0	0.8	2.5
1	1.1	0.9
1	1.5	1.9

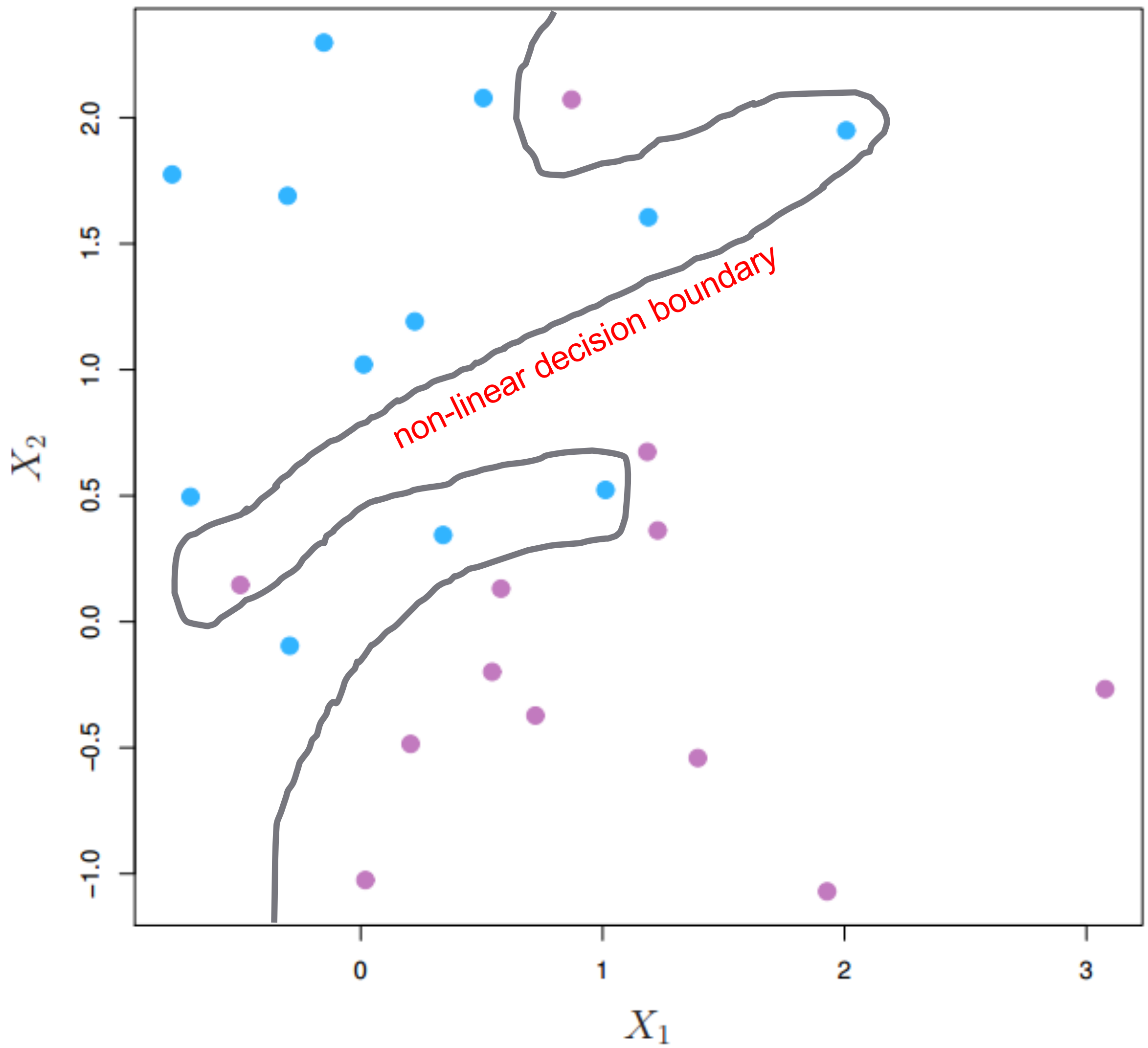
not actual values



Non-Linearly separable dataset

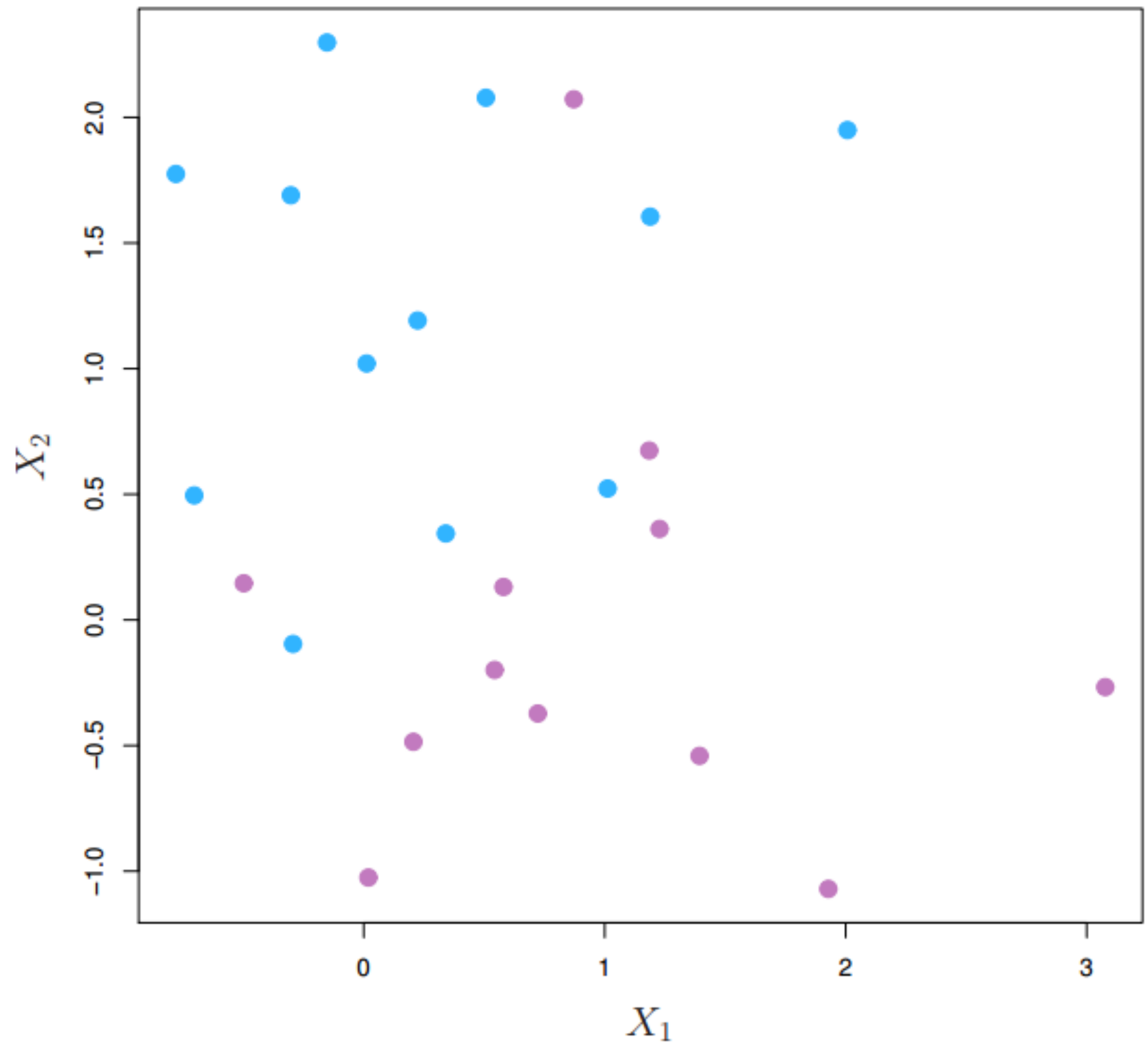
Y	X1	X2
0	2.5	1.5
0	1.7	0.6
1	2.3	1.1
0	0.8	2.5
1	1.1	0.9
1	1.5	1.9

not actual values



Non-Linearly separable dataset

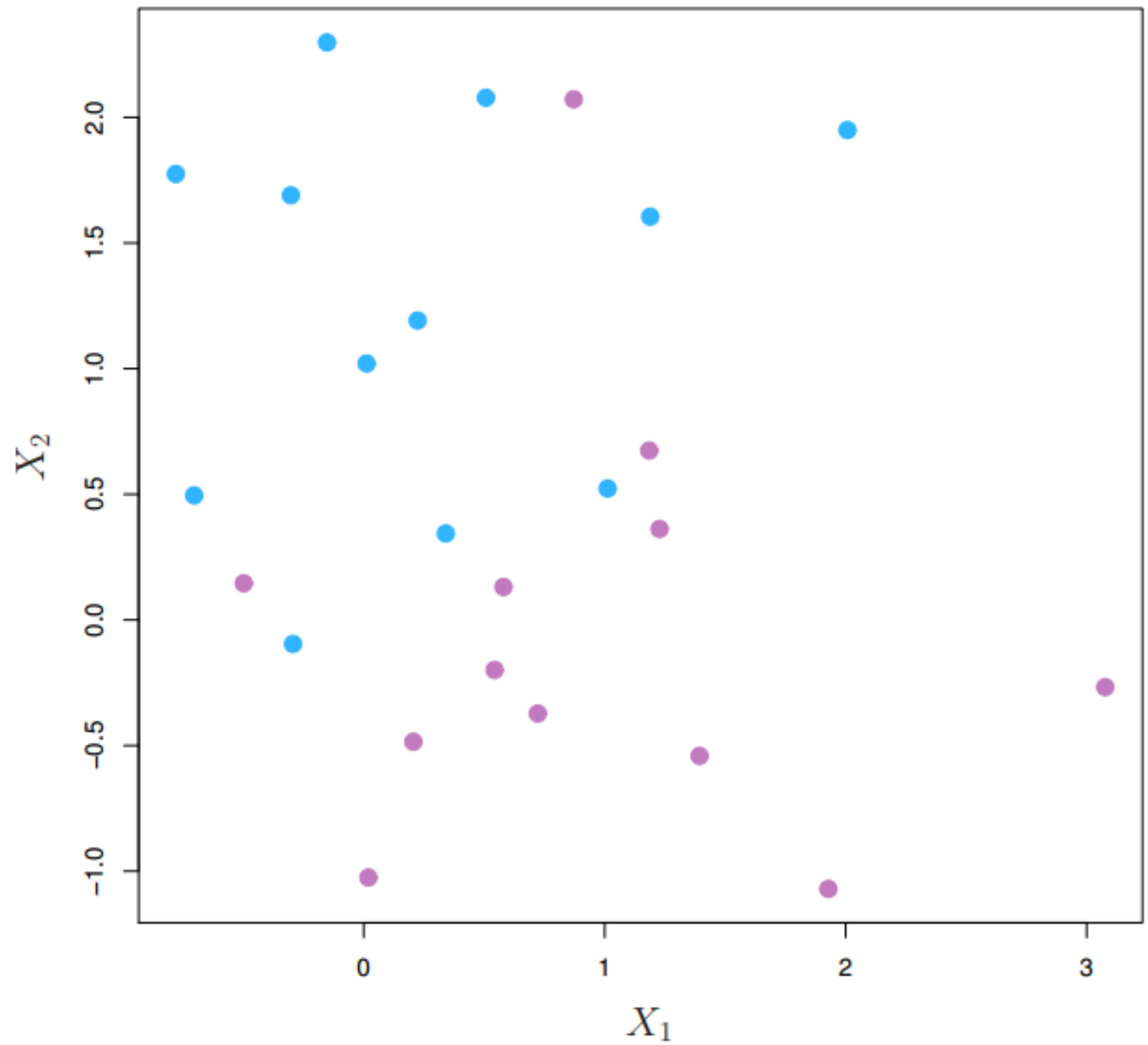
How to deal with non-linearly separable datasets?



Non-Linearly separable dataset

How to deal with non-linearly separable datasets?

Convert dataset into a linearly separable dataset



Non-Linearly separable dataset

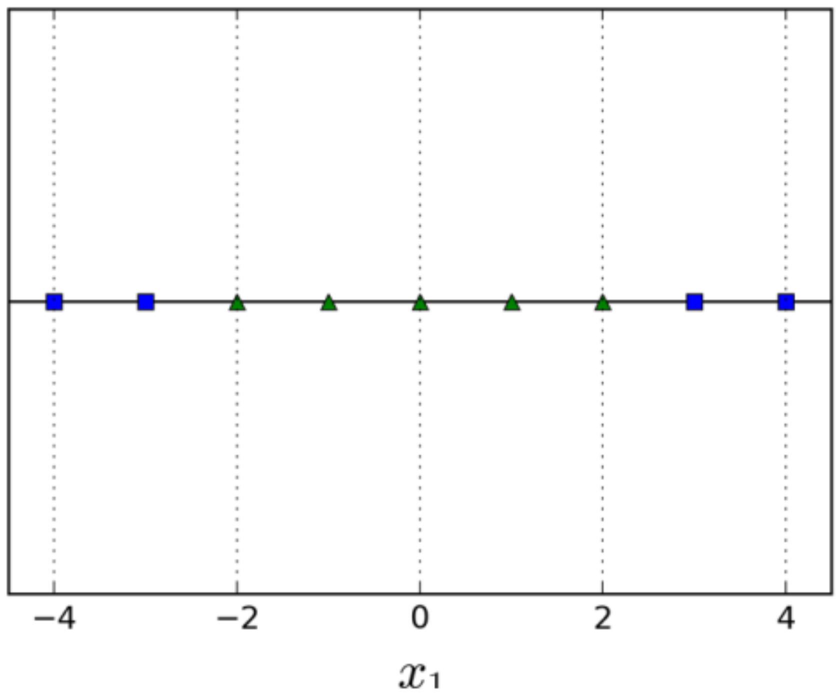
- To convert into a linearly separable dataset

  add more features (polynomial, exponential, etc.)

Non-Linearly separable dataset

To convert into a linearly separable dataset

add more features (polynomial, exponential, etc.)

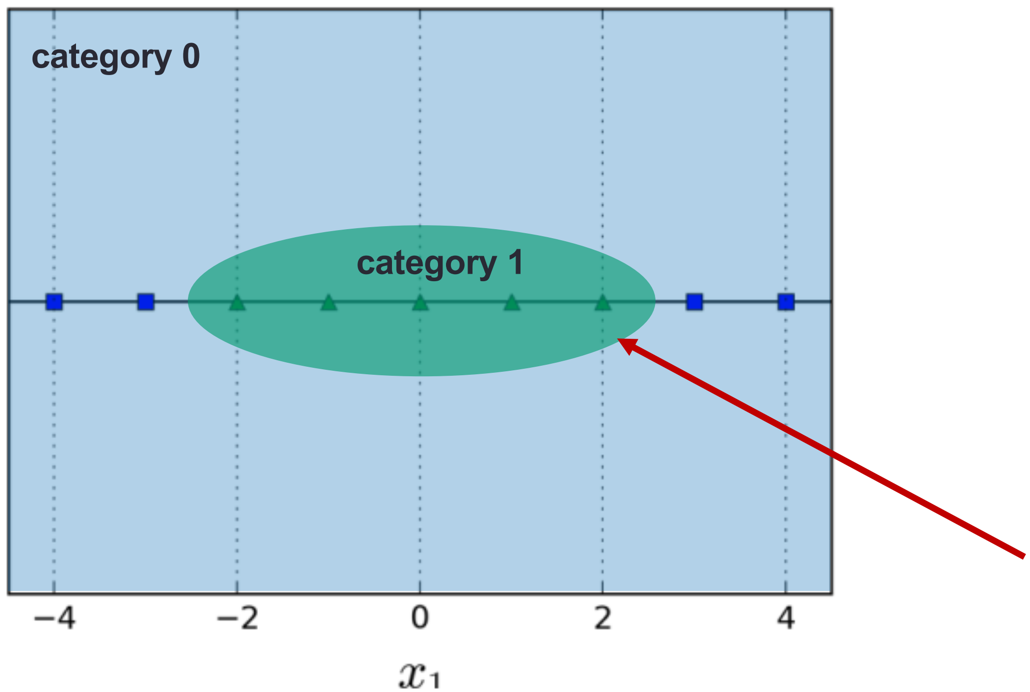


one-feature dataset

Non-Linearly separable dataset

To convert into a linearly separable dataset

add more features (polynomial, exponential, etc.)



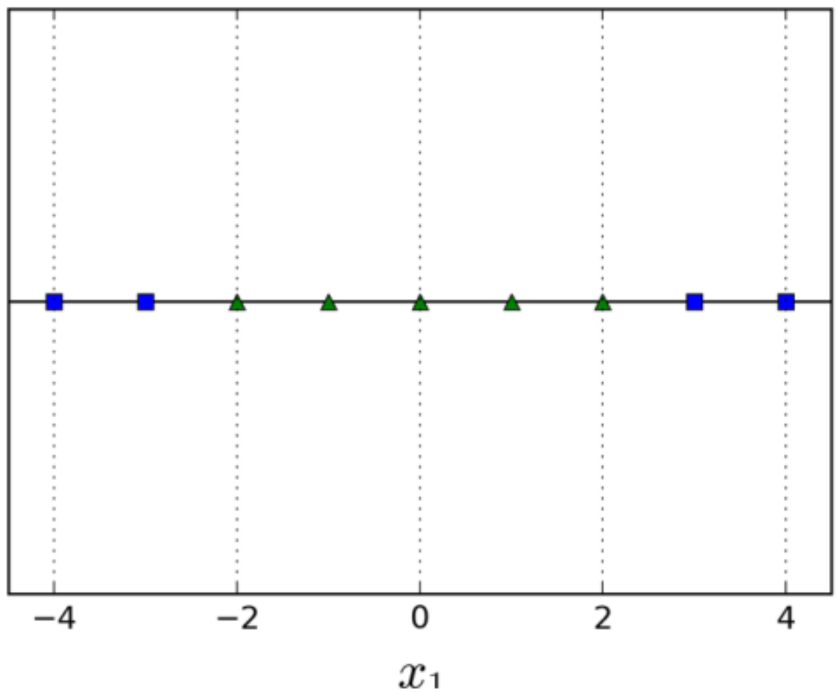
not linearly separable

not a linear boundary

Non-Linearly separable dataset

To convert into a linearly separable dataset

add more features (polynomial, exponential, etc.)



solution:

create new feature x_2

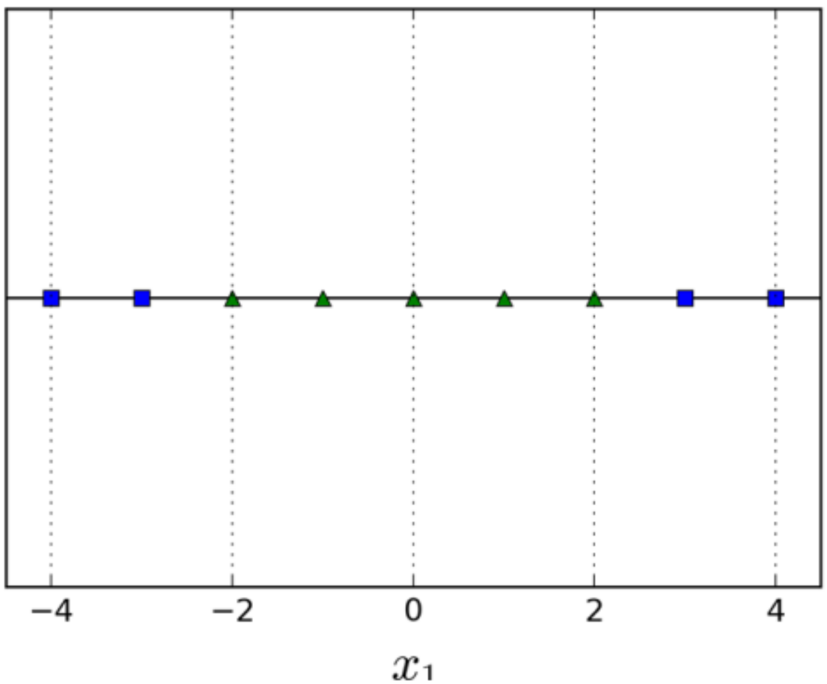
$$x_2 = x_1^2$$

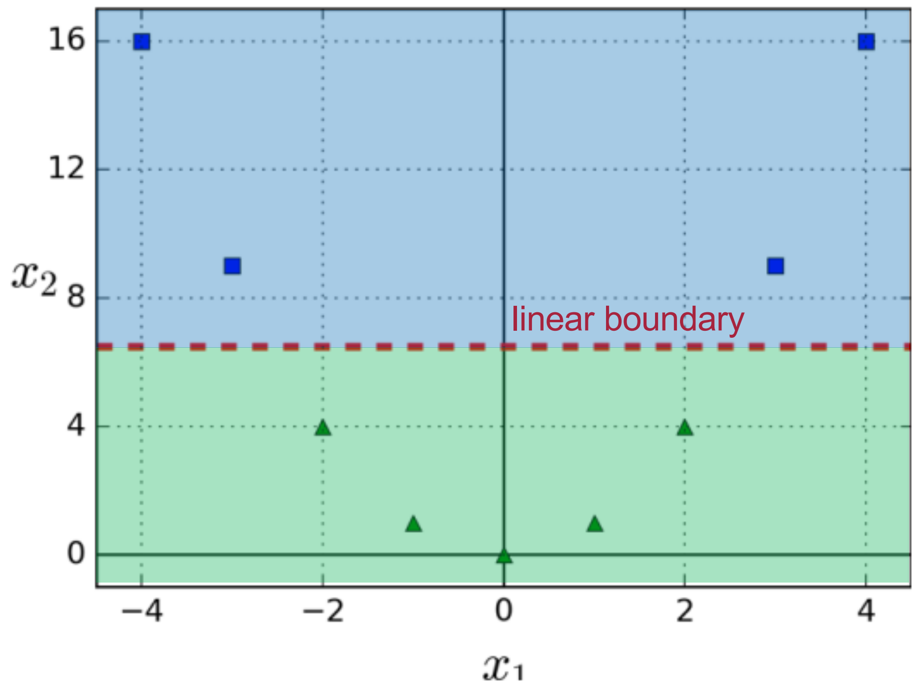
This “kernel” function transform the data into a linearly separable data

Non-Linearly separable dataset

To convert into a linearly separable dataset

add more features (polynomial, exponential, etc.)

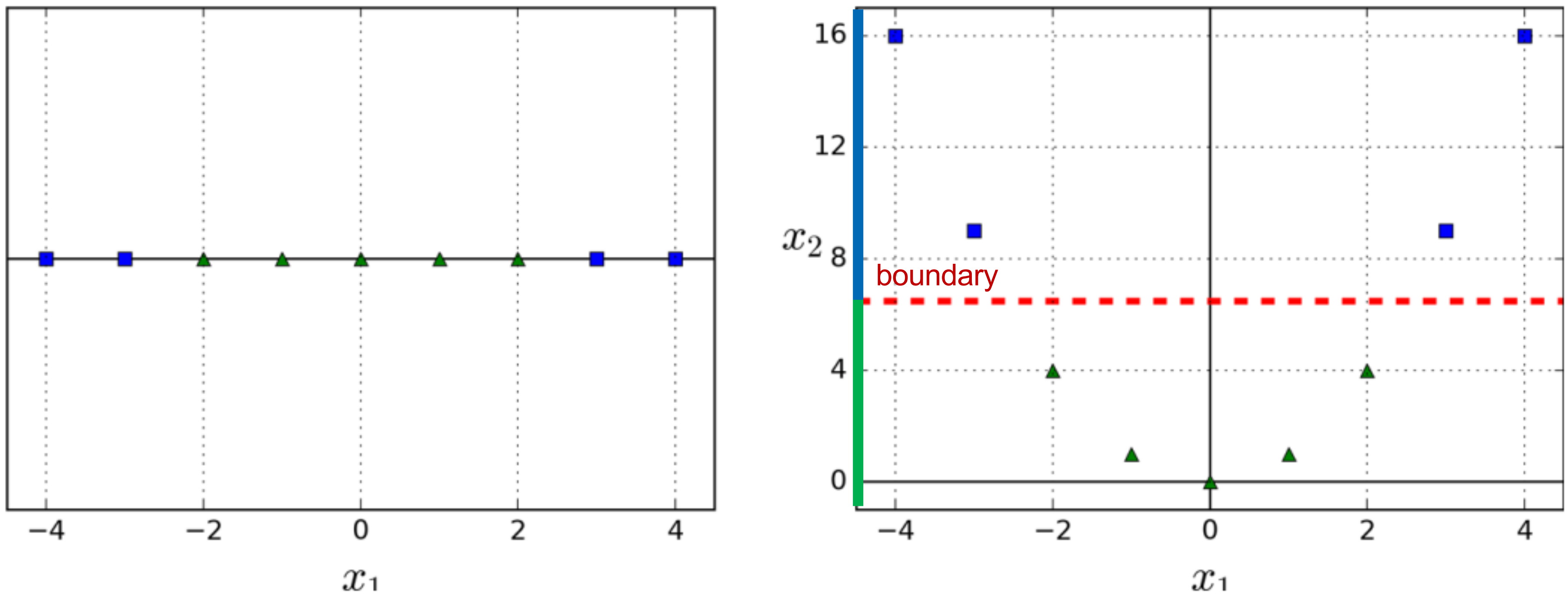




Non-Linearly separable dataset

To convert into a linearly separable dataset

add more features (polynomial, exponential, etc.)

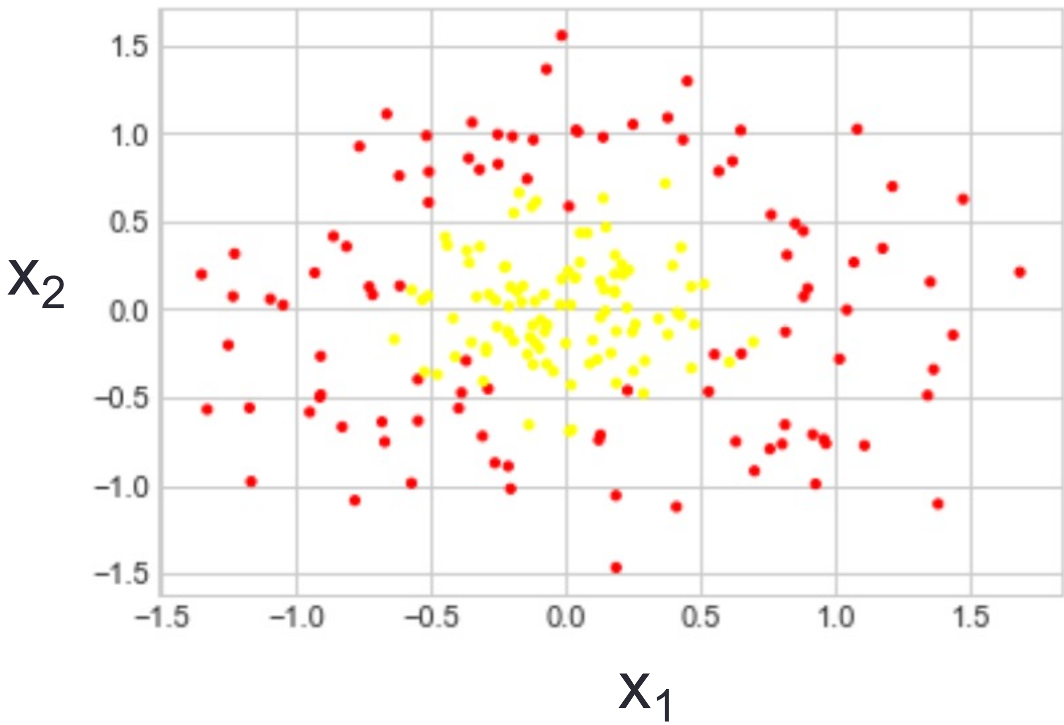


actually x_1 not needed (y is linearly separable on x_2 alone)

Non-Linearly separable dataset

To convert into a linearly separable dataset

add more features (polynomial, exponential, etc.)



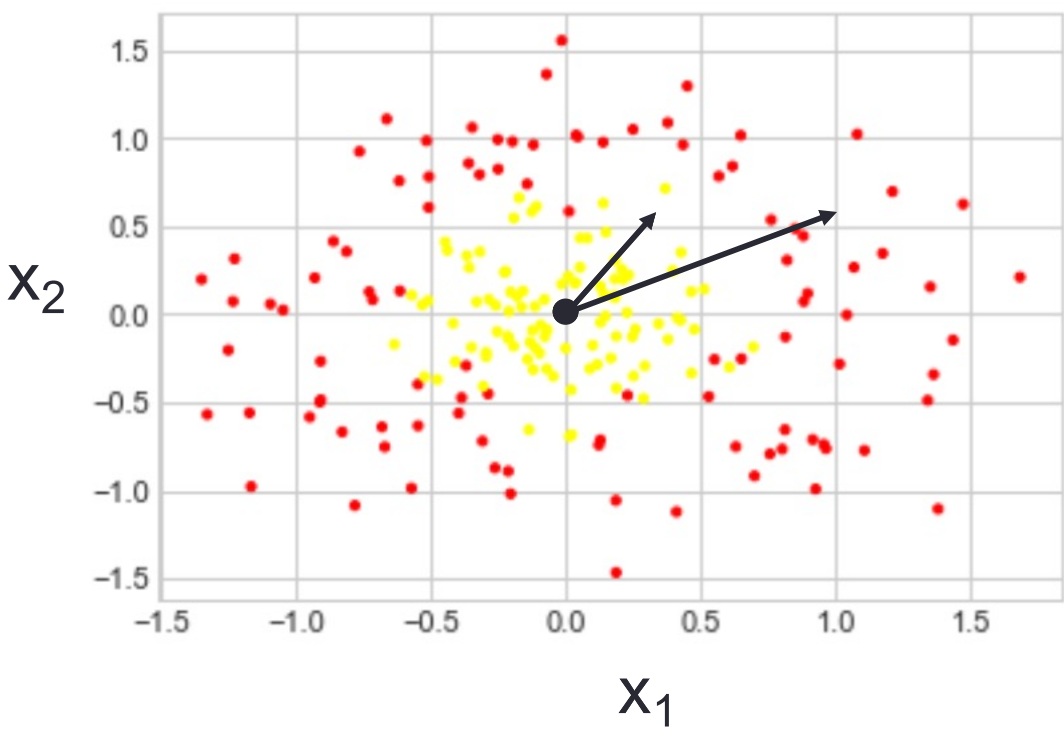
not linearly

separable

Non-Linearly separable dataset

To convert into a linearly separable dataset

add more features (polynomial, exponential, etc.)



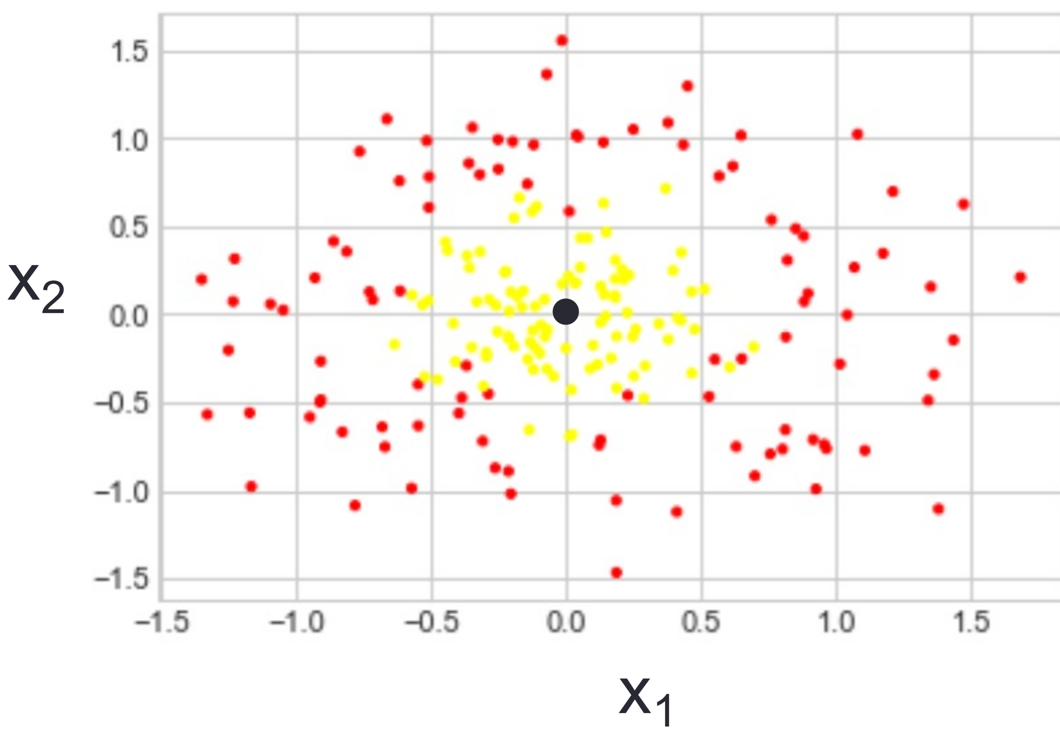
not linearly

separable

Non-Linearly separable dataset

To convert into a linearly separable dataset

add more features (polynomial, exponential, etc.)



add new feature x_3

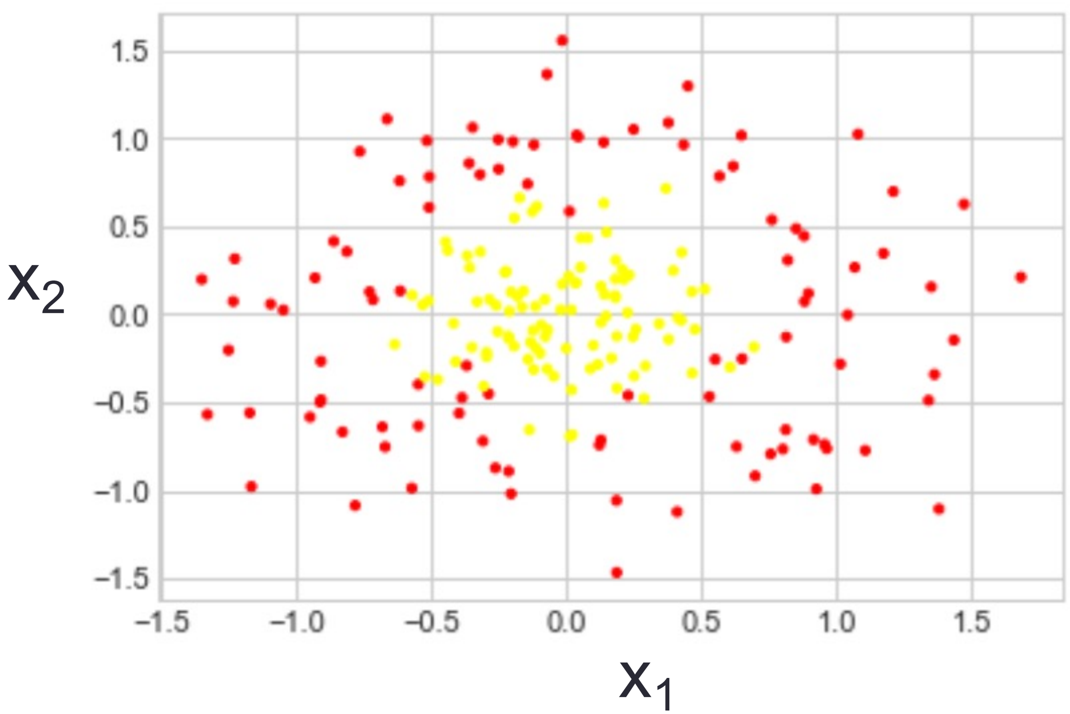
$$x_3 = x_1^2 + x_2^2$$

x_3 is the (squared) distance from the origin

Non-Linearly separable dataset

To convert into a linearly separable dataset

add more features (polynomial, exponential, etc.)

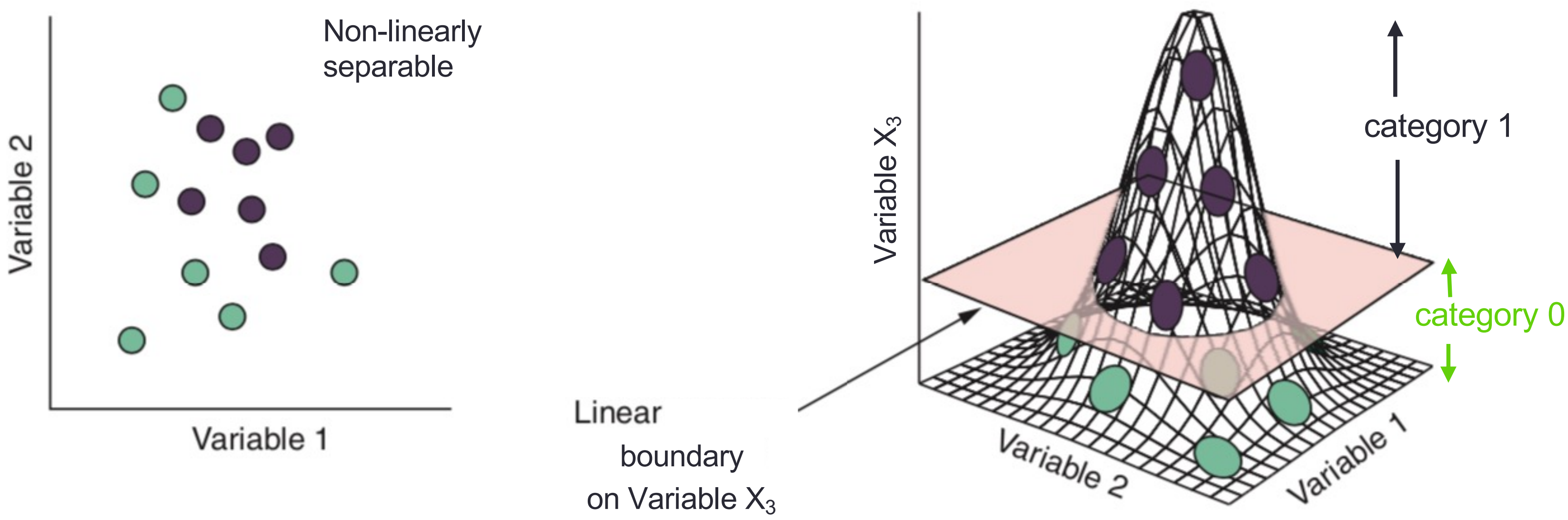


add new feature x_3

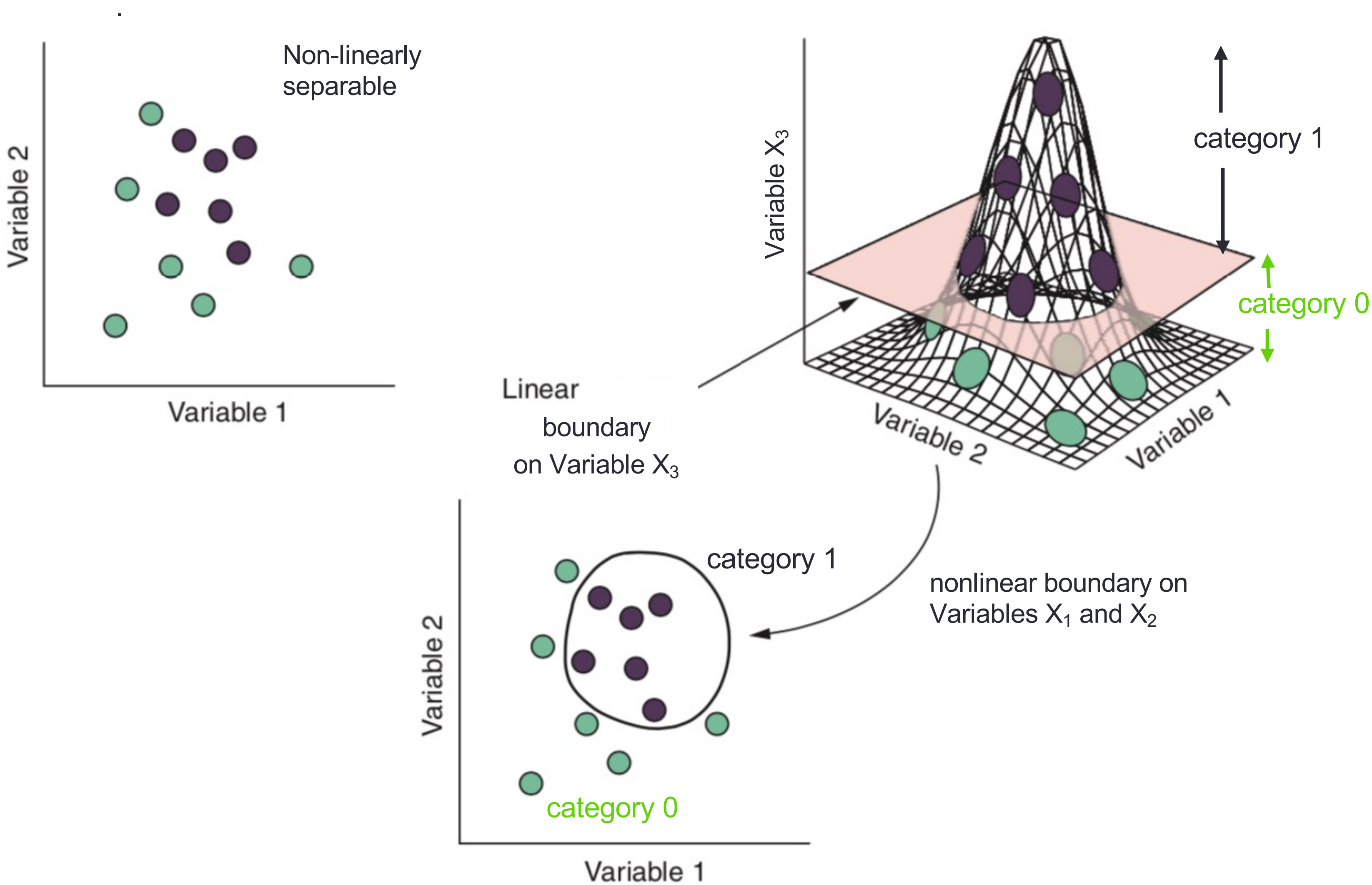
$$x_3 = x_1^2 + x_2^2$$

This "kernel" function transform the data into a linearly separable data

Radial Basis Function (*RBF*)



Radial Basis Function (*RBF*)



Transformation for a Linearly separable dataset

```
from sklearn.datasets import make_circles
```

```
X,y = make_circles(100,factor=0.1,noise=0.15)
```

```
X[:5]
```

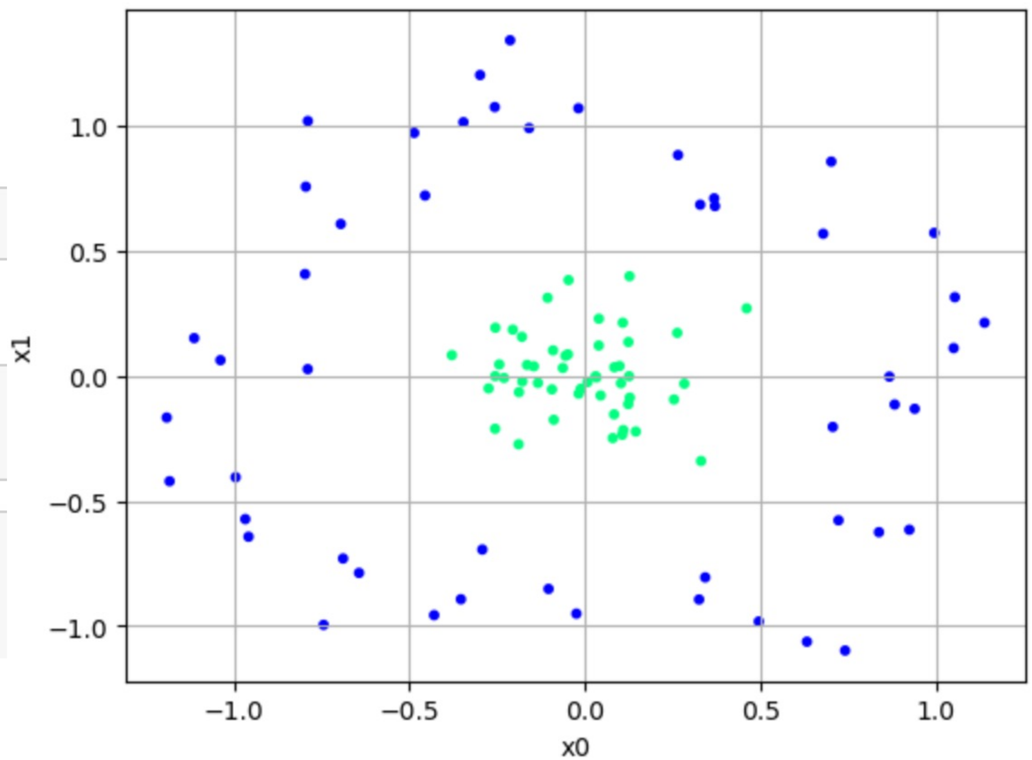
```
array([[ 0.74041877, -1.09870346],
       [-0.10386918, -0.85210173],
       [ 0.88145859, -0.11440858],
       [-0.29868561,  1.20389118],
       [-0.08891853, -0.17552215]])
```

```
y[:5]
```

```
array([0, 0, 0, 0, 1])
```

```
x0 = X[:,0]
x1 = X[:,1]
```

```
plt.scatter(x0,x1,c=y,s=10,cmap='winter')
plt.ylabel('x1')
plt.xlabel('x0')
```



Transformation for a Linearly separable dataset

```
from sklearn.datasets import make_circles
```

```
X,y = make_circles(100,factor=0.1,noise=0.15)
```

```
X[:5]
```

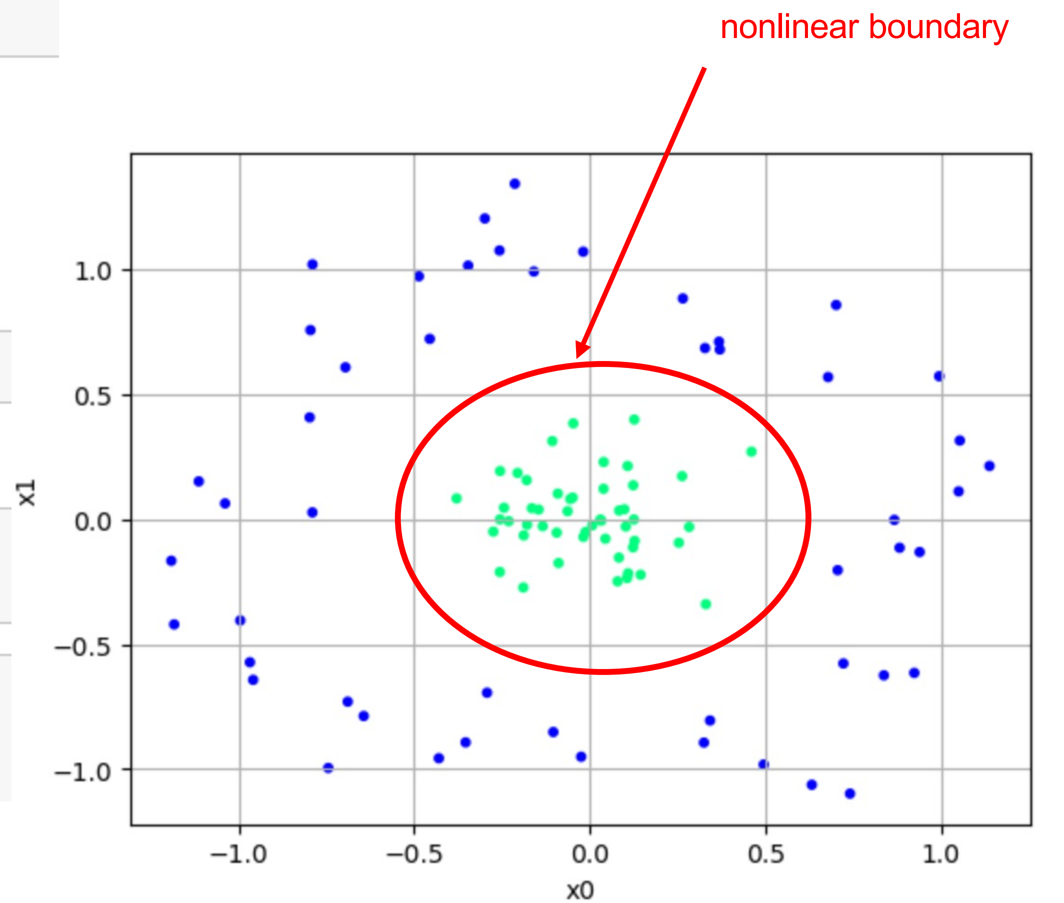
```
array([[ 0.74041877, -1.09870346],
       [-0.10386918, -0.85210173],
       [ 0.88145859, -0.11440858],
       [-0.29868561,  1.20389118],
       [-0.08891853, -0.17552215]])
```

```
y[:5]
```

```
array([0, 0, 0, 0, 1])
```

```
x0 = X[:,0]
x1 = X[:,1]
```

```
plt.scatter(x0,x1,c=y,s=10,cmap='winter')
plt.ylabel('x1')
plt.xlabel('x0')
```



Transformation for a Linearly separable dataset

```
X[:5]
```

```
array([[ 0.74041877, -1.09870346],
       [-0.10386918, -0.85210173],
       [ 0.88145859, -0.11440858],
       [-0.29868561,  1.20389118],
       [-0.08891853, -0.17552215]])
```

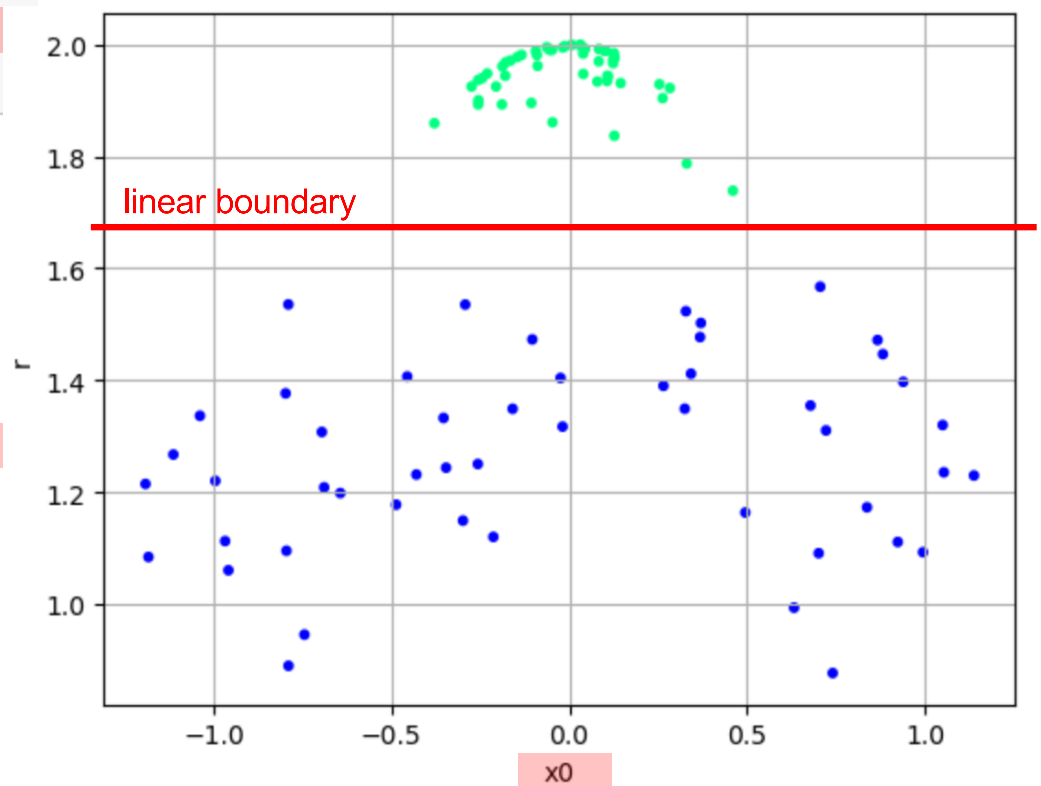
```
r = np.exp(-(X**2))    elementwise exponentiation
r[:5]
```

```
array([[0.57797772, 0.29904856],
       [0.98926918, 0.48380306],
       [0.45979743, 0.98699597],
       [0.91465065, 0.23472188],
       [0.99212467, 0.96966171]])
```

```
r = r.sum(1)    row sums
r[:3]
```

```
array([0.87702628, 1.47307224, 1.4467934
```

```
plt.scatter(x0,r,c=y,s=10,cmap='winter')
plt.xlabel('x0')
plt.ylabel('r')
plt.grid();
```



Transformation for a Linearly separable dataset

```
X[:5]
```

```
array([[ 0.74041877, -1.09870346],
       [-0.10386918, -0.85210173],
       [ 0.88145859, -0.11440858],
       [-0.29868561,  1.20389118],
       [-0.08891853, -0.17552215]])
```

```
r = np.exp(-(X**2))
r[:5]
```

elementwise exponentiation

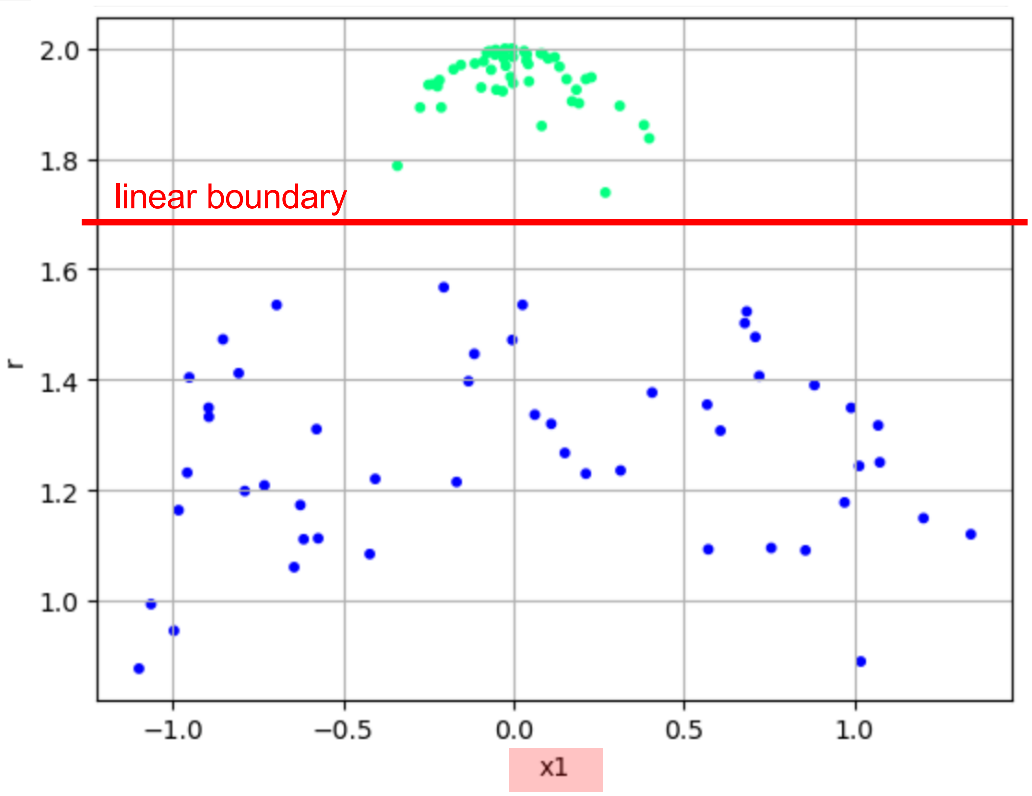
```
array([[0.57797772, 0.29904856],
       [0.98926918, 0.48380306],
       [0.45979743, 0.98699597],
       [0.91465065, 0.23472188],
       [0.99212467, 0.96966171]])
```

```
r = r.sum(1)
r[:3]
```

row sums

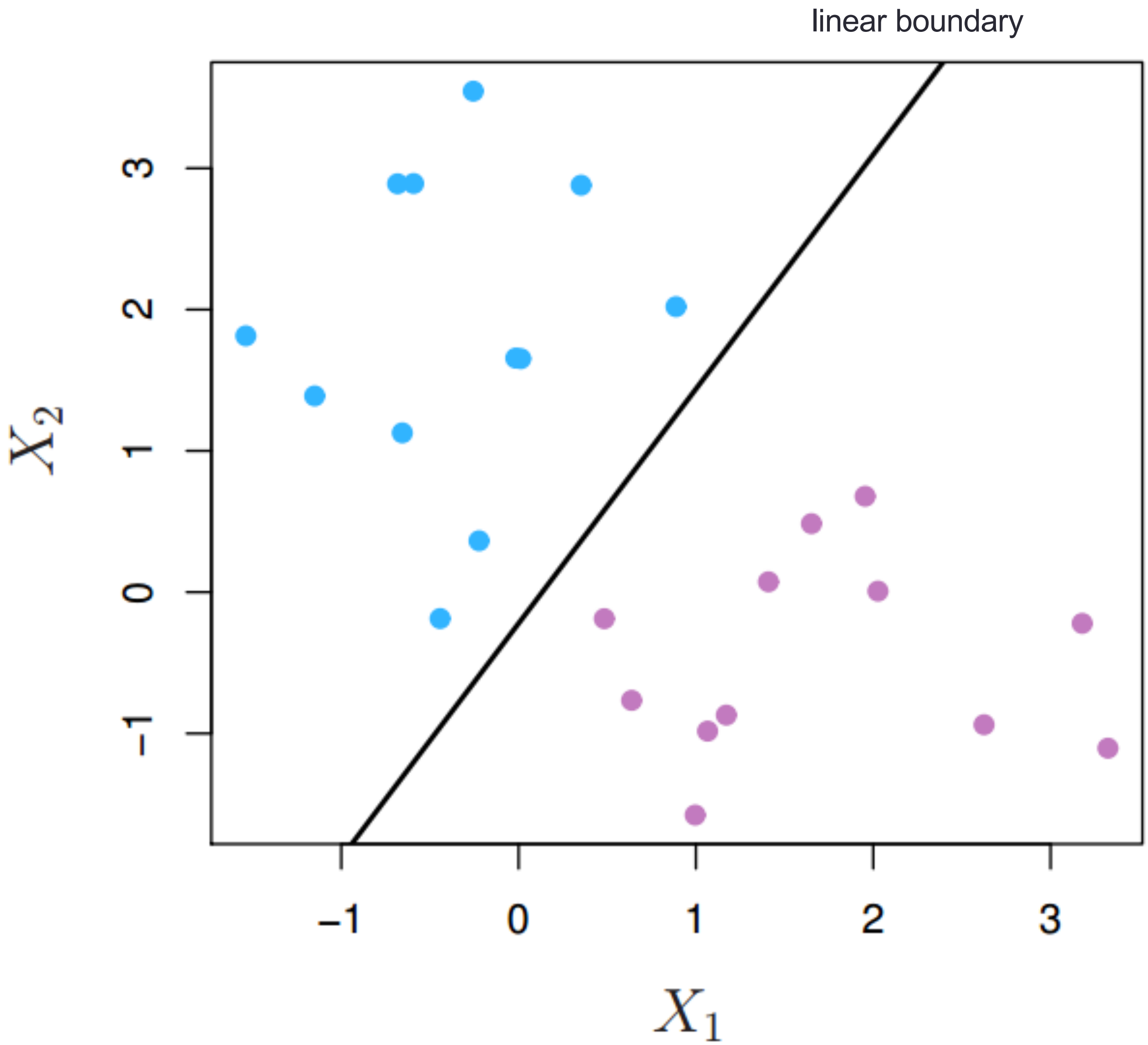
```
array([0.87702628, 1.47307224, 1.4467934
```

```
plt.scatter(x1,r,c=y,s=10,cmap='winter')
plt.xlabel('x1')
plt.ylabel('r')
plt.grid();
```



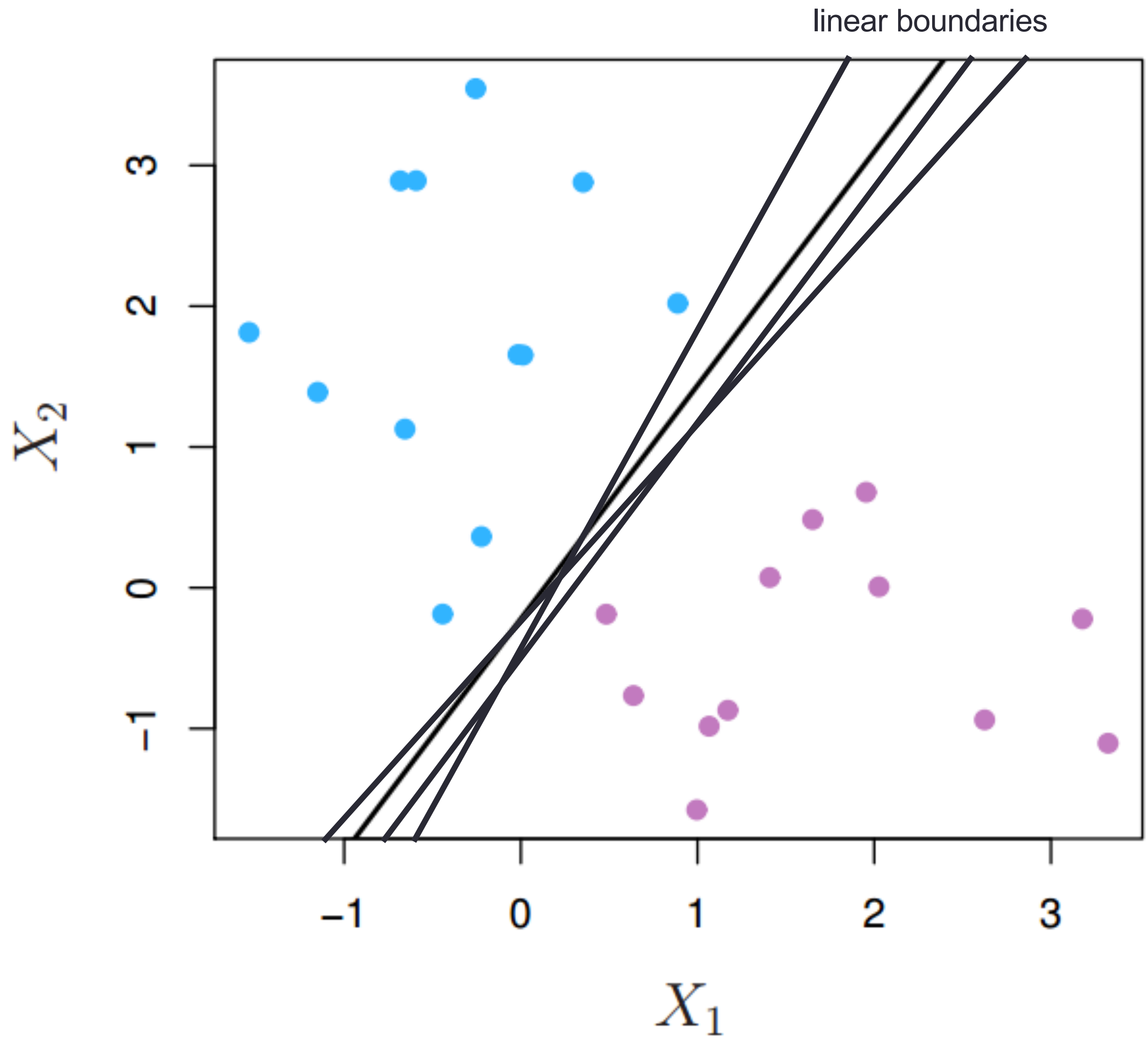
Linearly separable dataset

If a dataset is linearly separable then, there are many linear boundaries available, always



Linearly separable dataset

If a dataset is linearly separable then, there are many linear boundaries available, always



Linearly separable dataset

If a dataset is

linearly separable

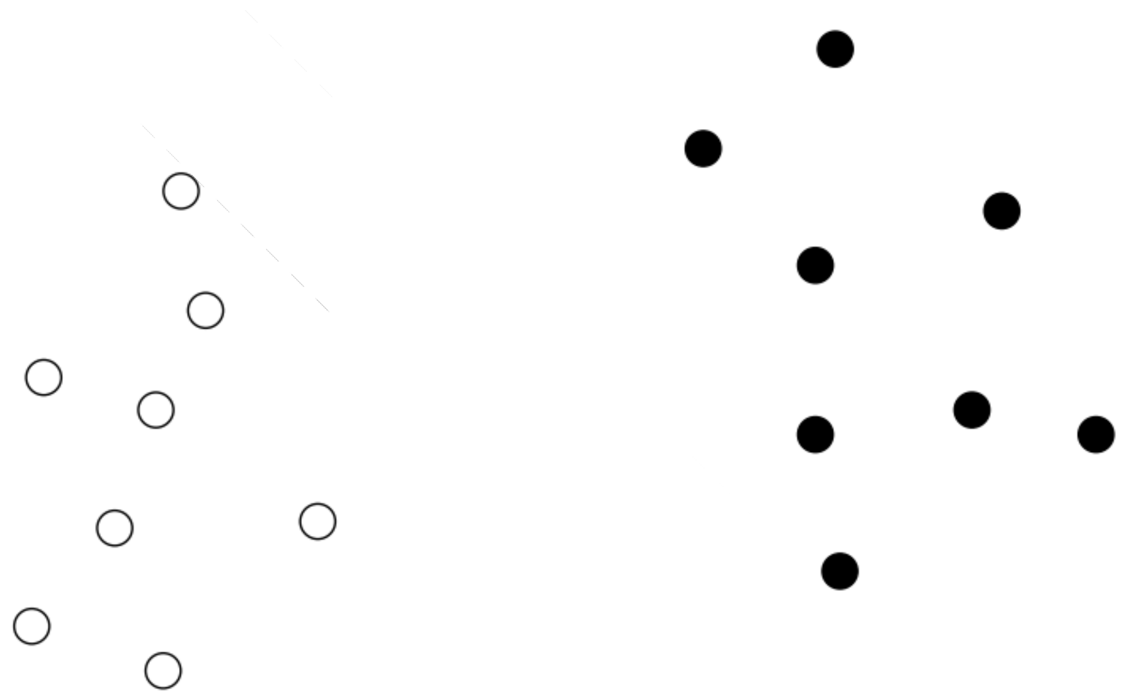
then, there are

many linear

boundaries

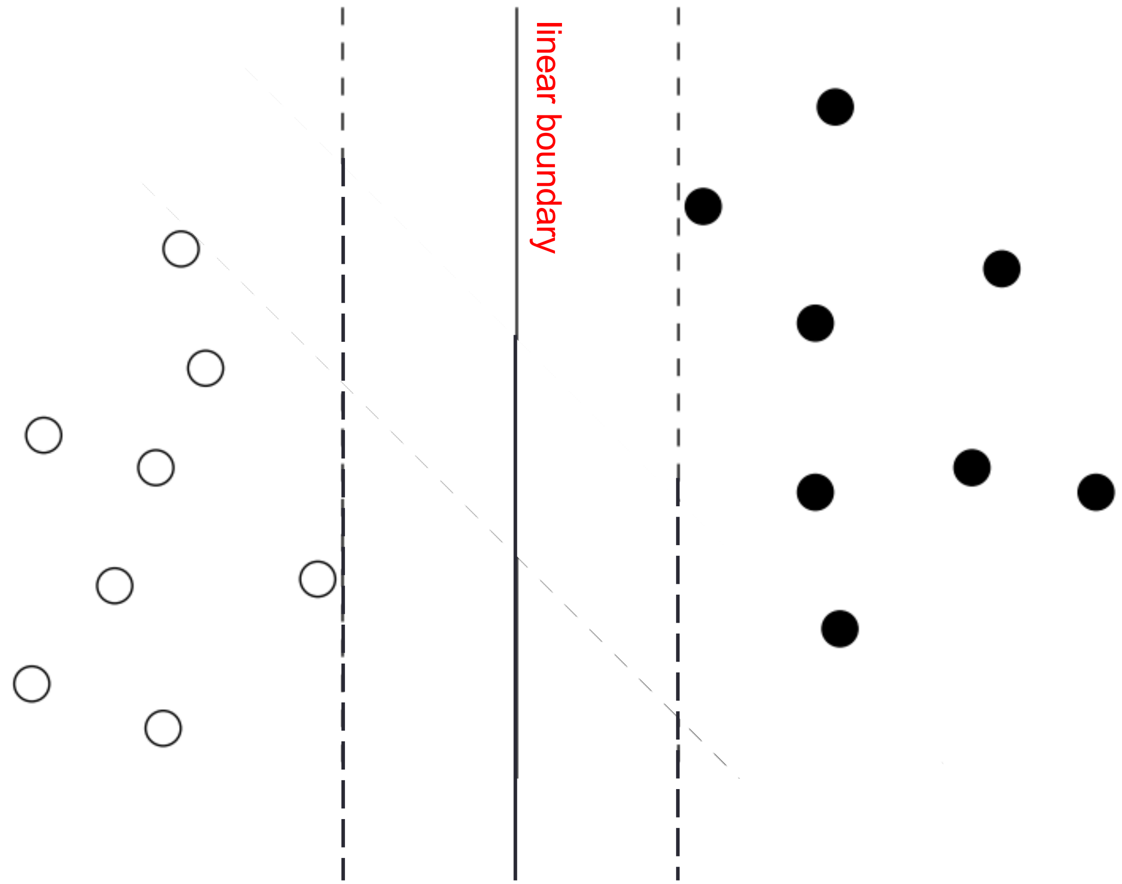
available,

always



Linearly separable dataset

If a dataset is linearly separable then, there are many linear boundaries available, always



Linearly separable dataset

If a dataset is

linearly separable

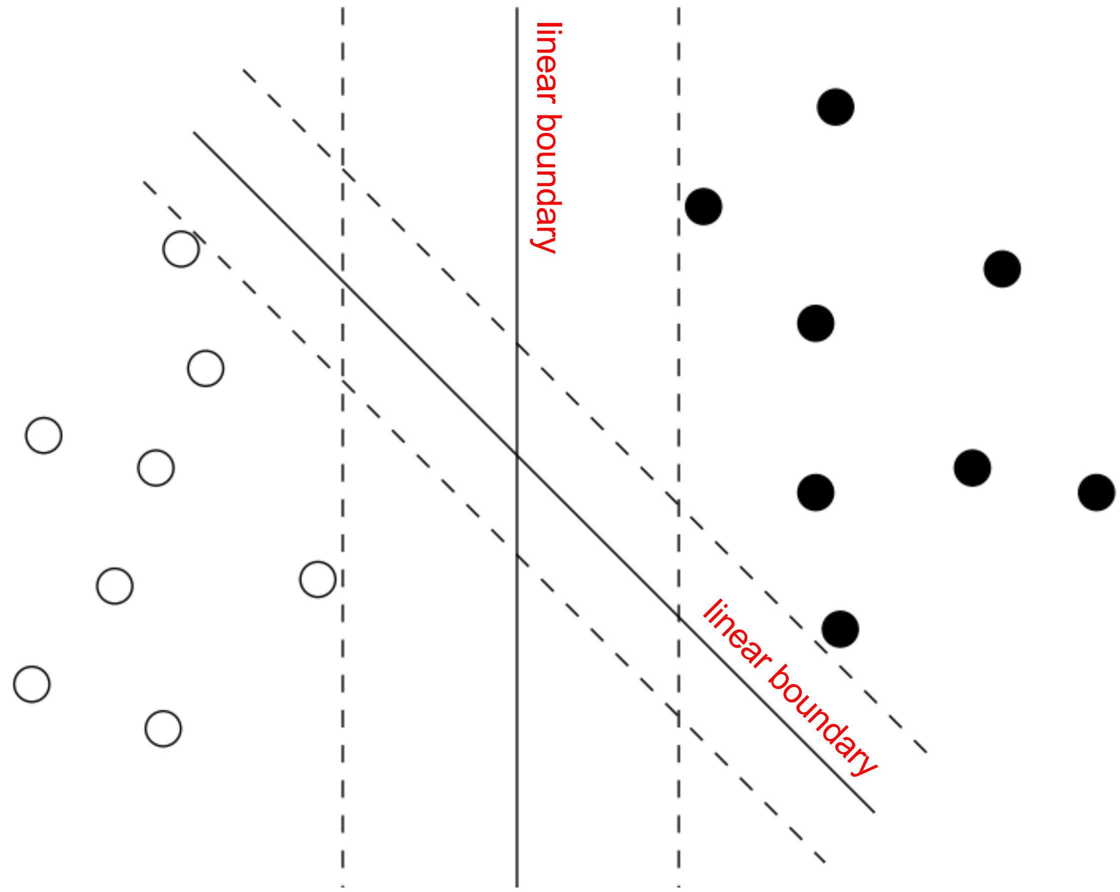
then, there are

many linear

boundaries

available,

always

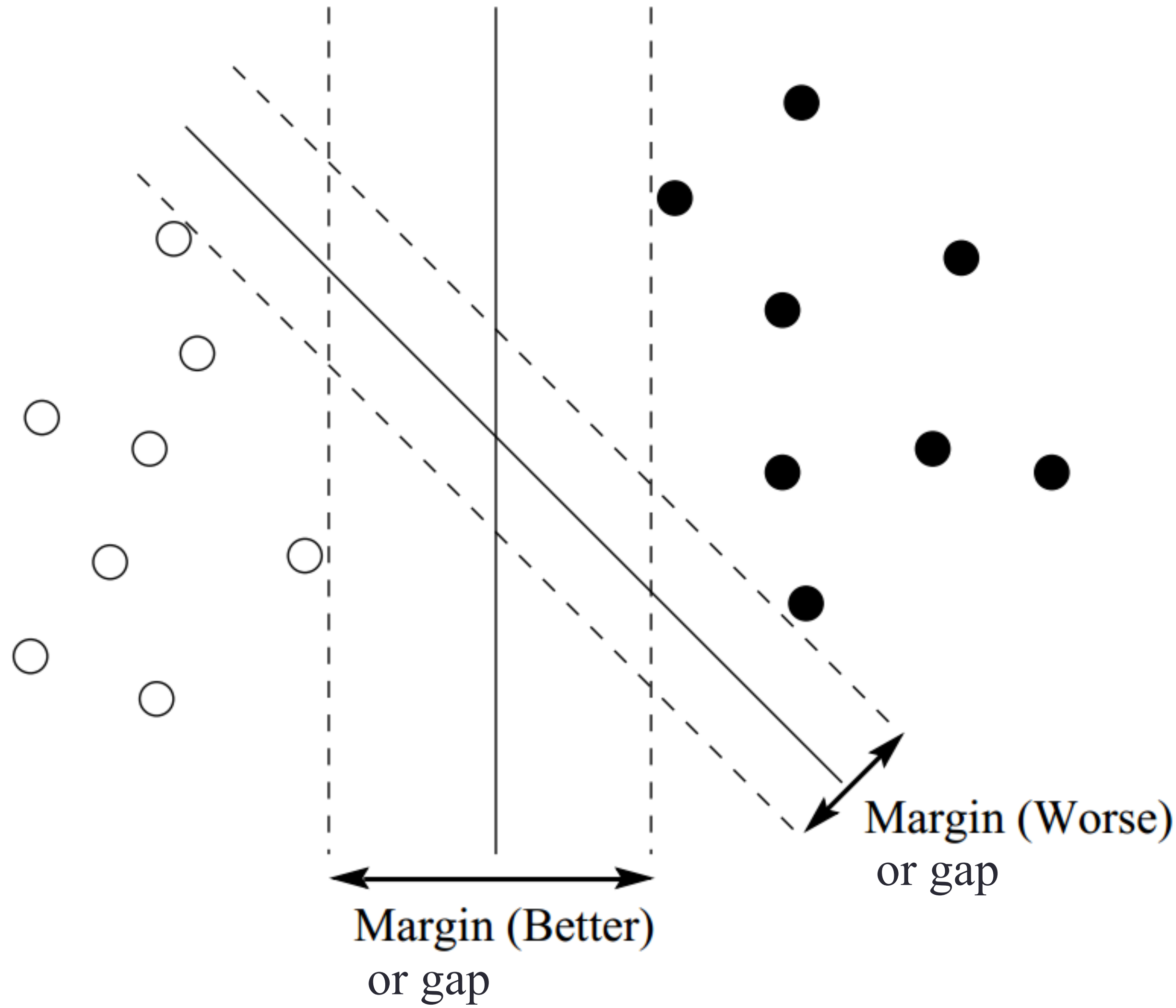


Maximal Margin Classifier

Maximal Margin Classifier

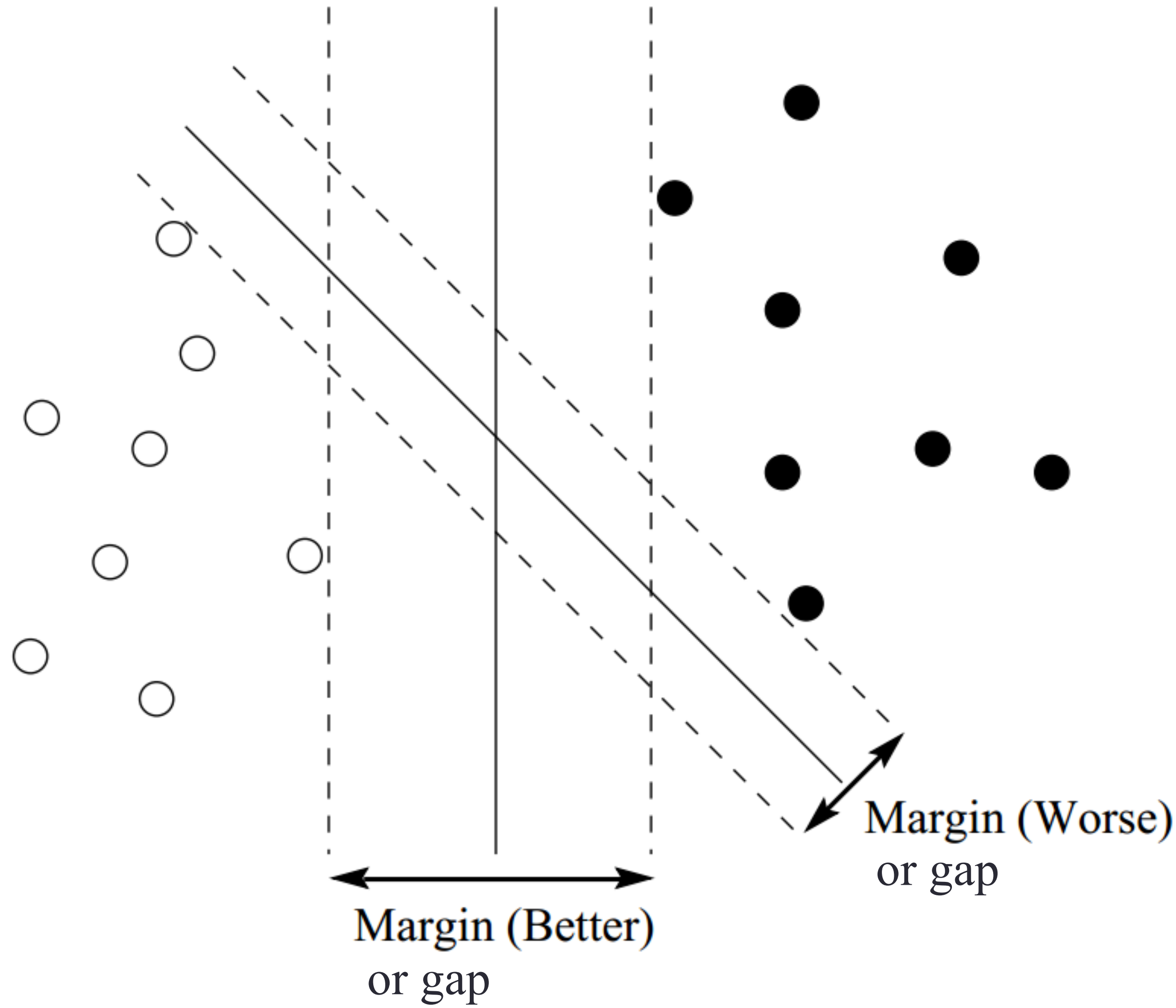
Find the separating hyperplane that makes the biggest gap (or margin) between the two classes

Objective is to maximize the Margin



Maximal Margin Classifier

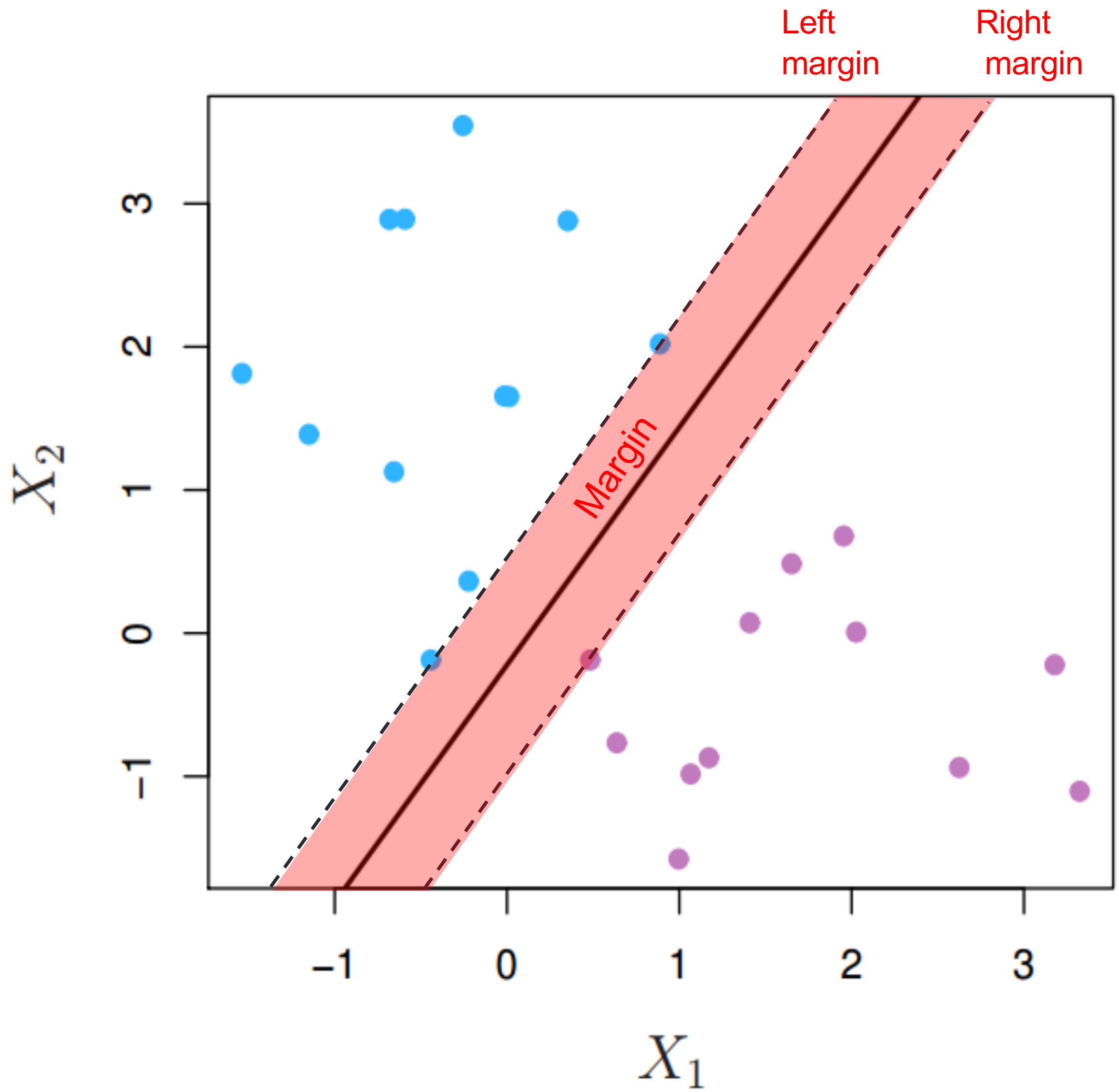
Think of fitting the widest street between the two classes



Hard Margin Classifier

Hard Margin Classifier

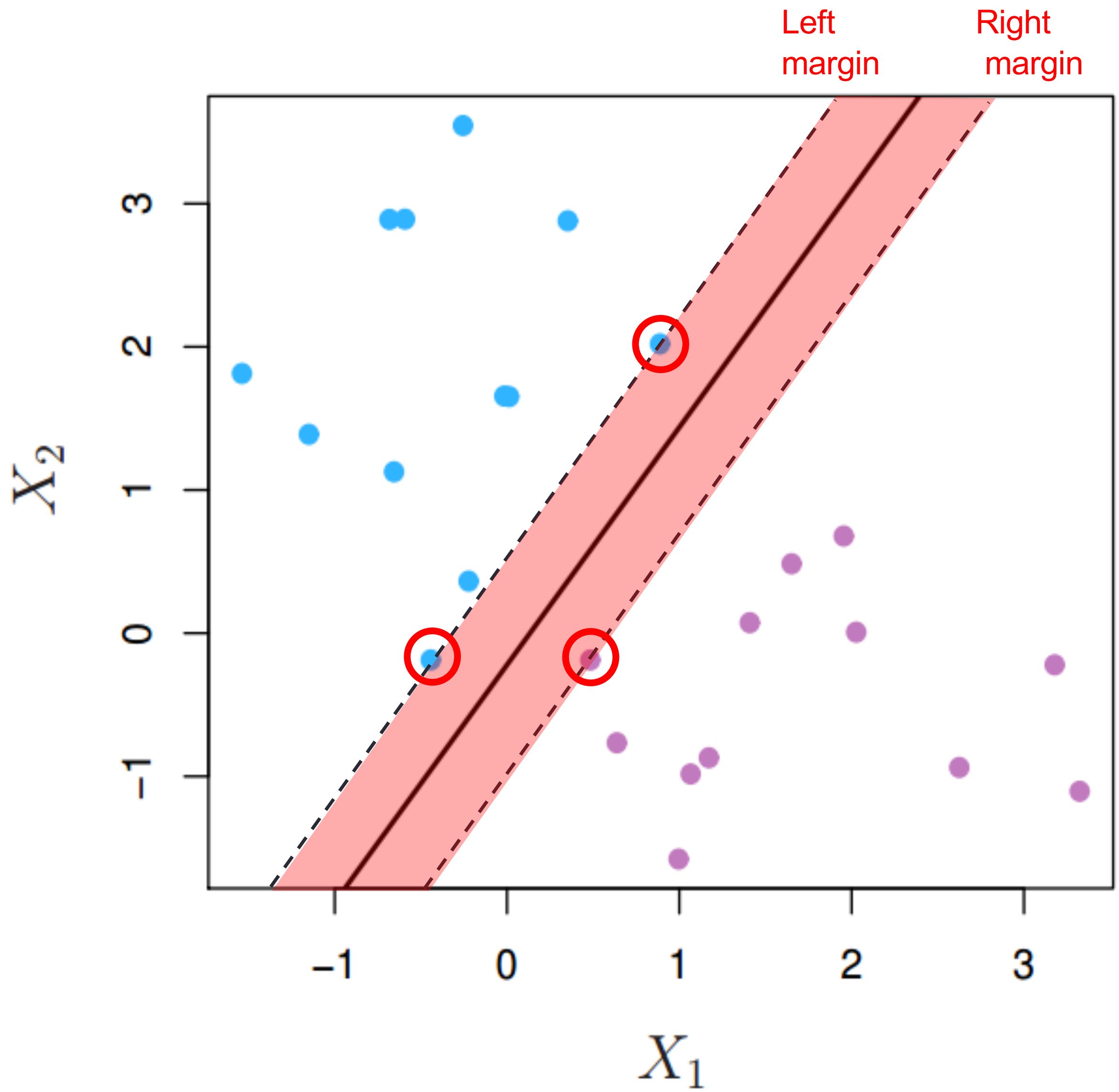
Restrict that all observations must be off the street and on the correct side



Hard Margin Classifier

Restrict that all observations must be off the street and on the correct side

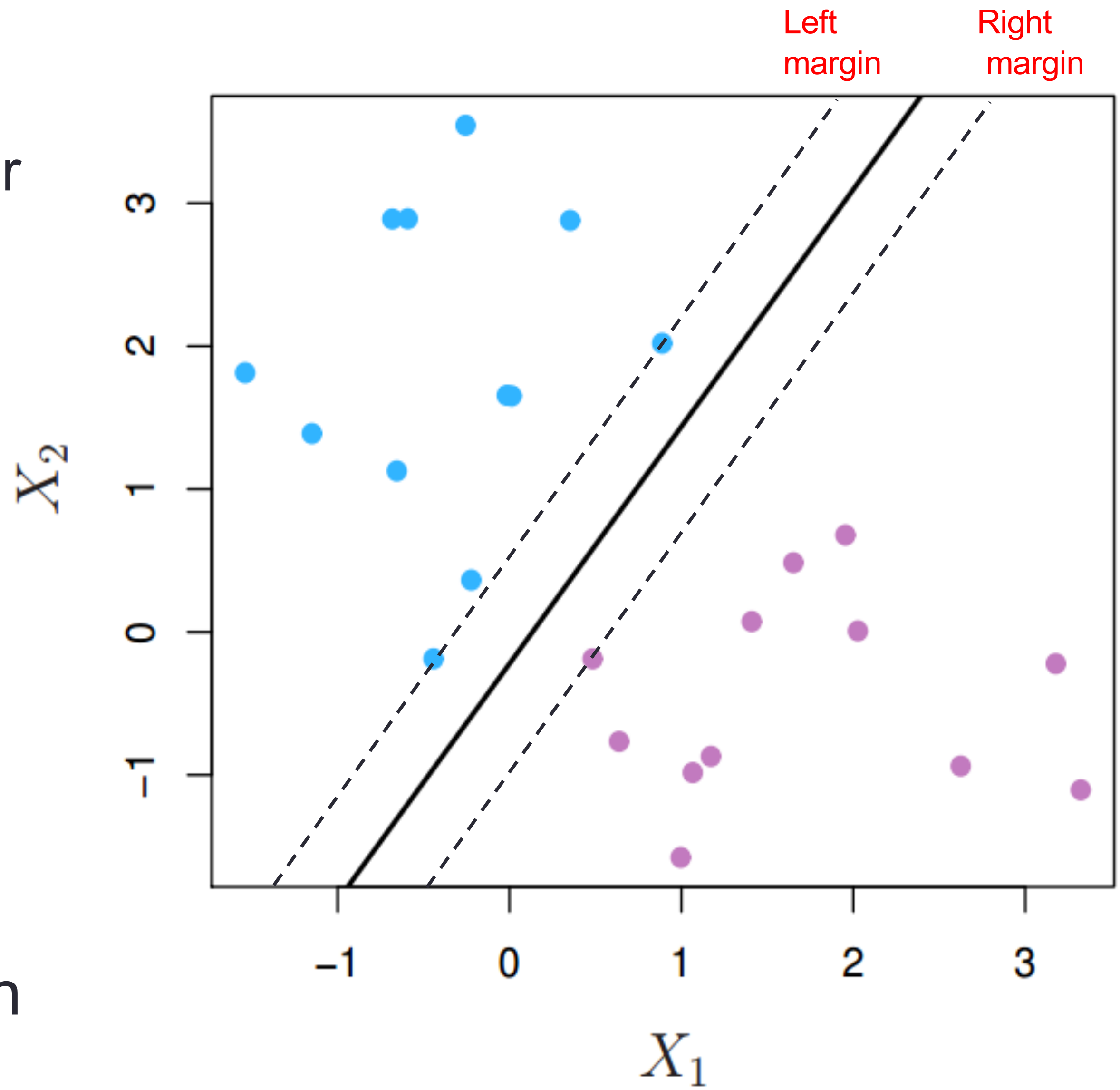
Data points **on** the left or right margins are called **support vectors**



Hard Margin Classifier

Hard Margin Classifier works if the data is linearly separable, only.

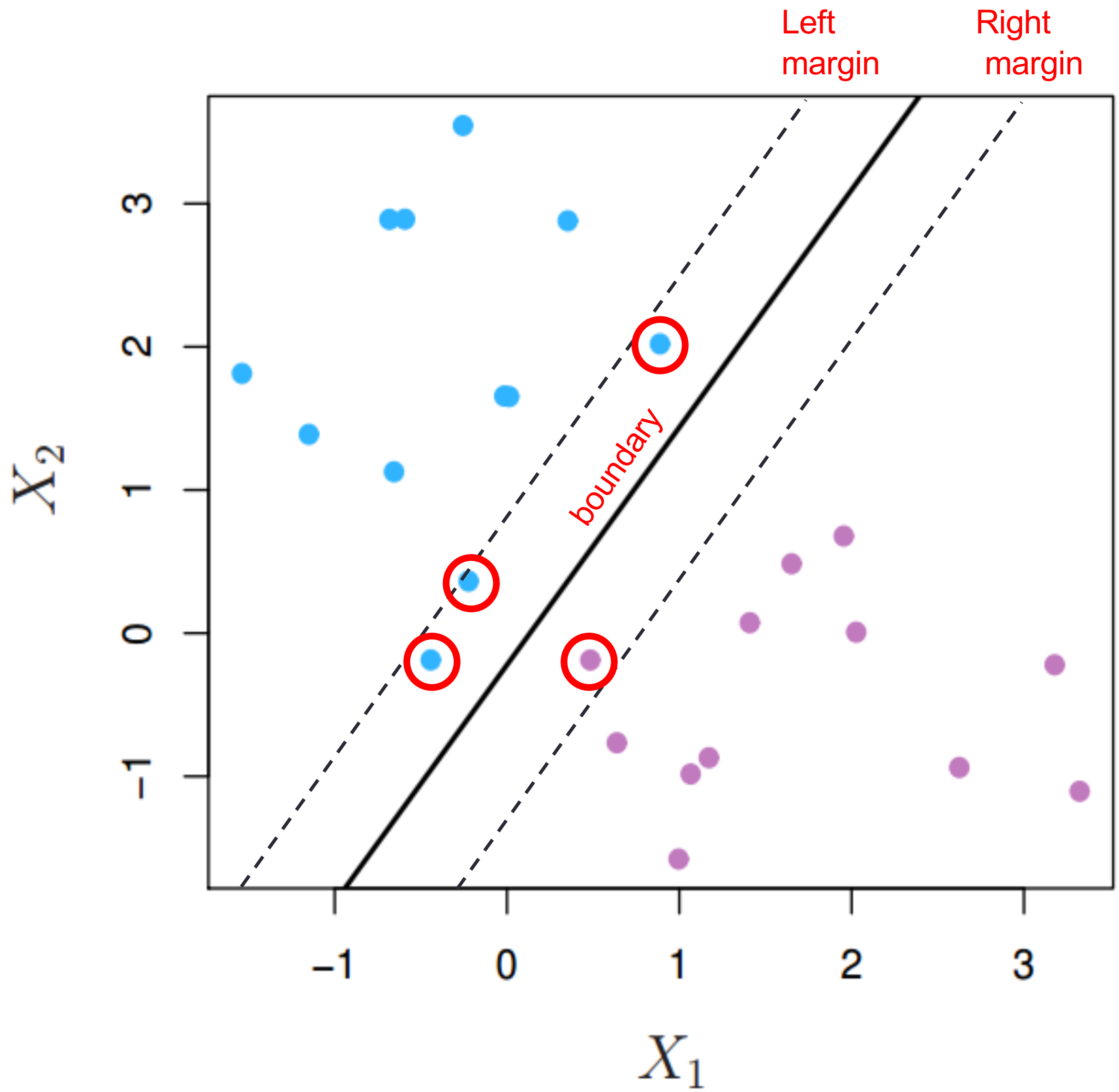
Otherwise the optimization model concludes that there is no feasible solution



Soft Margin Classifier

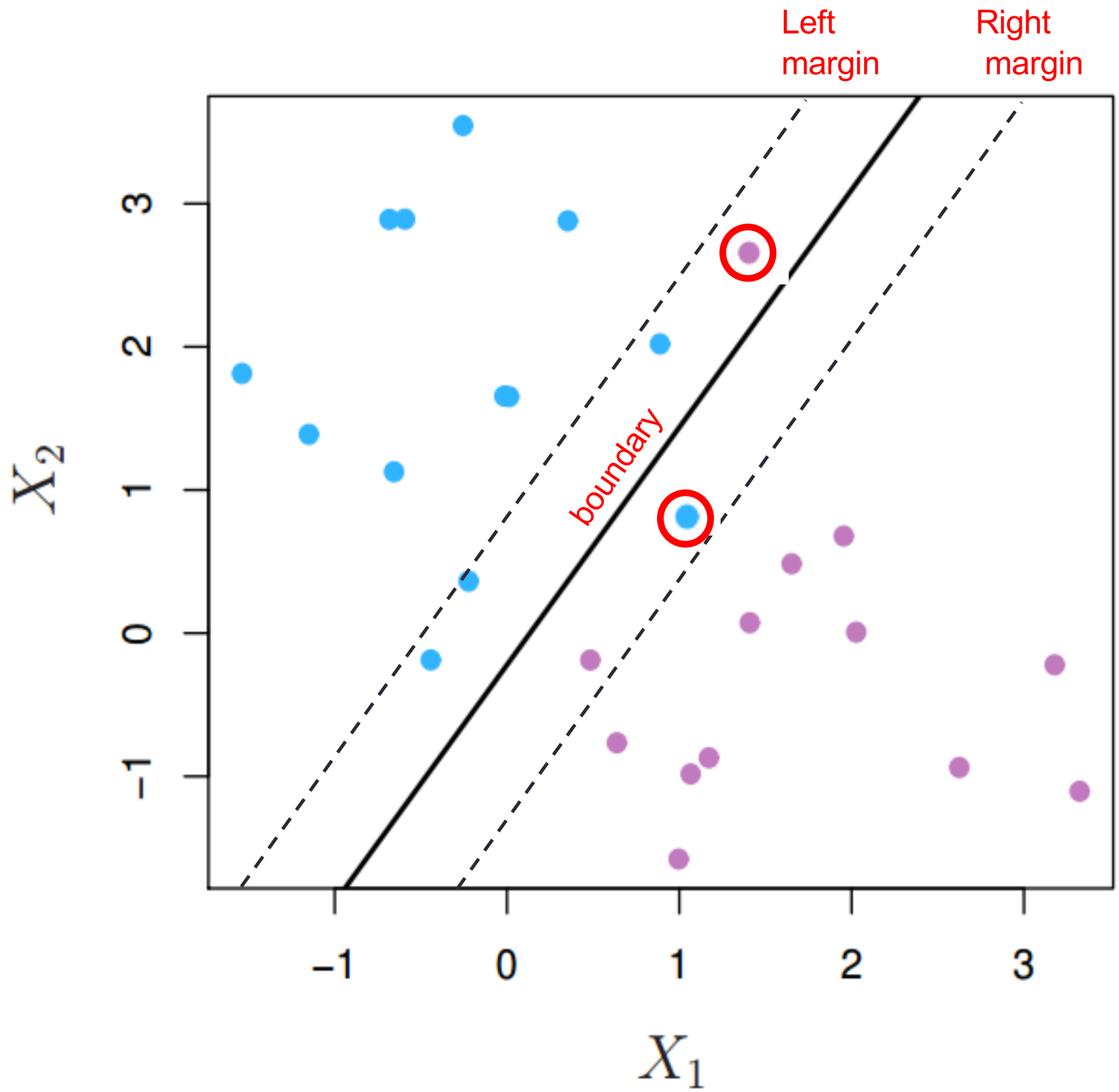
Soft Margin Classifier

- Allow for some margin violations, but still on the correct side of the boundary



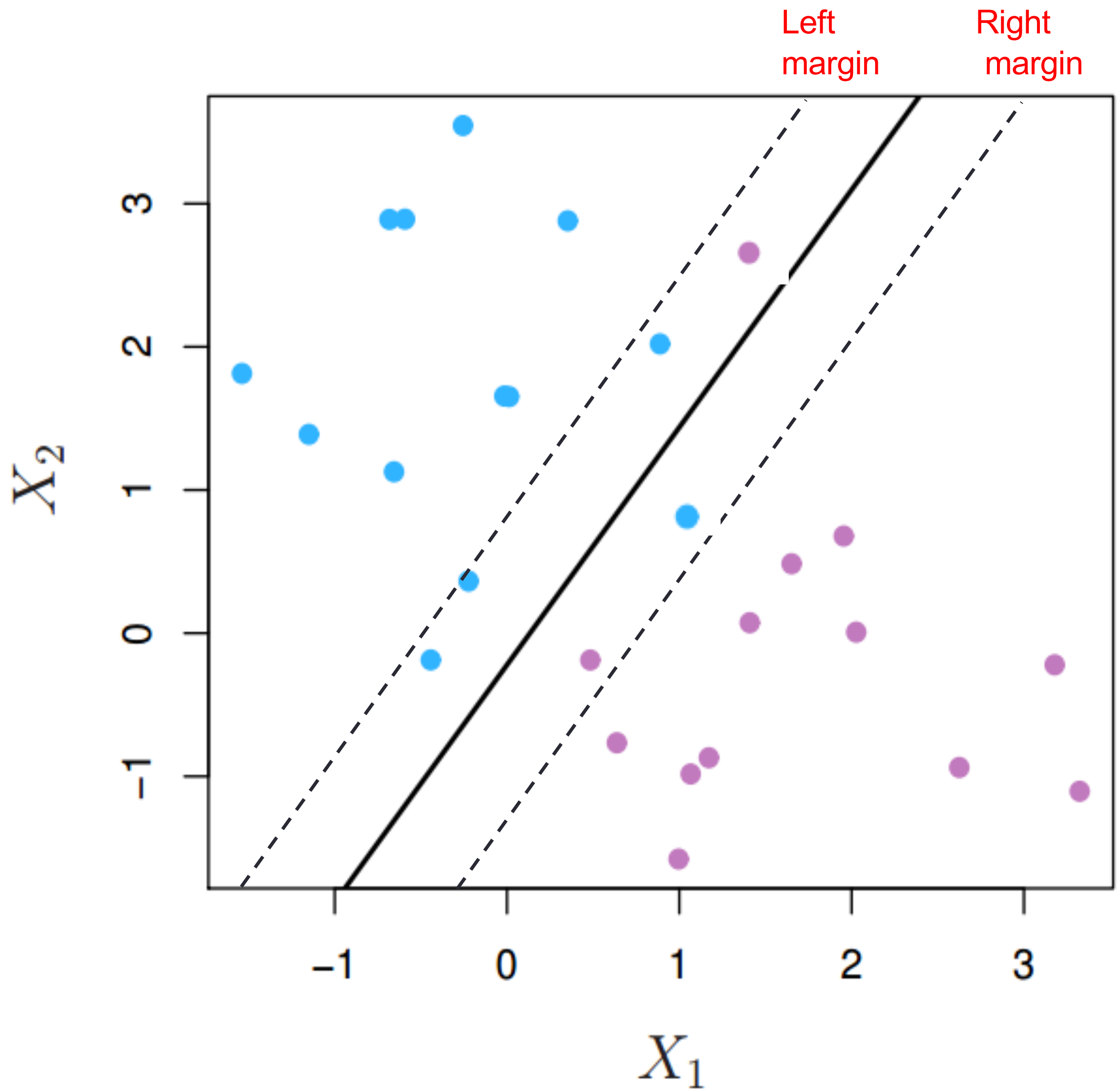
Soft Margin Classifier

- Allow for some boundary violations (on the wrong side of the boundary)



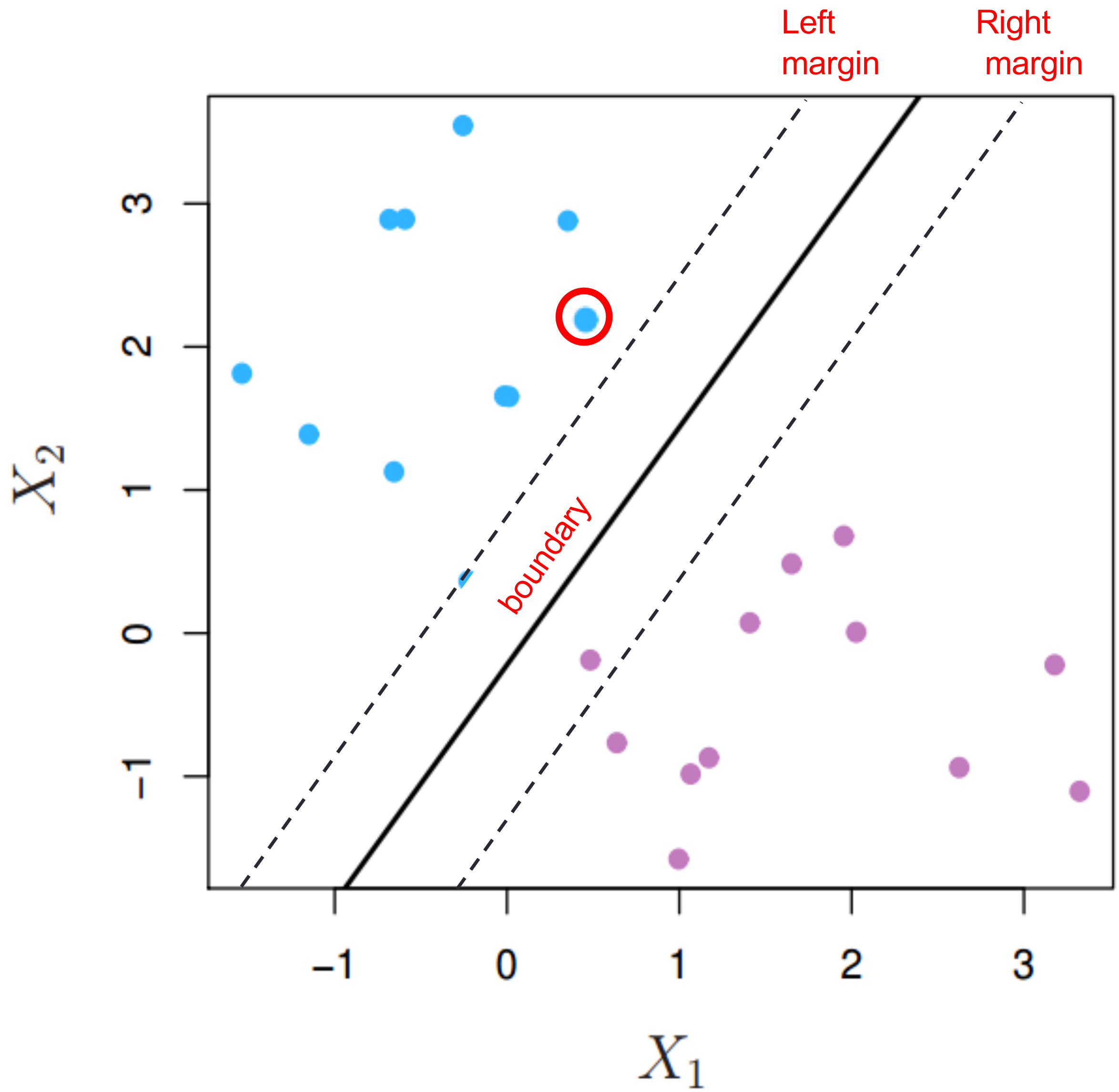
Soft Margin Classifier

- Allow for some margin violations
- Allow for some boundary violations
- But still want to have the *street* as wide as possible



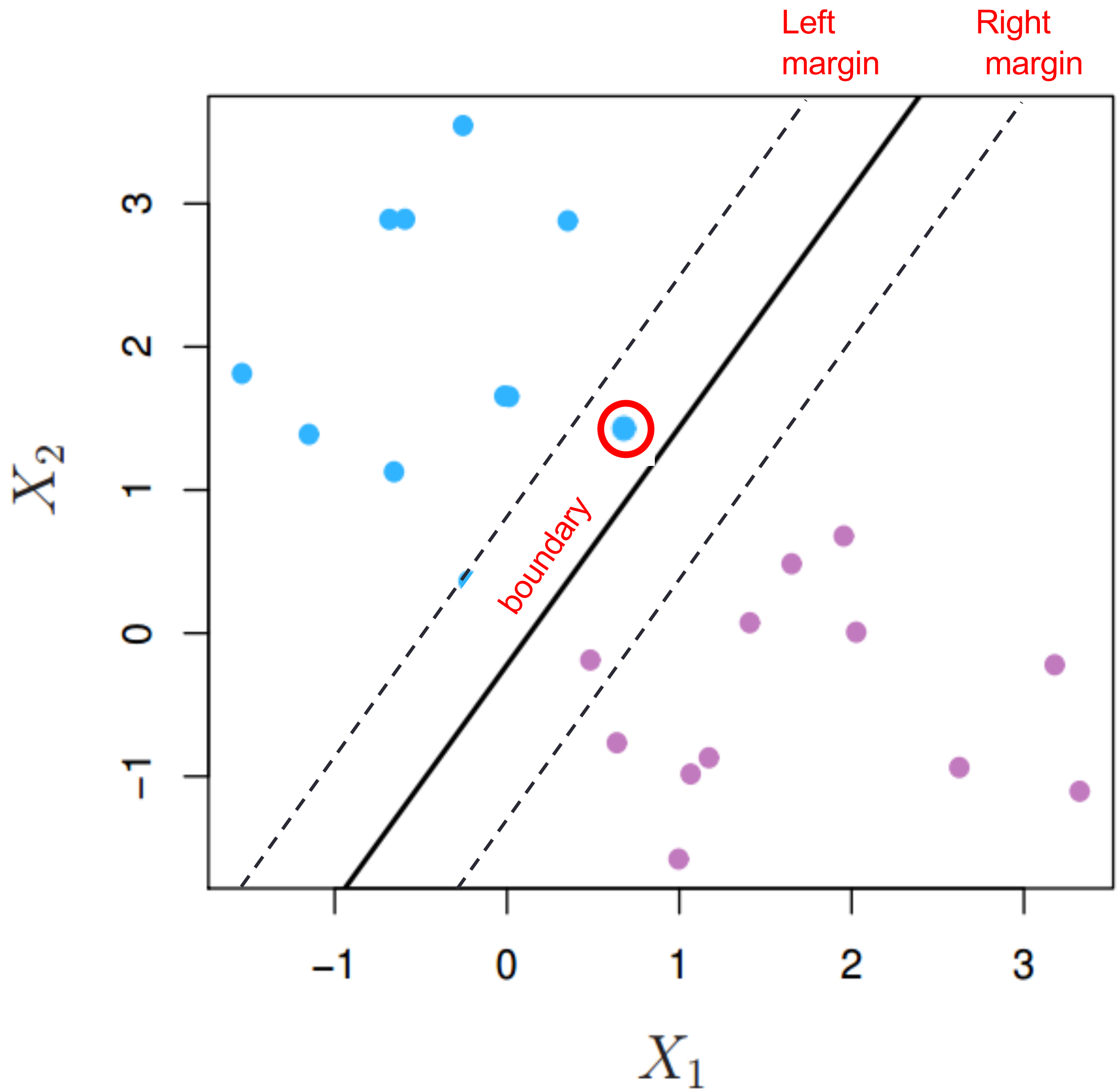
Soft Margin Classifier

- Let ζ_i be the slack of i^{th} observation
- If $\zeta_i = 0$ then the obs is on the right side



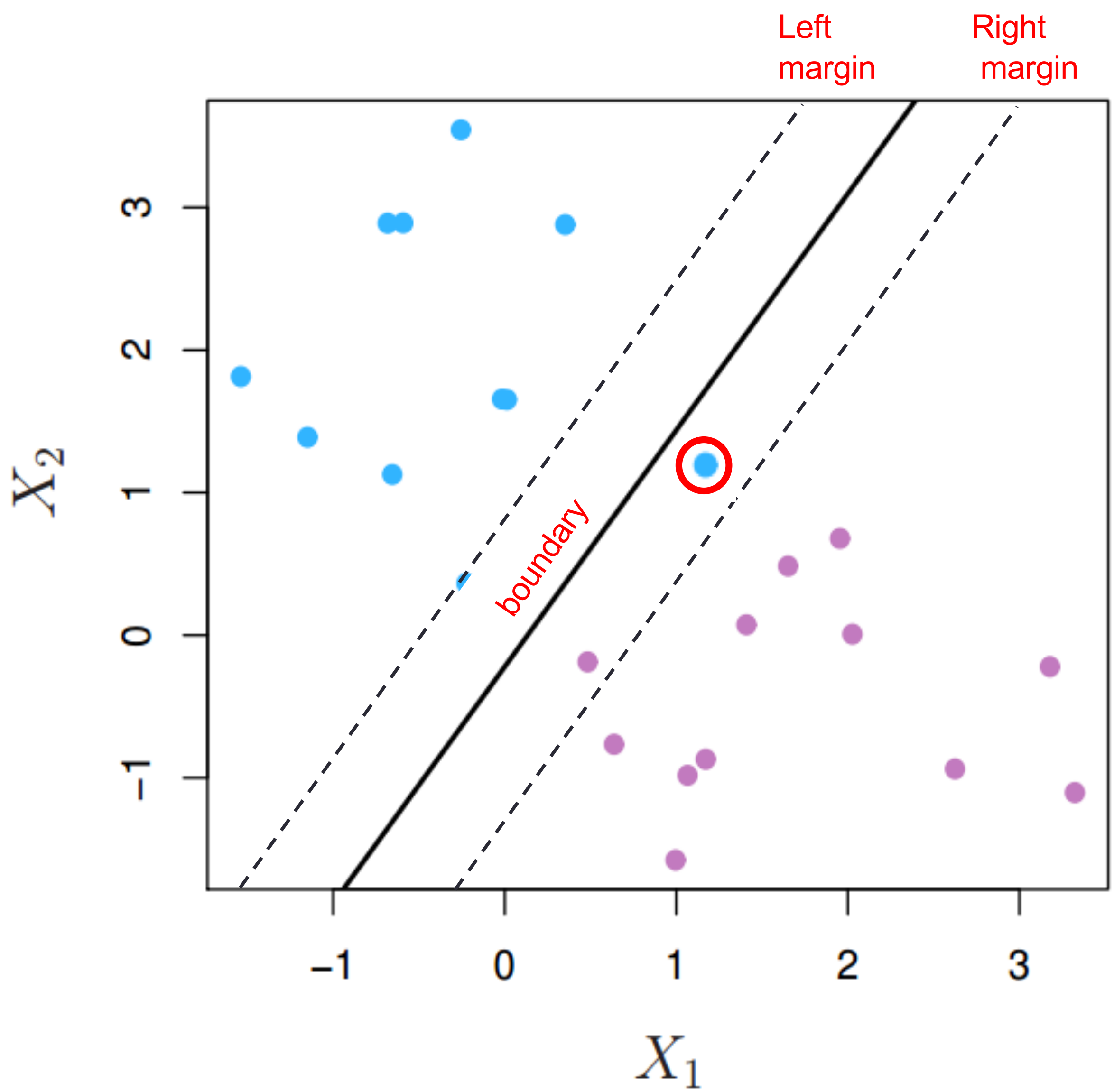
Soft Margin Classifier

- Let ζ_i be the slack of i^{th} observation
- If $\zeta_i = 0$ then the obs is on the right side
- If $0 < \zeta_i < 1$ then the obs has violated its margin



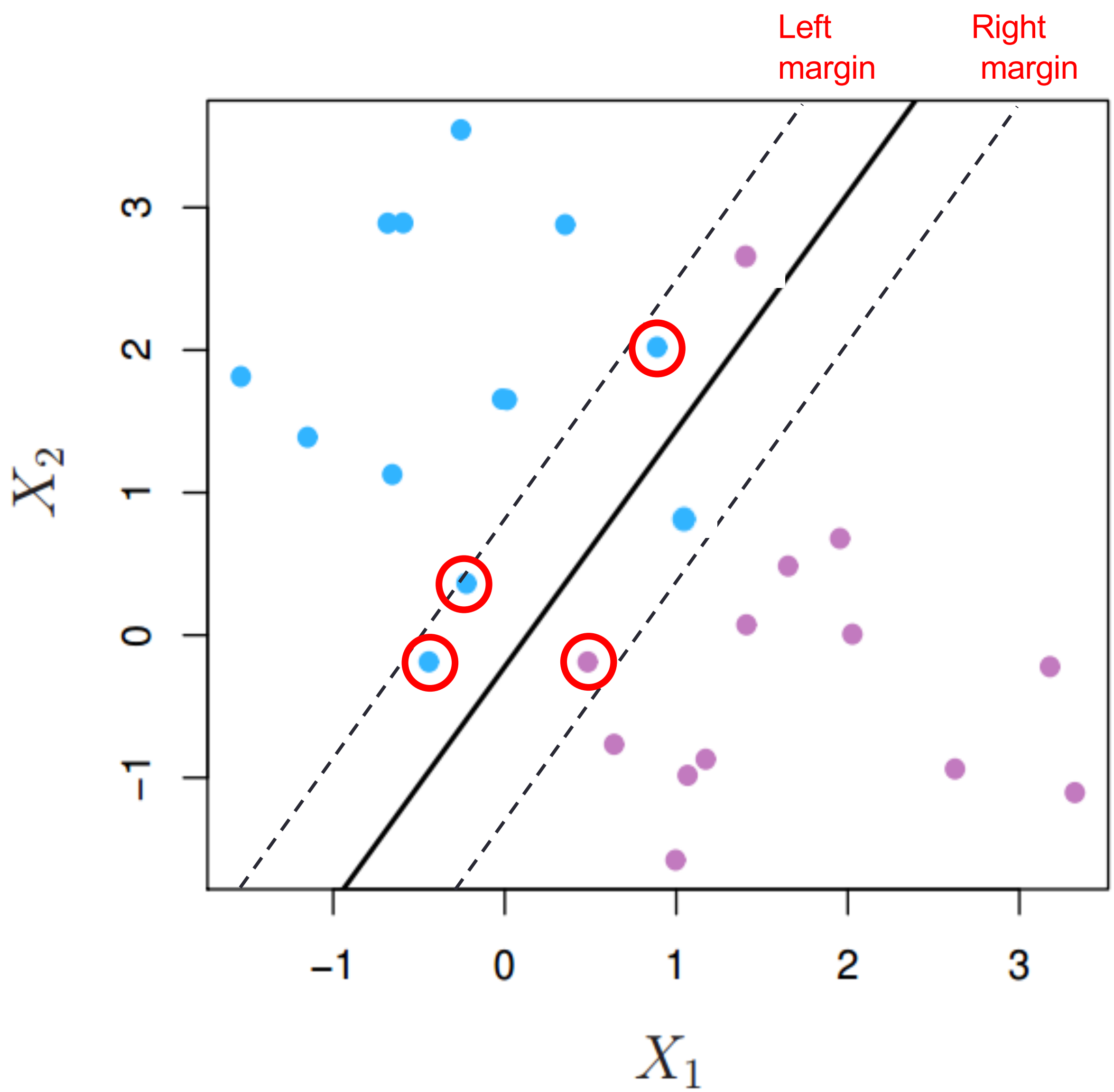
Soft Margin Classifier

- Let ζ_i be the slack of i^{th} observation
- If $\zeta_i = 0$ then the obs is on the right side
- If $0 < \zeta_i < 1$ then the obs has violated its margin
- If $\zeta_i > 1$ then the obs has violated the boundary



Soft Margin Classifier

- The Soft Margin Classifier is also called the Support Vector Classifier
- Observations that violated their margin but not the boundary are the support vectors



Support Vector Classifier - Fundamentals

Support Vector Classifier

Predicts the category of y with a linear decision function h

$$h = \mathbf{w}^T \cdot \mathbf{x} + b = w_1 x_1 + \cdots + w_p x_p + b$$

$$\hat{y} = \begin{cases} 0 & \text{if} \quad h < 0 \\ 1 & \text{if} \quad h \geq 0 \end{cases}$$

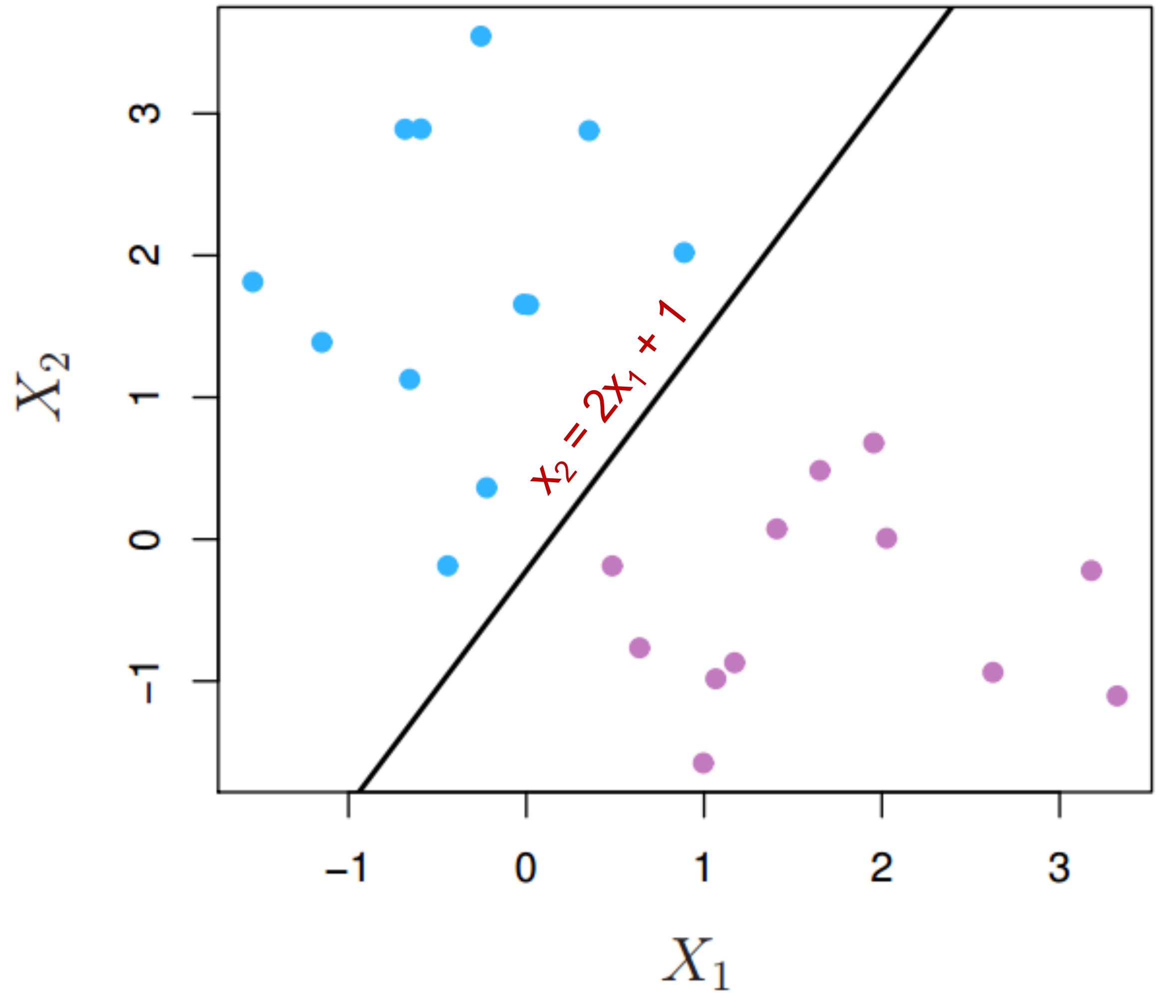
$w_1, \ldots, w_p$ are the feature weights,

b is the bias term

Linearly separable dataset

Suppose boundary is

$$x_2 = 2x_1 + 1$$



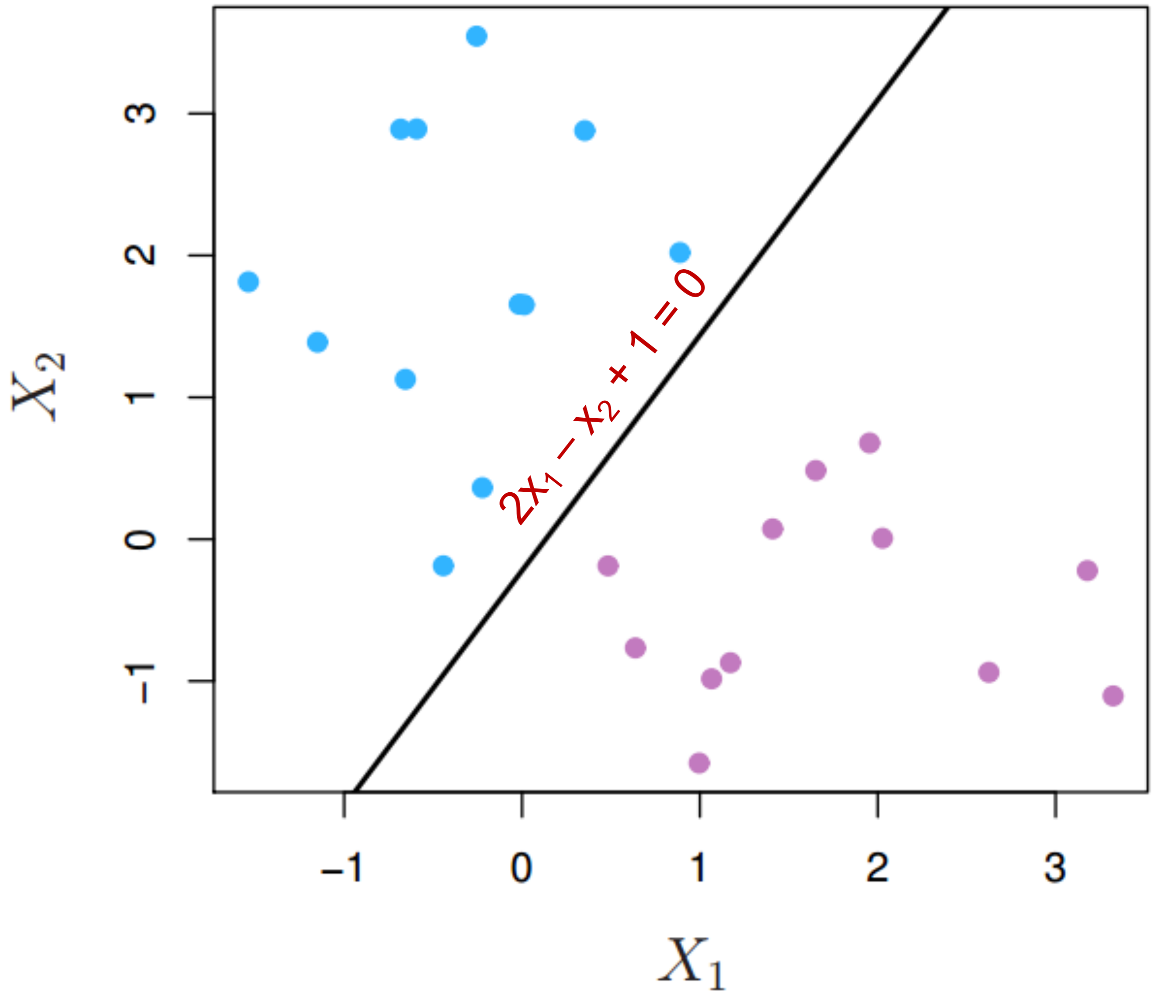
Linearly separable dataset

Suppose boundary is

$$x_2 = 2x_1 + 1$$

or

$$2x_1 - x_2 + 1 = 0$$



Linearly separable dataset

Suppose boundary is

$$x_2 = 2x_1 + 1$$

or

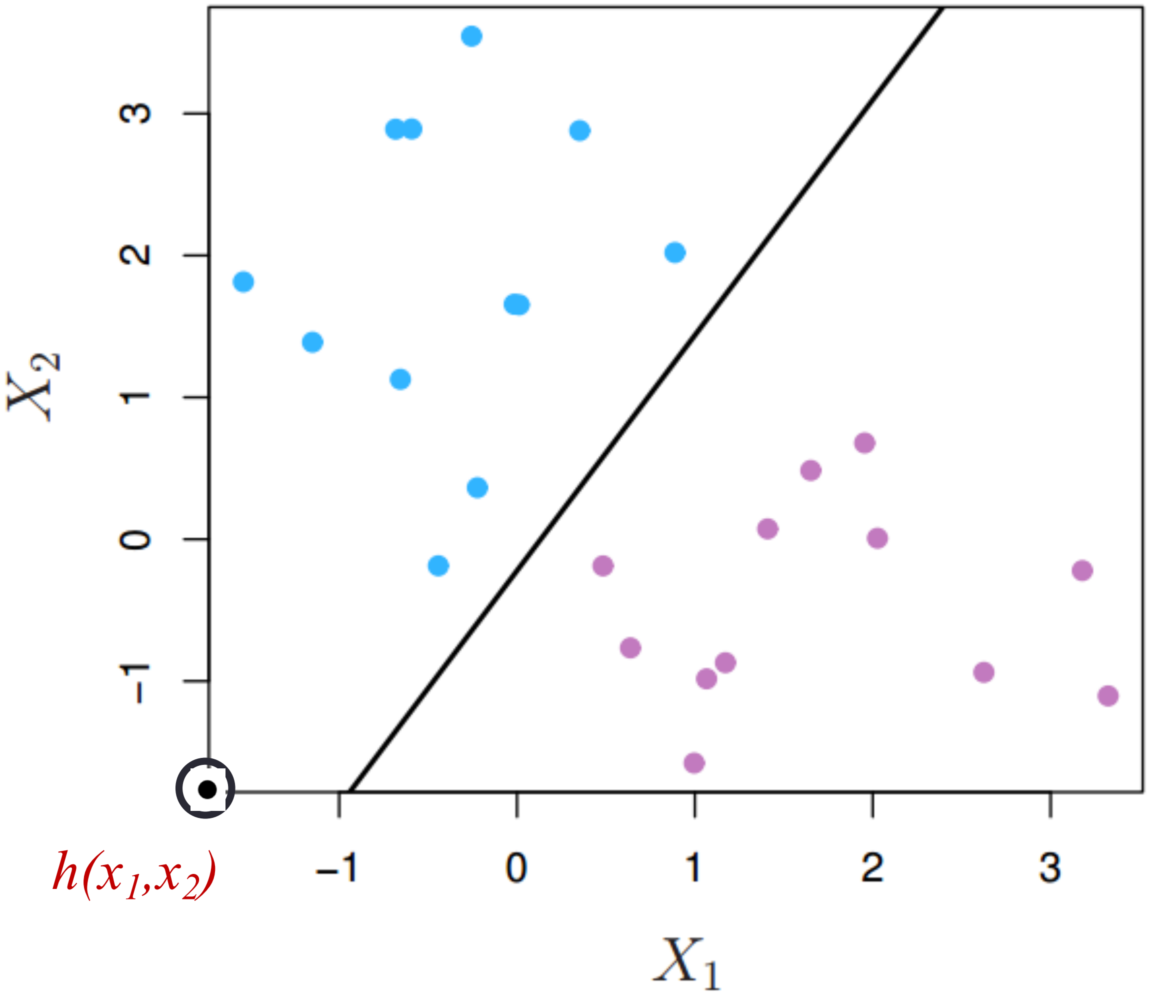
$$2x_1 - x_2 + 1 = 0$$

Define a linear function

$$h(x_1,x_2) = 2x_1 - x_2 + 1$$

and call it

decision function



Linearly separable dataset

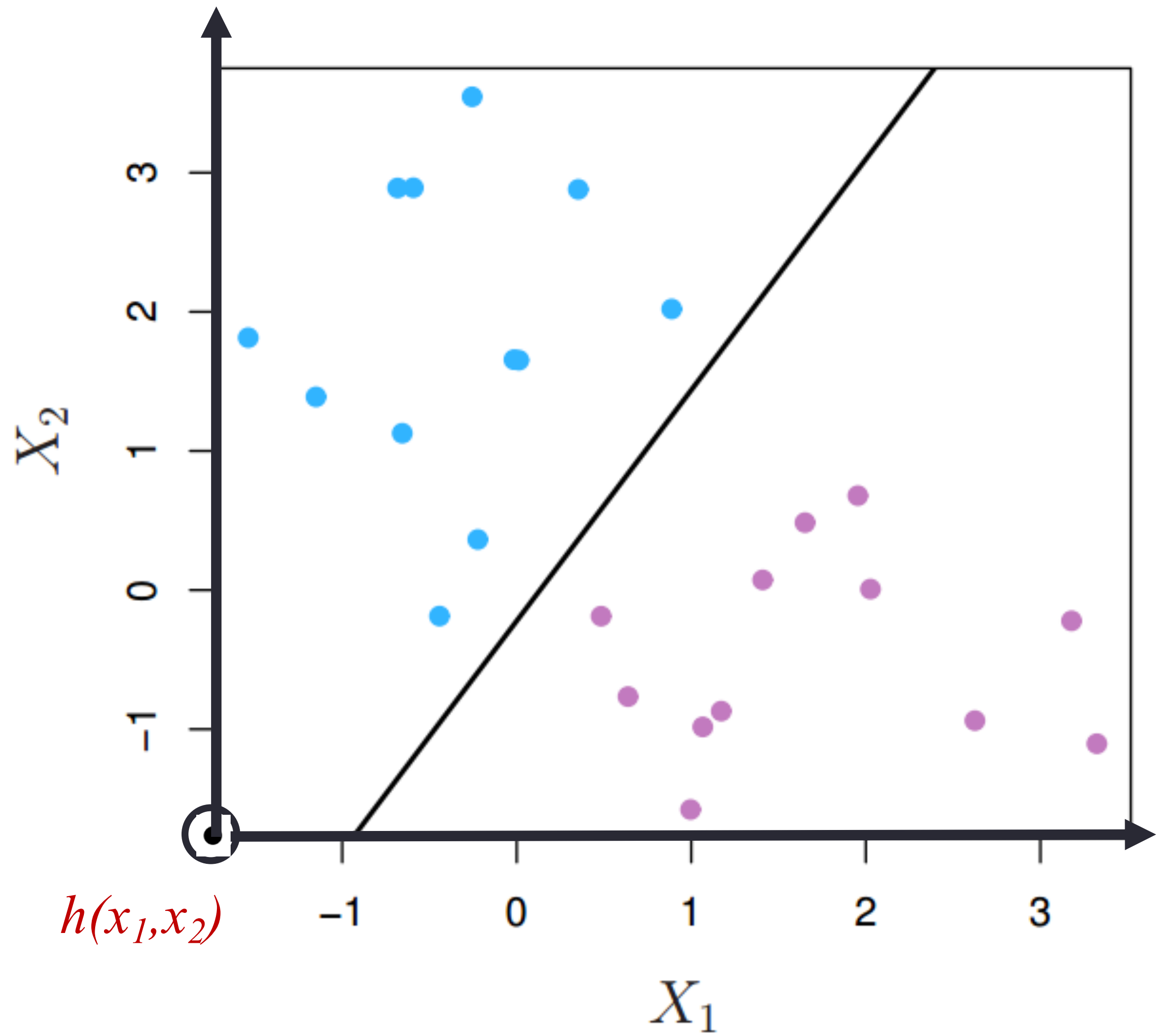
This function

$$h(x_1,x_2) = 2x_1 - x_2 + 1$$

is a plane in

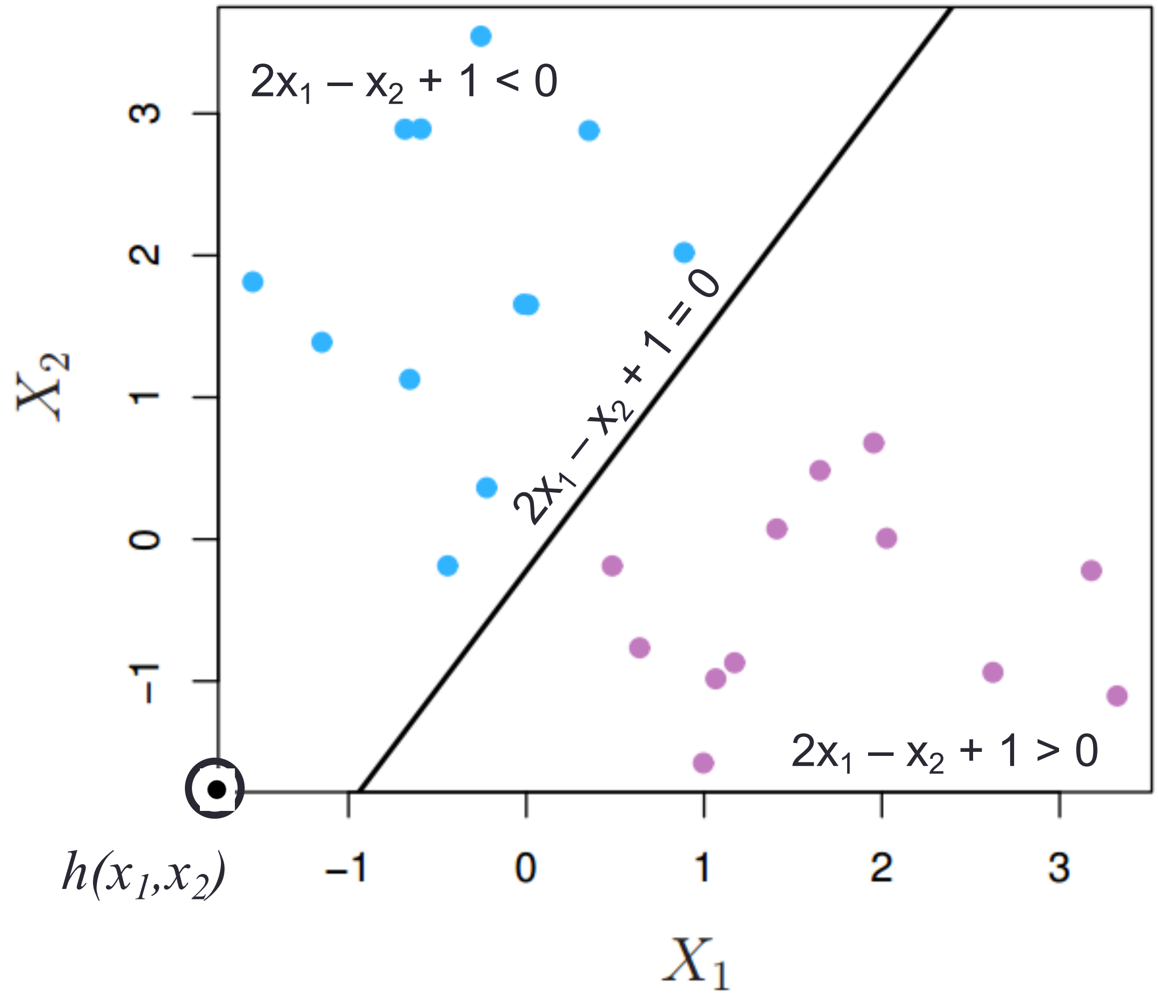
3D space with axes

X_1, X_2, $h(x_1, x_2)$



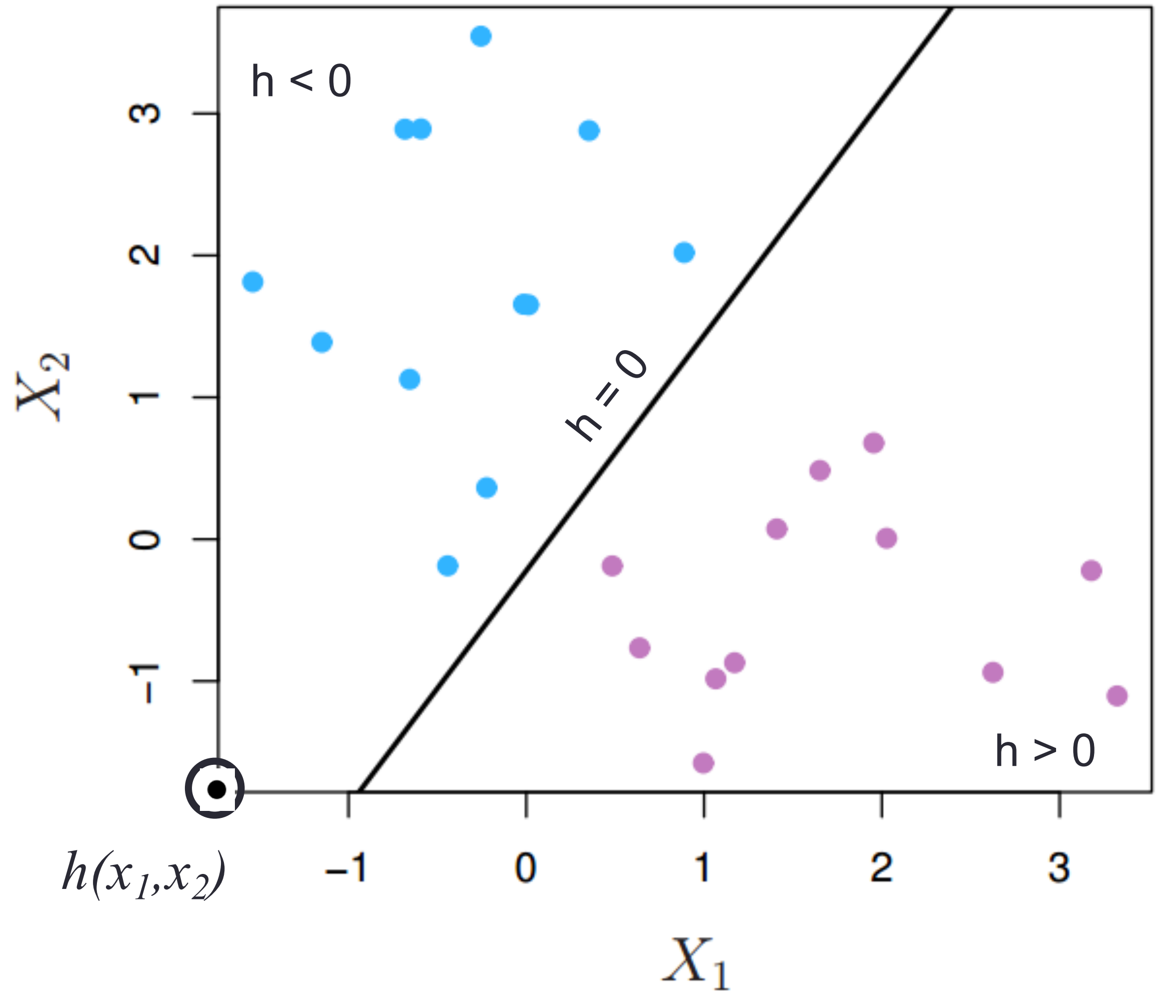
Decision function

$h(x_1,x_2) = 2x_1 - x_2 + 1$



Decision function

$h(x_1,x_2) = 2x_1 - x_2 + 1$



Decision function

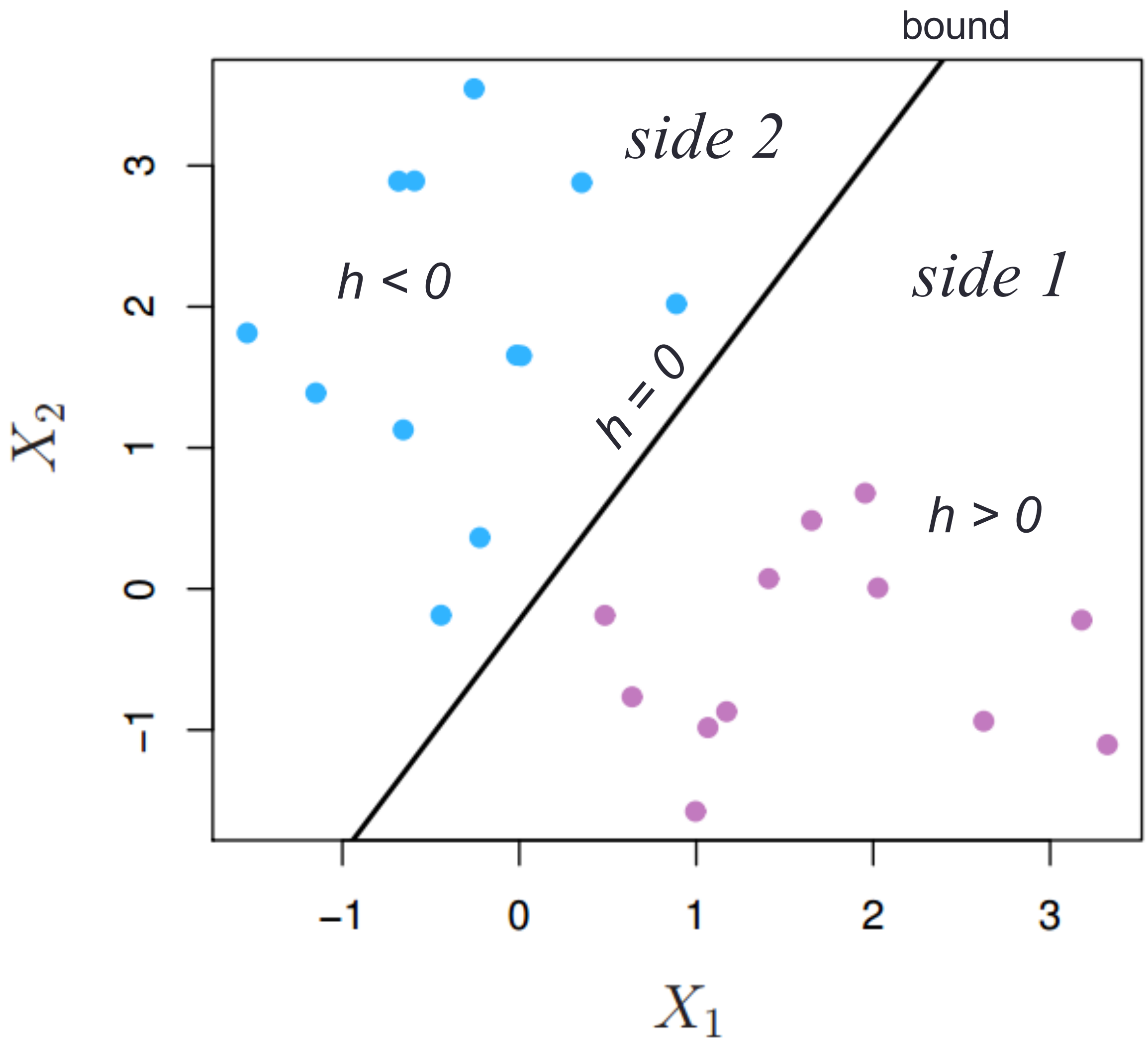
$h(x_1,x_2) = 2x_1 - x_2 + 1$

If

$h = 0$ (x_1,x_2) on bound

$h > 0$ (x_1,x_2) on *side* 1

$h < 0$ (x_1,x_2) on *side* 2



Decision function

$h(x_1,x_2) = 2x_1 - x_2 + 1$

If

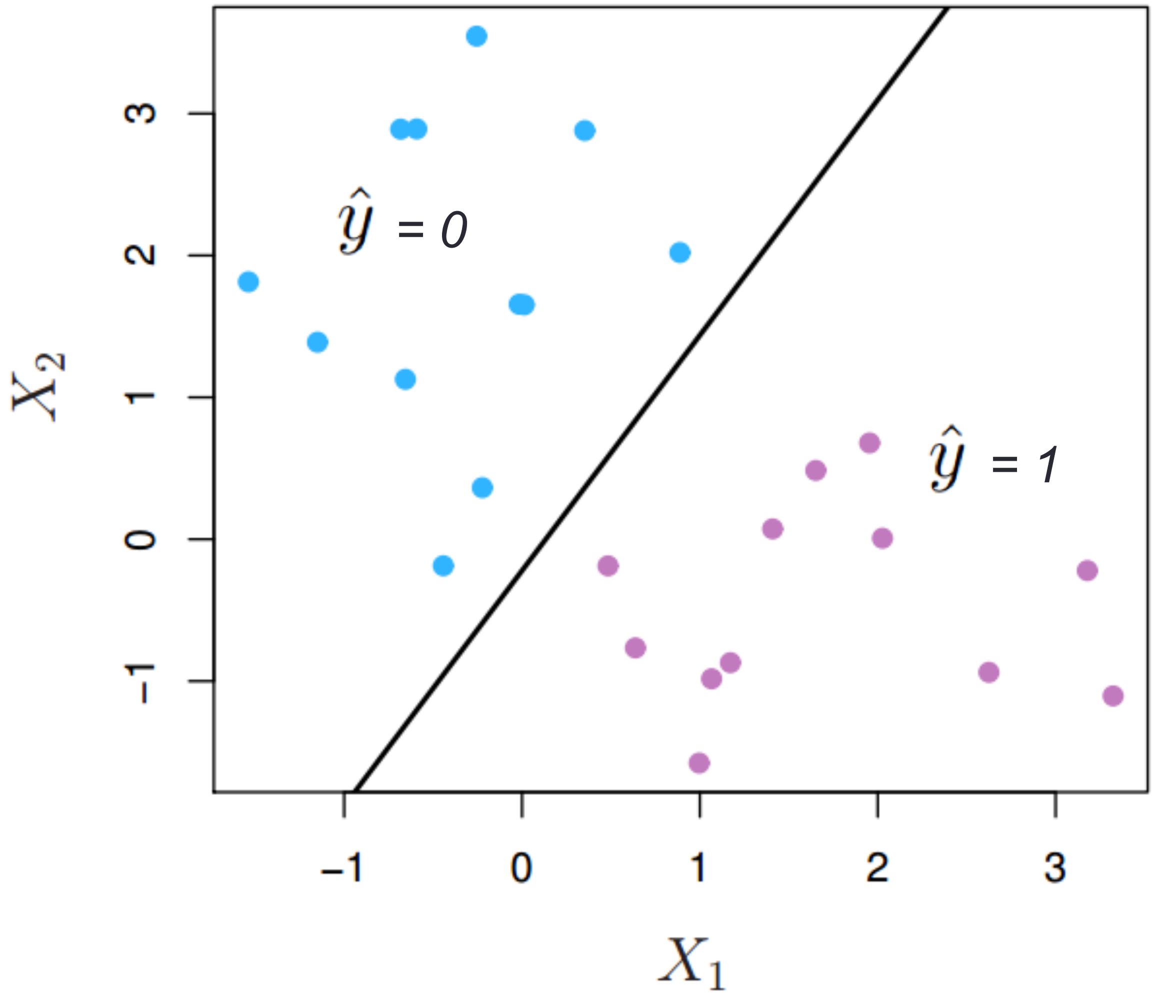
$h = 0$ (x_1,x_2) on bound

$h < 0$ (x_1,x_2) on *side* 1

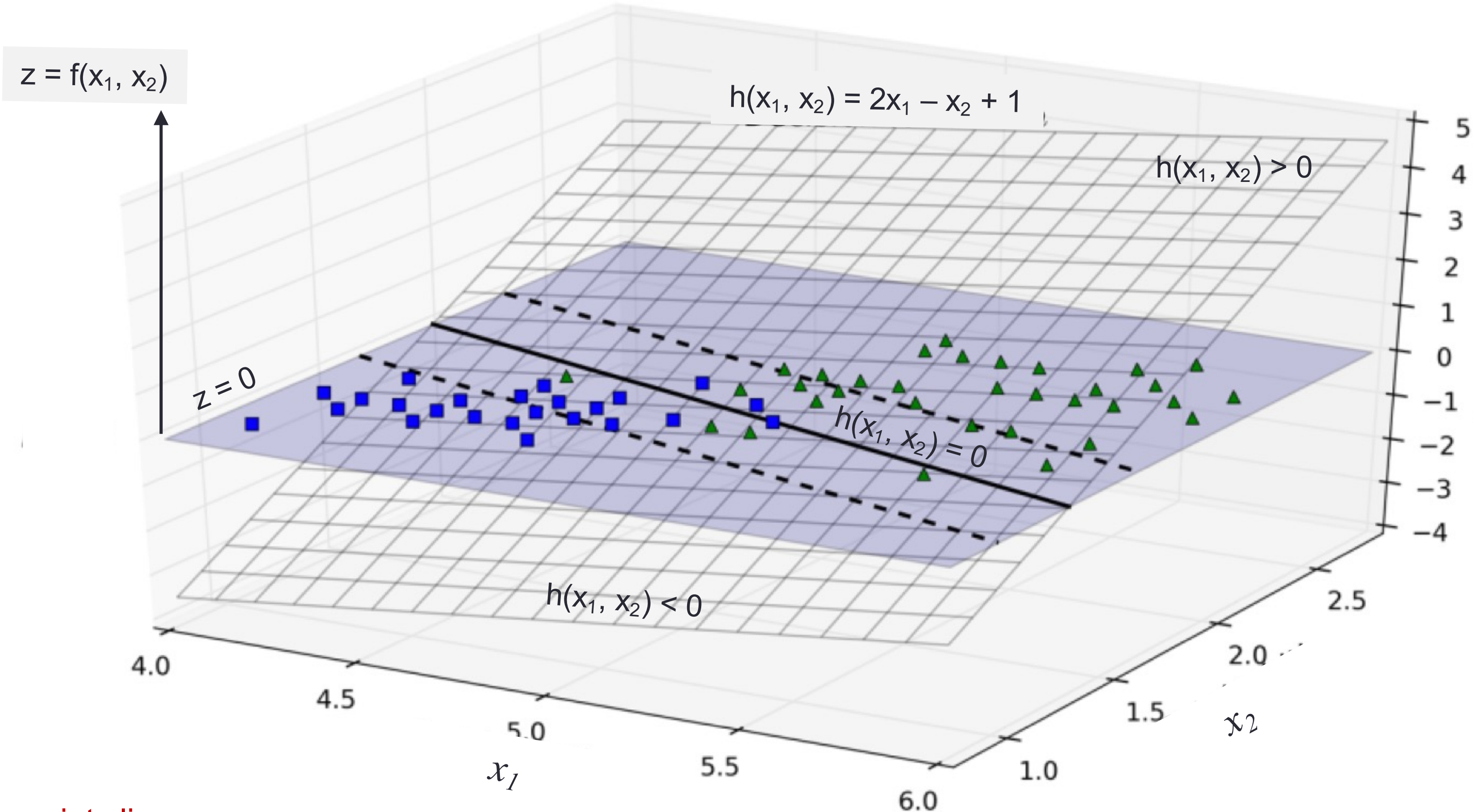
$h > 0$ (x_1,x_2) on *side* 2

Let

$$\hat{y} = \begin{cases} 0 & \text{if} \quad h < 0 \\ 1 & \text{if} \quad h \geq 0 \end{cases}$$



Decision function



All data points lie on
the plane $k(x_1, x_2) = 0$

Decision function

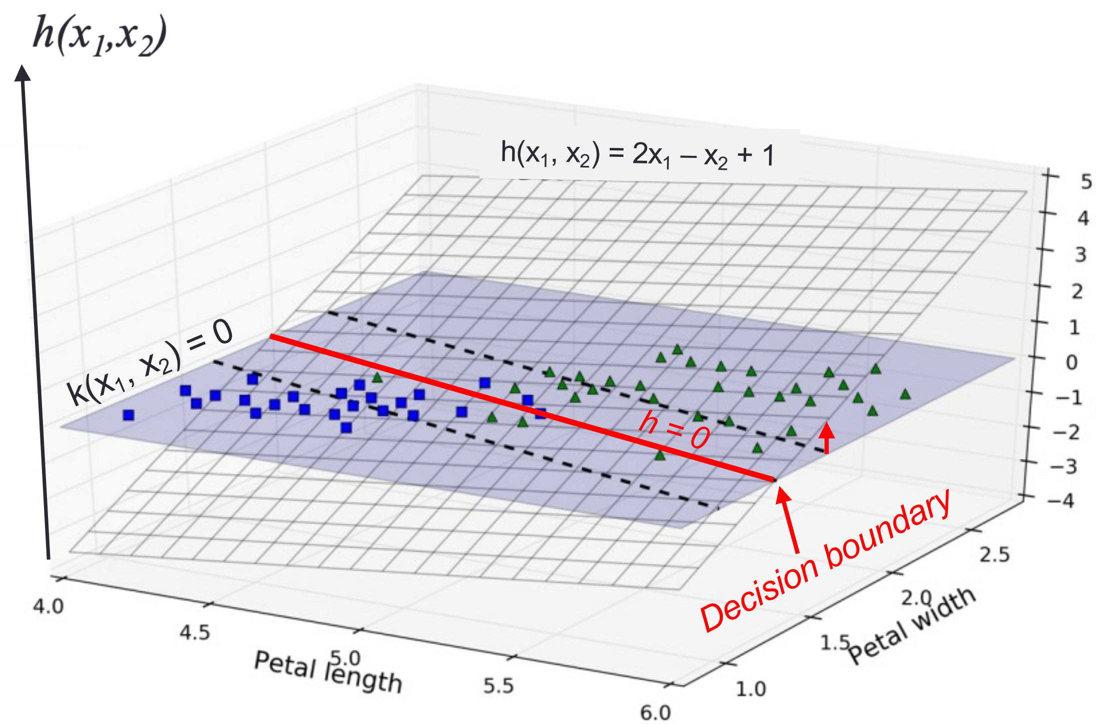
Decision boundary is the set of points

- on the plane $h(x_1,x_2)$ with $h = 0$
- at the intersection of the planes

$k(x_1,x_2) = 0$

and

$h(x_1,x_2) = 2x_1 - x_2 + 1$



Decision function

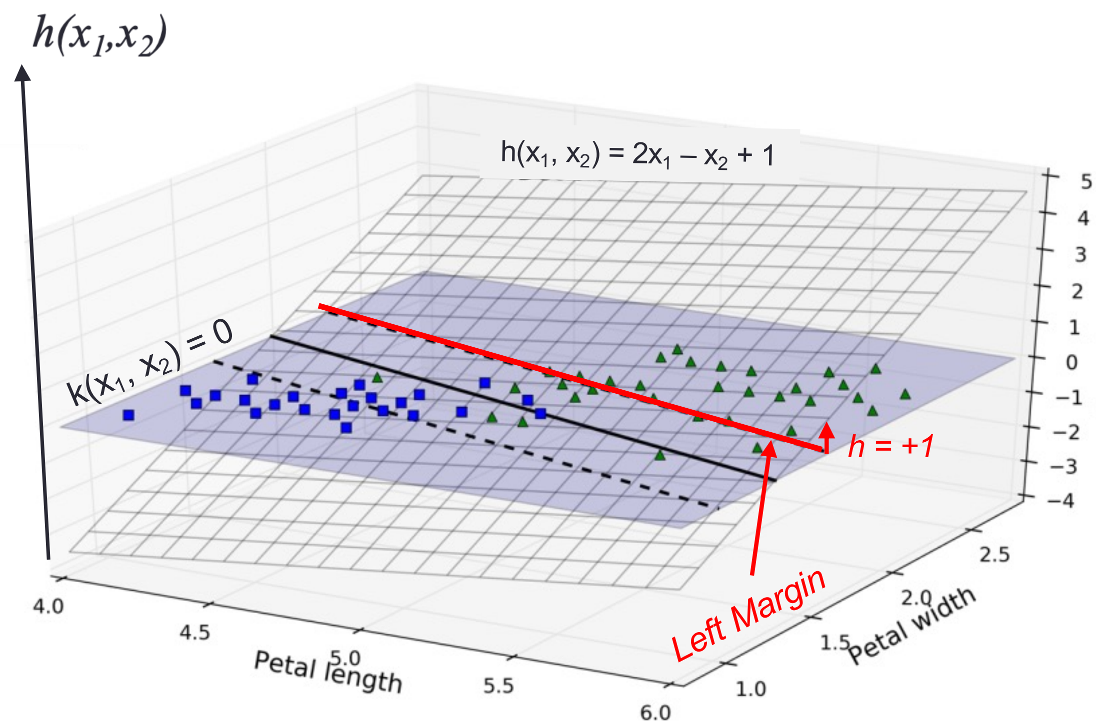
Left Margin is the set of points

- on the plane

  $k(x_1,x_2) = 0$

  with

  $h(x_1,x_2) = 1$



Decision function

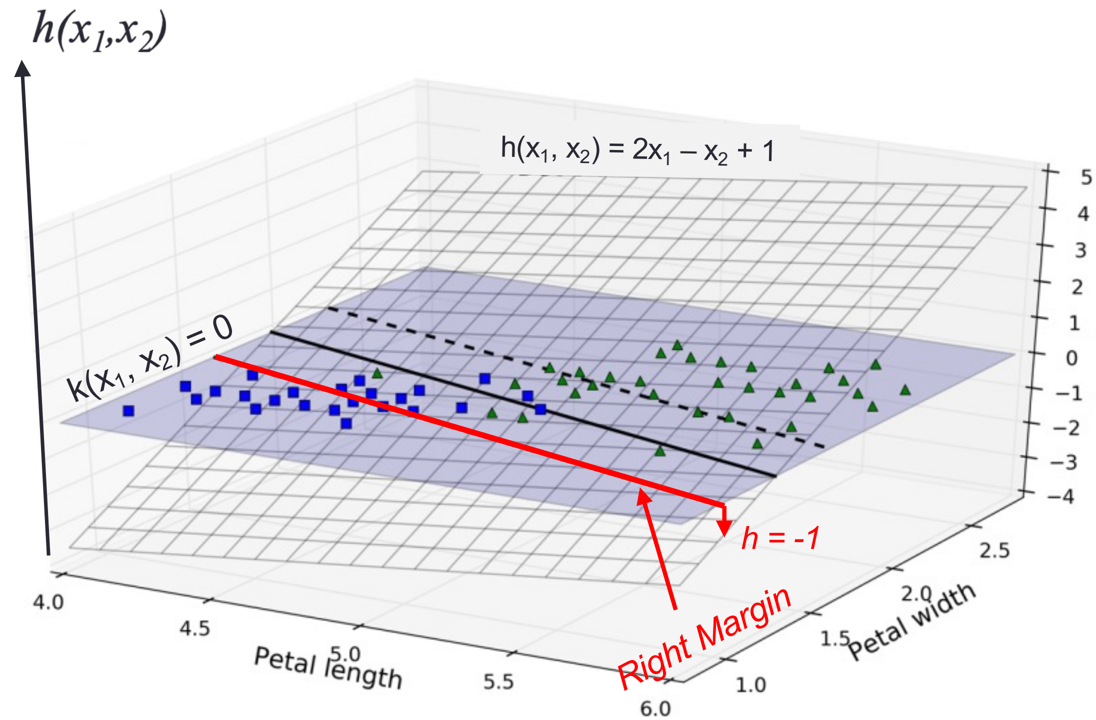
Right Margin is the set of points

- on the plane

  $k(x_1,x_2) = 0$

  and with

  $h(x_1,x_2) = -1$



Decision function

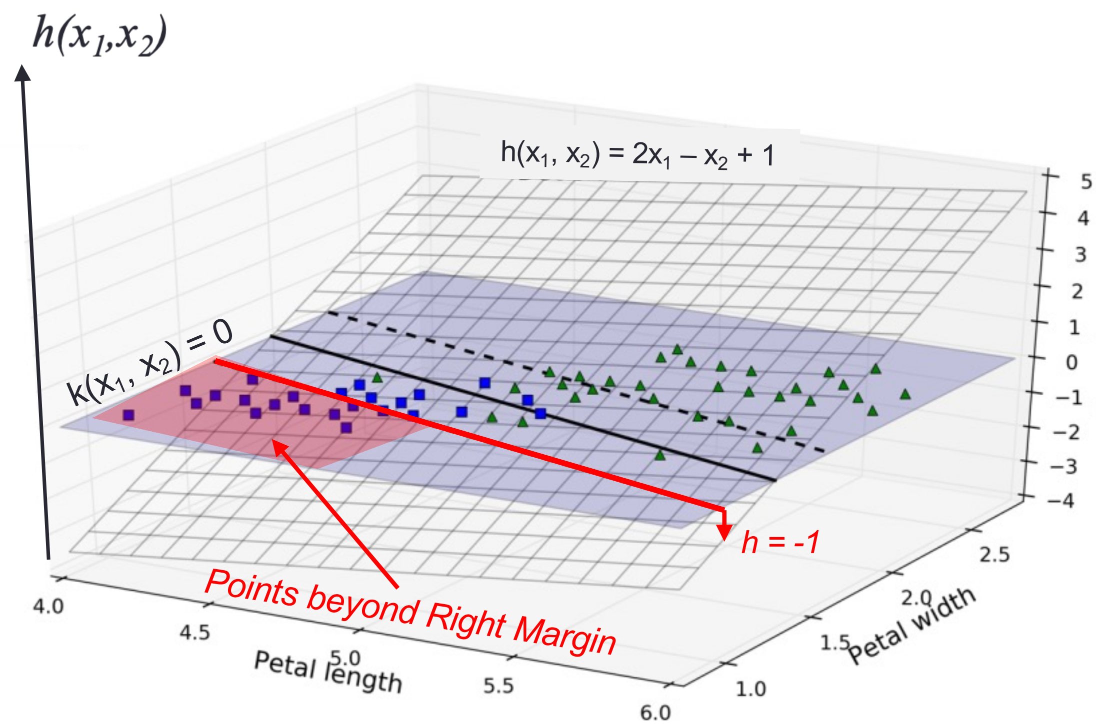
Right Margin is the set of points

- on the plane

  $k(x_1,x_2) = 0$

  and with

  $h(x_1,x_2) = -1$



Decision function

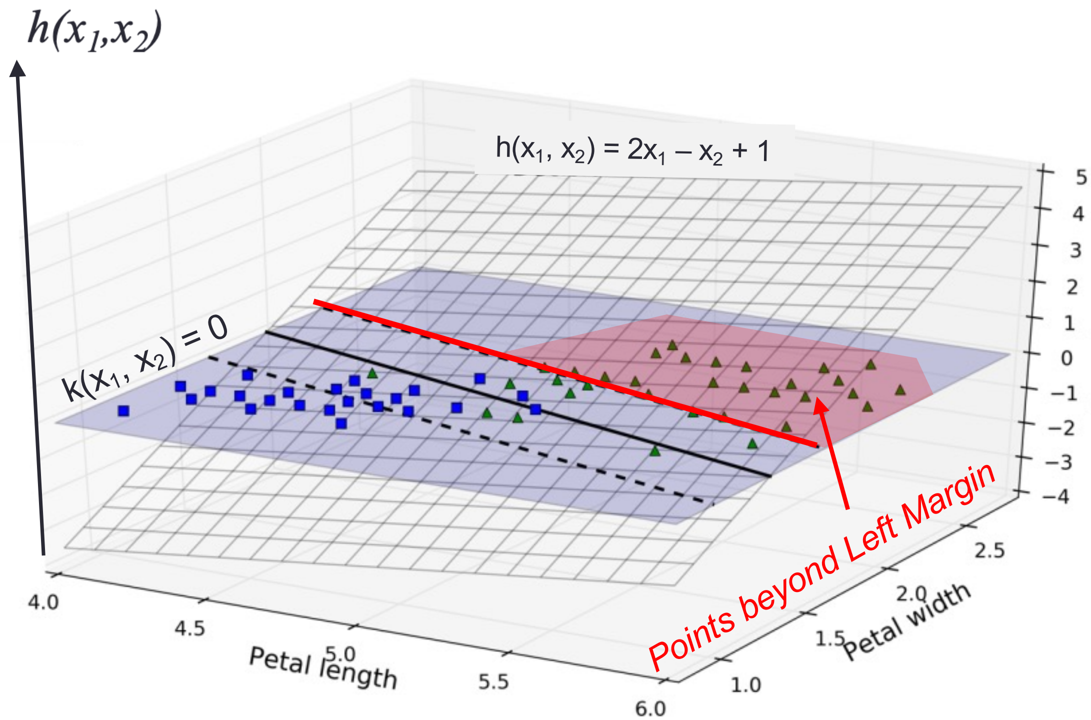
Left Margin is the set of points

- on the plane

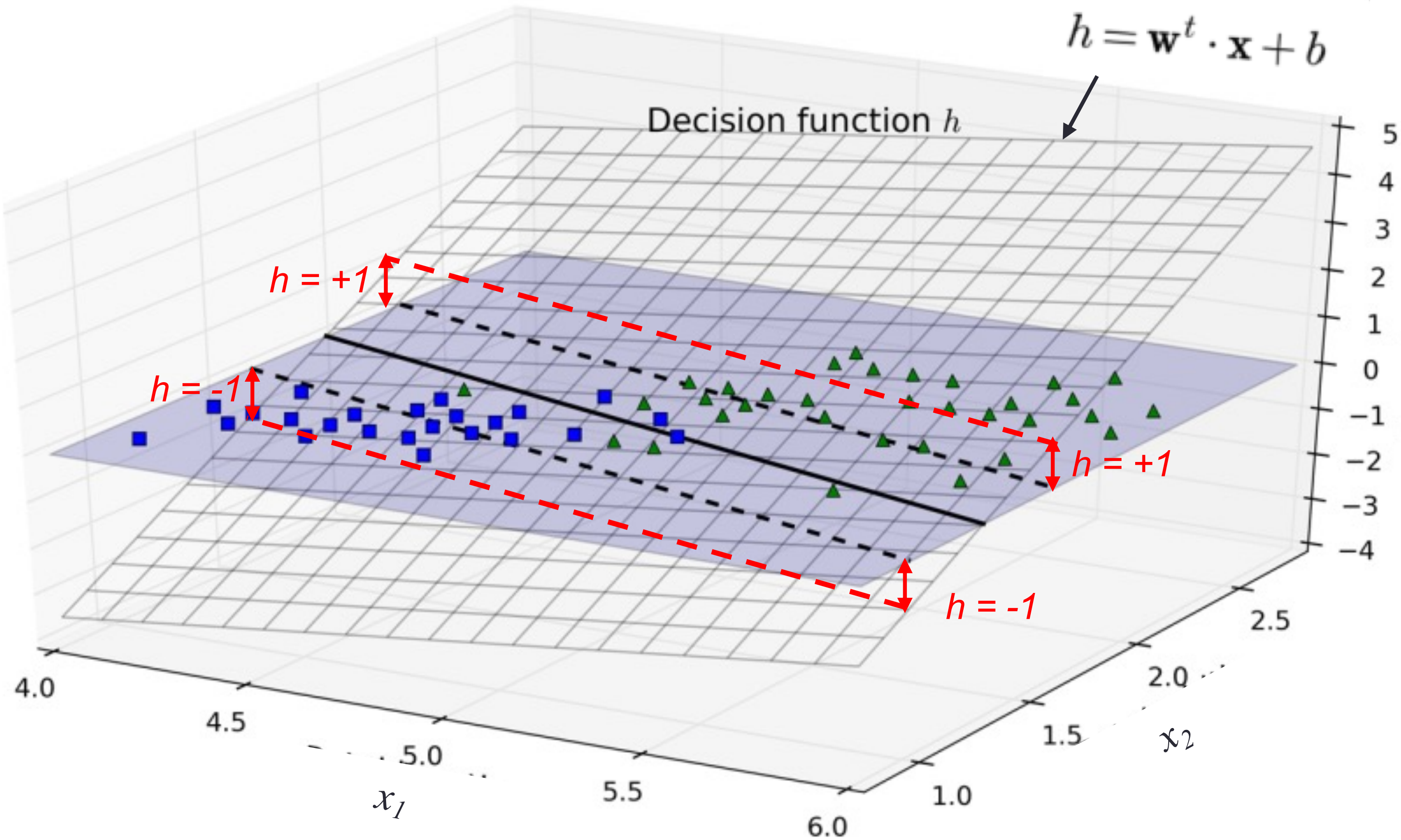
  $k(x_1,x_2) = 0$

  with

  $h(x_1,x_2) = 1$

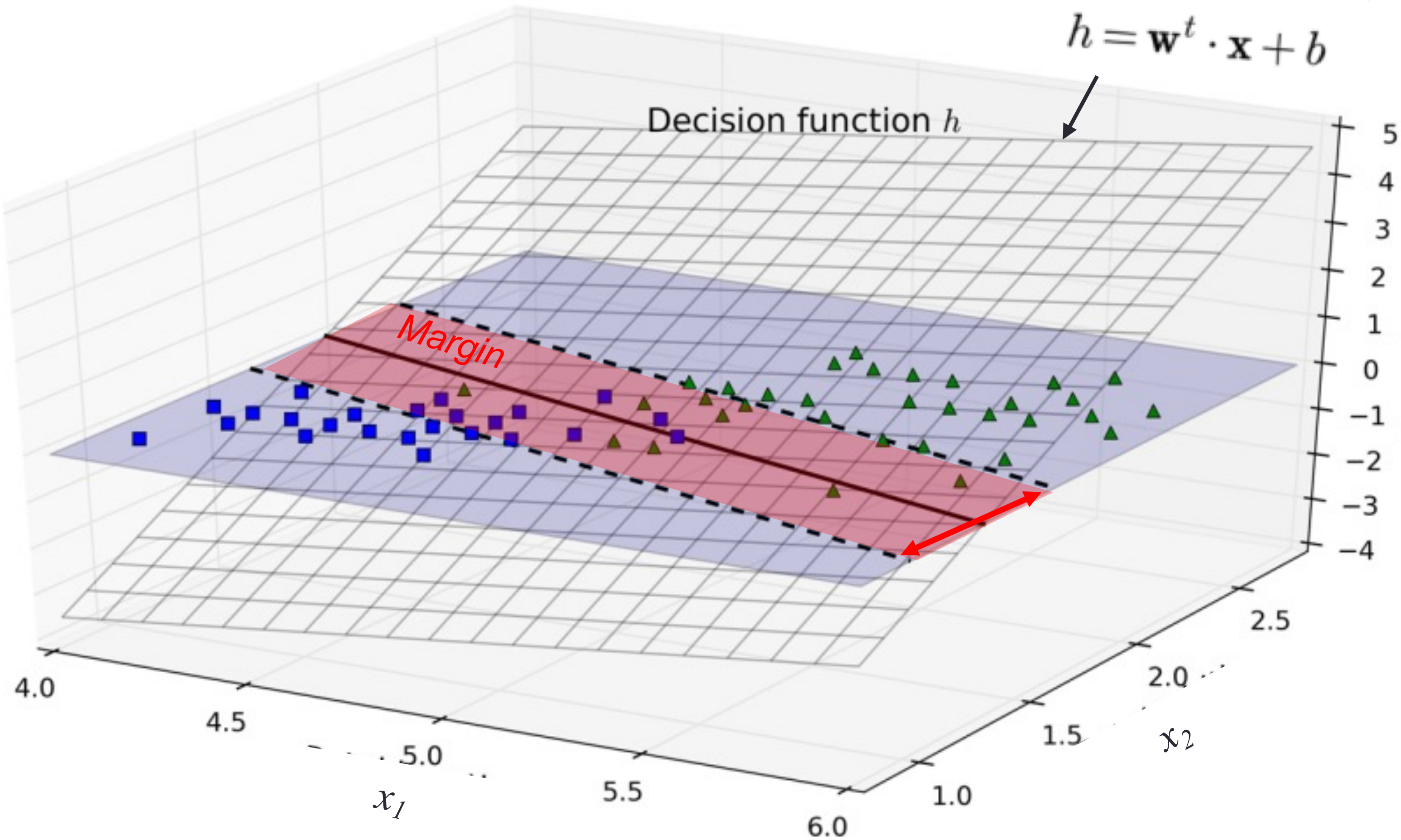


Decision function



Dashed lines show the points where the Decision function value (height) is equal to +1 or -1

Decision function

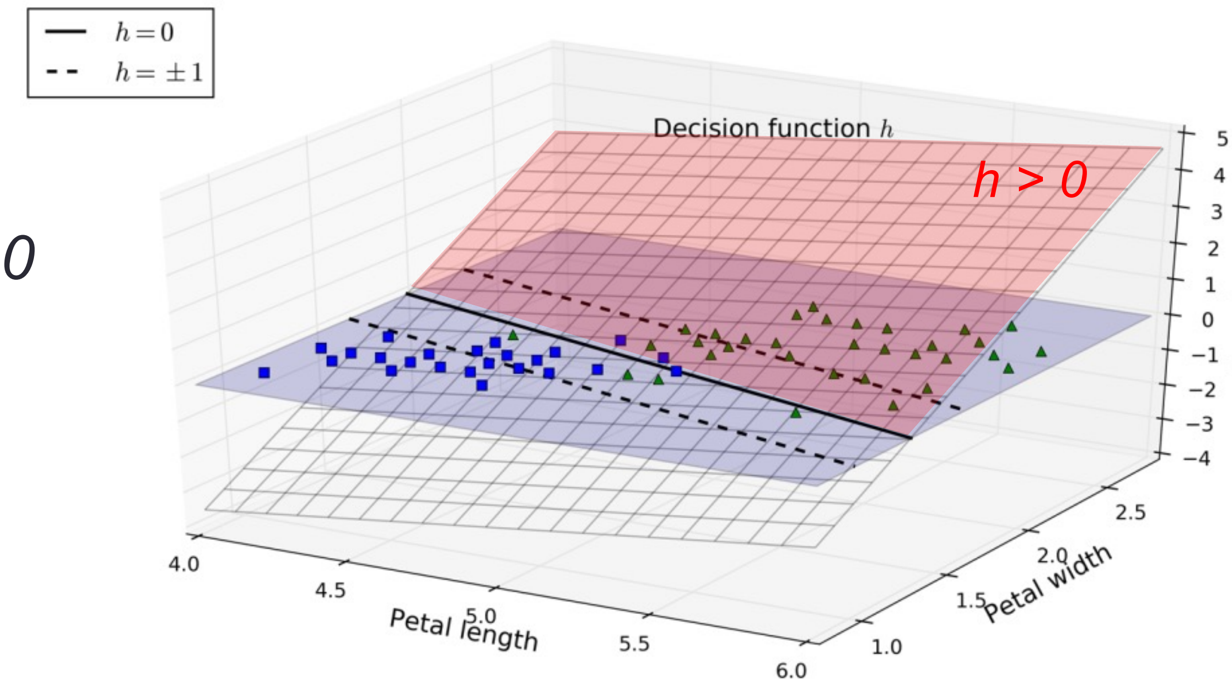


Margin is the region between the dashed lines

Decision function

Predict

one class when *h > 0*



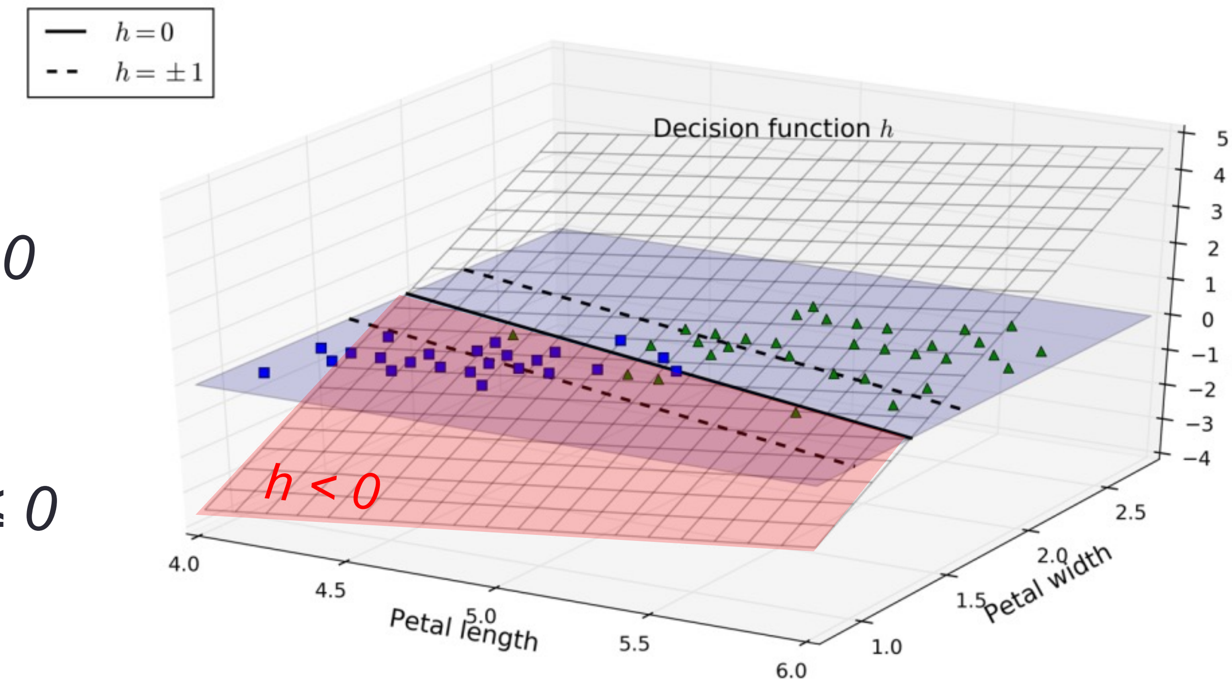
Decision function

Predict

one class when *h > 0*

or

the other class if *h ≤ 0*



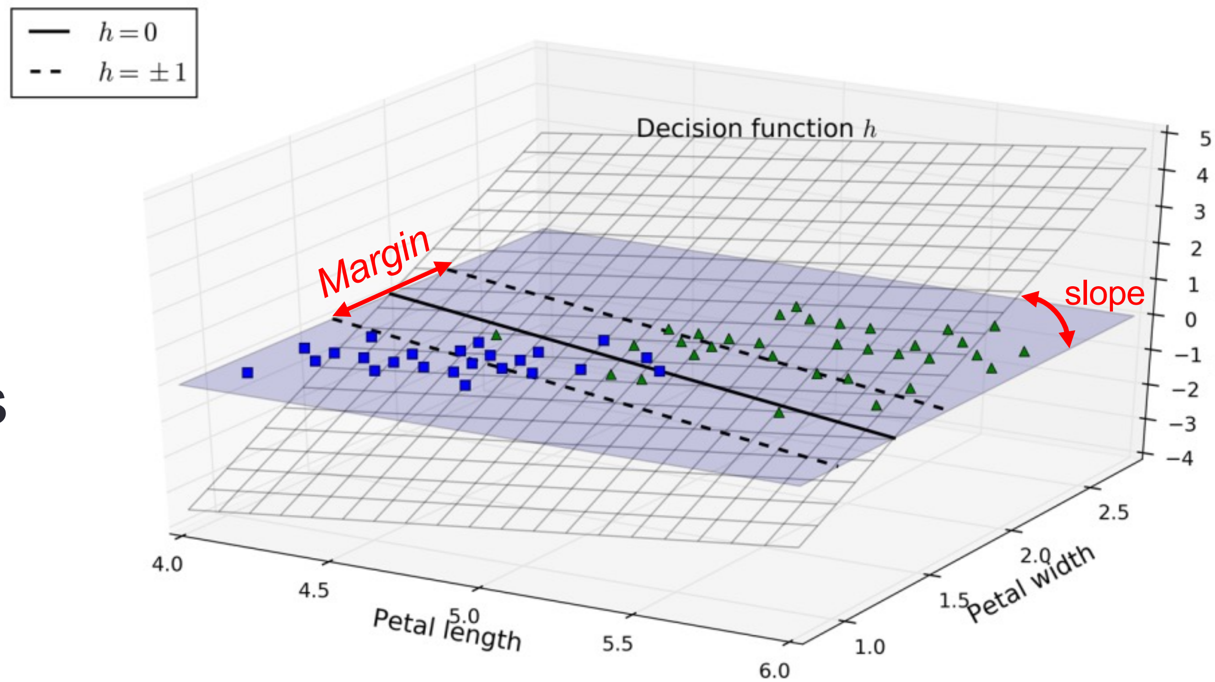
Decision function

Slope of $h(x_1, x_2)$

A small slope gives a wide margin

A large slope gives a narrow margin

The slope is given by w



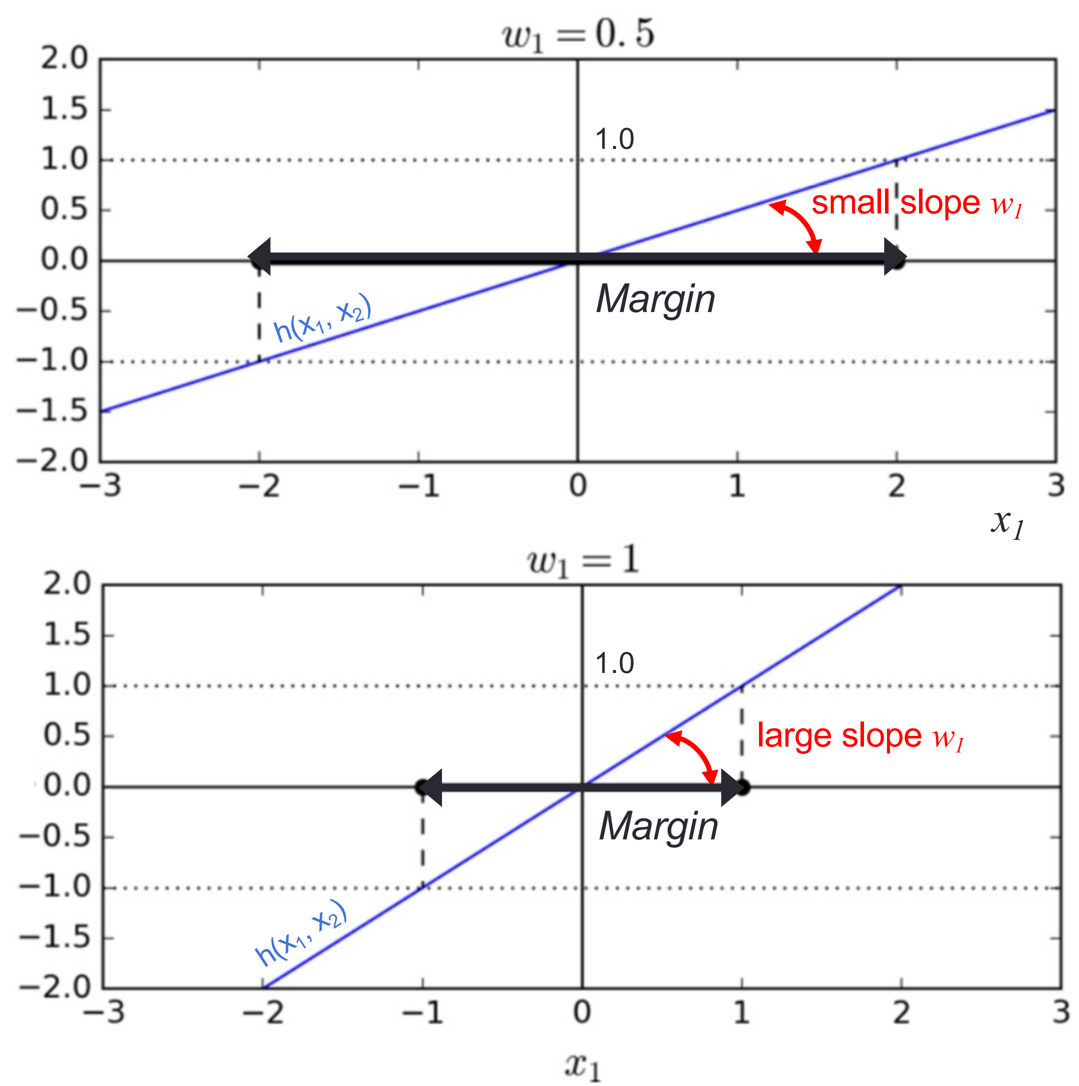
Decision function

Slope of *h(x₁, x₂)*

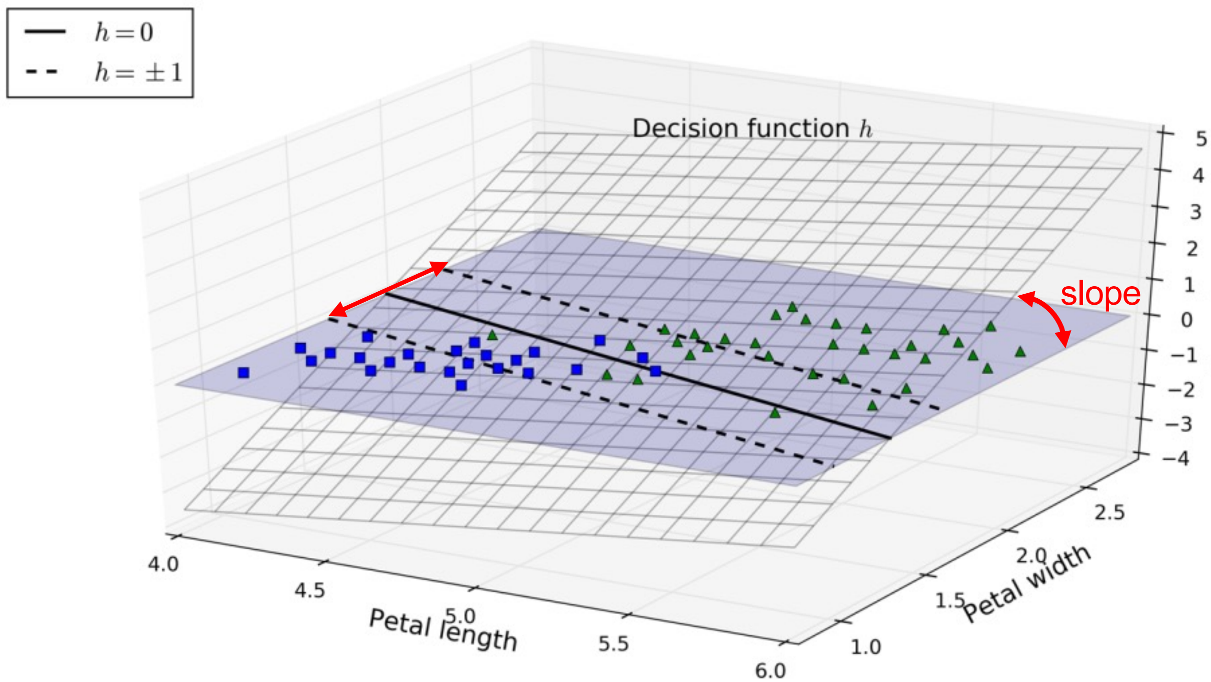
A small slope gives a wide margin

A large slope gives a narrow margin

The slope is given by *w*



Decision function



- We want the widest margin,

  which is found by finding the smallest *w*,
- Or equivalently the smallest norm of *w*

Support Vector Classifier

Decision function

$$h = \mathbf{w}^T \cdot \mathbf{x} + b = w_1 x_1 + \cdots + w_p x_p + b$$

Let define the y-categories as 0 and 1

$$y = \begin{cases} 0 & \text{if} \quad h \leq 0 \\ 1 & \text{if} \quad h > 0 \end{cases}$$

all points category zero on one side

all points category one on the other side

Support Vector Classifier

Decision function

$$h = \mathbf{w}^T \cdot \mathbf{x} + b = w_1\, x_1 + \cdots + w_p\, x_p + b$$

Let define the y-categories as **-1** and +1

$$y = \begin{cases} -1 & \text{if} \quad h \leq -1 \\ 1 & \text{if} \quad h > 1 \end{cases}$$

all points category -1 beyond left margin

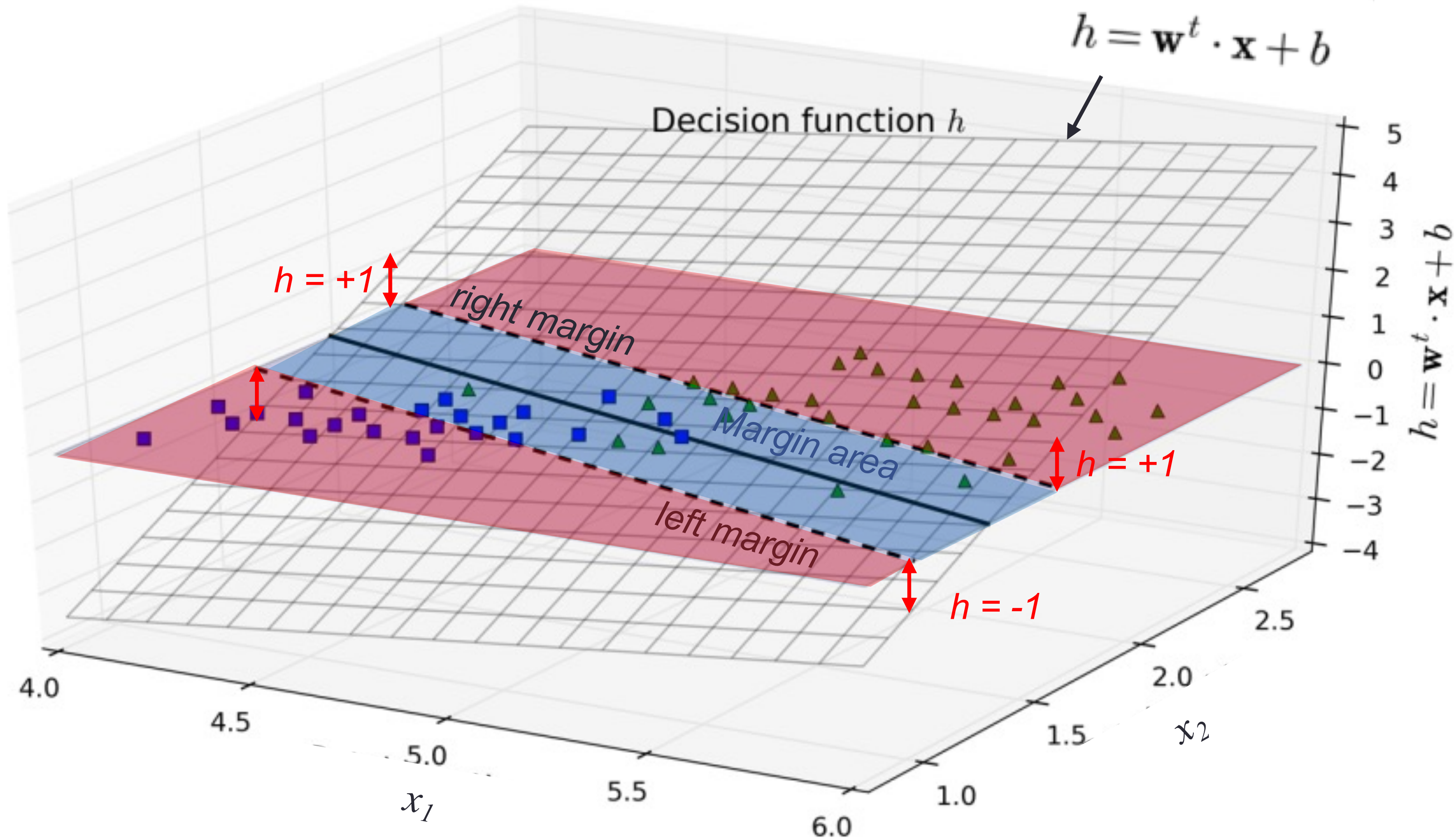
all points category 1 beyond right margin

this definition of y prevents any Margin violation

Decision function

colors for the categories

$$y = \begin{cases} -1 \\ 1 \end{cases}$$



Dashed lines show the points where the Decision function value (height) is equal to +1 or -1

Support Vector Classifier

Decision function

$$h = \mathbf{w}^T \cdot \mathbf{x} + b = w_1 x_1 + \cdots + w_p x_p + b$$

Let define the y-categories as -1 and +1

$$y = \begin{cases} -1 & \text{if} \quad h \leq -1 \\ 1 & \text{if} \quad h > 1 \end{cases}$$

all points category -1 beyond left margin

all points category 1 beyond right margin

note that $y_i h_i \geq 1$ for all data points of both categories

Hard Margin Classifier

Constrained optimization problem

Find $b, w_1, \ldots, w_p$ to

Obj. function $$\text{Min} \quad \frac{1}{2}\,\mathbf{w}$$ (equivalent to Maximize the margin)

Constraints $$\text{subject to} \quad y_i\, h_i \geq 1 \qquad i = 1, \ldots, n$$

but we cannot Minimize a vector

Hard Margin Classifier

Constrained optimization problem

Find $b, w_1, \ldots, w_p$ to

Obj. function $$\text{Min} \quad \frac{1}{2}\,\mathbf{w}^T \cdot \mathbf{w}$$ (Minimize the norm of w)

Constraints $$\text{subject to} \quad y_i\, h_i \geq 1 \qquad i = 1, \ldots, n$$

Hard Margin Classifier

Constrained optimization problem

Find $b,\ w_1, \ldots, w_p$ to

$$\text{Min} \qquad \frac{1}{2}\,[w_1^2 + \cdots + w_p^2]$$

$$\text{subject to} \qquad y_i\,(w_1\,x_{1i} + \cdots + w_p\,x_{pi} + b) \geq 1 \qquad i = 1, \ldots, n$$

a quadratic optimization problem on $b,\ w_1, \ldots, w_p$

(x_i and y_i are known from the data set)

Soft Margin Classifier

Allows for margin violations of size $\zeta_i \geq 0$,

ζ_i is the slack of i^{th} observation

ζ_i measures how much the i^{th} observation is allowed to violate the margin

The soft margin classifier formulation includes ζ_i in the optimization problem as follows

Soft Margin Classifier

Find $b,\ w_1, \ldots, w_p,\ \zeta_1, \ldots, \zeta_n$ to

$$\text{Min} \quad \frac{1}{2}\left[w_1^2 + \cdots + w_p^2\right] + C\sum_{i=1}^{n} \zeta_i$$

$$\text{subject to} \quad y_i\,(w_1\,x_{1i} + \cdots + w_p\,x_{pi} + b) \geq 1 - \zeta_i \qquad i = 1, \ldots, n$$

$$\zeta_1, \ldots, \zeta_n \geq 0$$

reducing $\left[w_1^2 + \cdots + w_p^2\right]$ increases the margin

which also increases $\sum_{i=1}^{n} \zeta_i$

Soft Margin Classifier

Find $b, w_1, \ldots, w_p, \zeta_1, \ldots, \zeta_n$ to

$$\text{Min} \quad \frac{1}{2}\left[w_1^2 + \cdots + w_p^2\right] \downarrow + C \sum_{i=1}^{n} \zeta_i \uparrow$$

$$\text{subject to} \quad y_i\left(w_1 x_{1i} + \cdots + w_p x_{pi} + b\right) \geq 1 - \zeta_i \qquad i = 1, \ldots, n$$

$$\zeta_1, \ldots, \zeta_n \geq 0$$

reducing $\left[w_1^2 + \cdots + w_p^2\right]$ increases the margin
which also increases $\sum_{i=1}^{n} \zeta_i$

} trade-off

Soft Margin Classifier

Find $b, w_1, \ldots, w_p, \zeta_1, \ldots, \zeta_n$ to

$$\text{Min} \quad \frac{1}{2}\left[w_1^2 + \cdots + w_p^2\right] \downarrow + C \sum_{i=1}^{n} \zeta_i \uparrow$$

$$\text{subject to} \quad y_i\left(w_1 x_{1i} + \cdots + w_p x_{pi} + b\right) \geq 1 - \zeta_i \qquad i = 1, \ldots, n$$

$$\zeta_1, \ldots, \zeta_n \geq 0$$

reducing $\left[w_1^2 + \cdots + w_p^2\right]$ increases the margin

which also increases $\sum_{i=1}^{n} \zeta_i$

} trade-off

Find hyperparameter *C* with cross validation

Support Vector Machine

Support Vector Machine *(SVM)*

SVM can predict the class of y

with a nonlinear decision function *h*

Support Vector Machine *(SVM)*

SVM is an extension of the SVC

that results from extending the set of

predictors by means of kernels

Support Vector Machine *(SVM)*

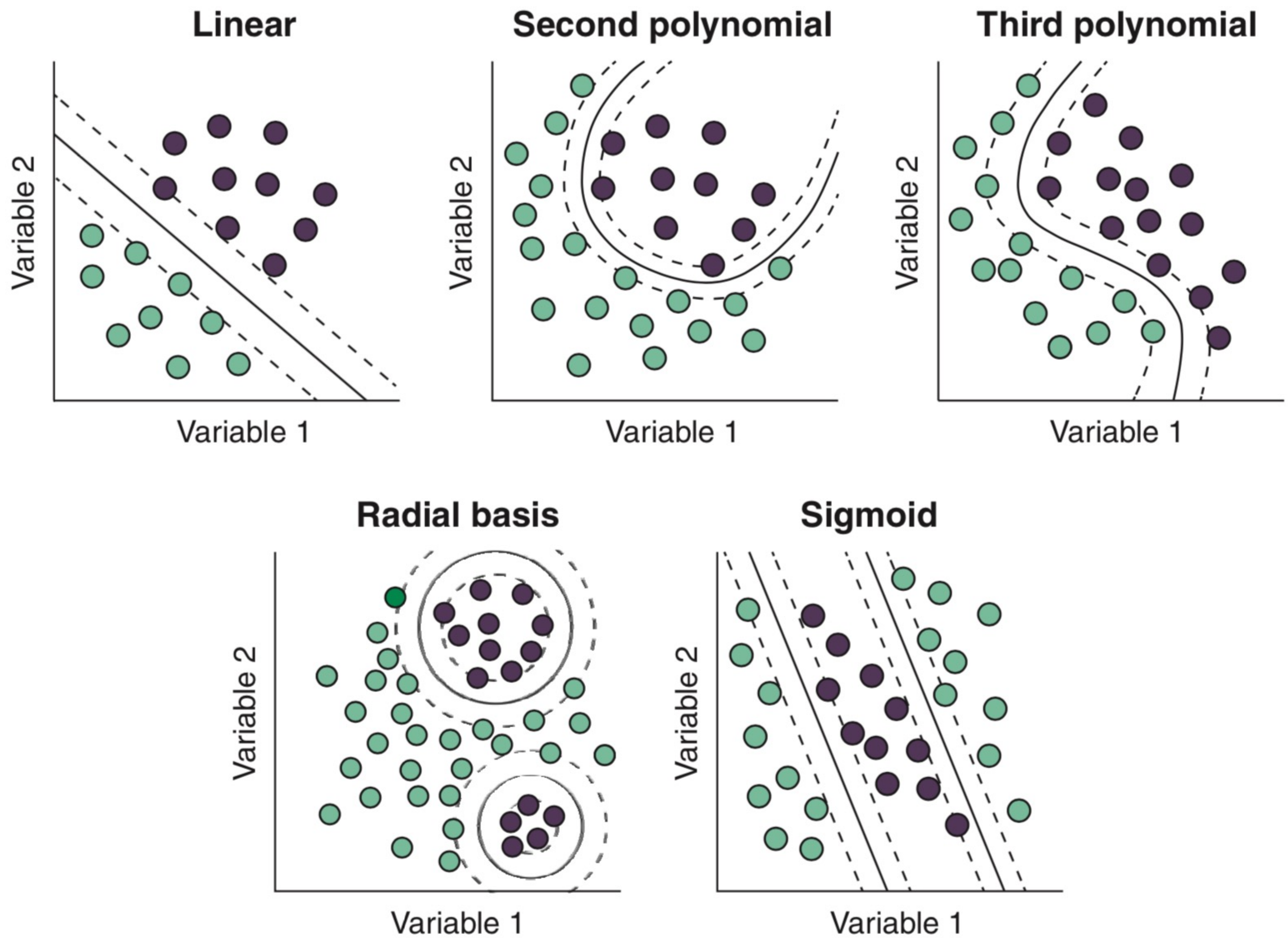
A kernel is a function that transforms a non-linearly separable dataset into a linearly separable dataset

Support Vector Machine *(SVM)*

Kernel types

- linear
- polynomial
- radial basis function (RBF)
- sigmoid

SVM – Boundaries resulting from different kernels



SVM hyperparameters

- Cost *C*
- Kernel type (linear, polynomial, radial, sigmoid)
- Degree (polynomial kernel)
- Gamma (radial kernel)

SVM Extension for K classes

Two approaches

One vs. One

One vs. All

SVM for K classes - **One vs. One**

Fit SVMs (one for each pair of categories)

Classify the observations using each SVM

Assign the observation to the class to which
it was most frequently predicted

SVM for K classes - **One vs. All**

Reclassify observations

+1 if it belongs to category $i = 1$

-1 otherwise

Fit SVM and classify all observations

Repeat for categories $i = 2, \ldots, k$

At the end, each observation has k predictions

Assign each observation to the class to which it was most frequently predicted

EXAMPLE 1

Example 1

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
```

Example 1 – function to display boundary

```
def plot_svc(svc, X, y, h=0.02, pad=0.25):
    x_min, x_max = X[:, 0].min()-pad, X[:, 0].max()+pad
    y_min, y_max = X[:, 1].min()-pad, X[:, 1].max()+pad
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure(figsize=(10,5))
    plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.2)

    my_dict = {-1:'r', 1:'b', 0:'g'}
    colors = np.vectorize(my_dict.get)(y)
    plt.scatter(X[:,0], X[:,1], s=30, c=colors, alpha=0.7)

    sv = svc.support_vectors_
    plt.scatter(sv[:,0], sv[:,1], c='k',marker = 'x', s=30,
                cmap = mpl.cm.Paired, linewidths='1')
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.grid()
    print('Number of support vectors: ', svc.support_.size)
```

```
# X and y must be arrays
```

Example – nonlinearly separable data

```
df = pd.read_csv('data1.csv')
df[:5]
```

	x1	x2	y
0	0.441227	-0.330870	1
1	2.430771	-0.252092	1
2	0.109610	1.582481	1
3	-0.909232	-0.591637	1
4	0.187603	-0.329870	1

```
df.shape
```

```
(40, 3)
```

```
# first 20 rows are train set
df_train = df[:20]
df_test = df[-20:]
```

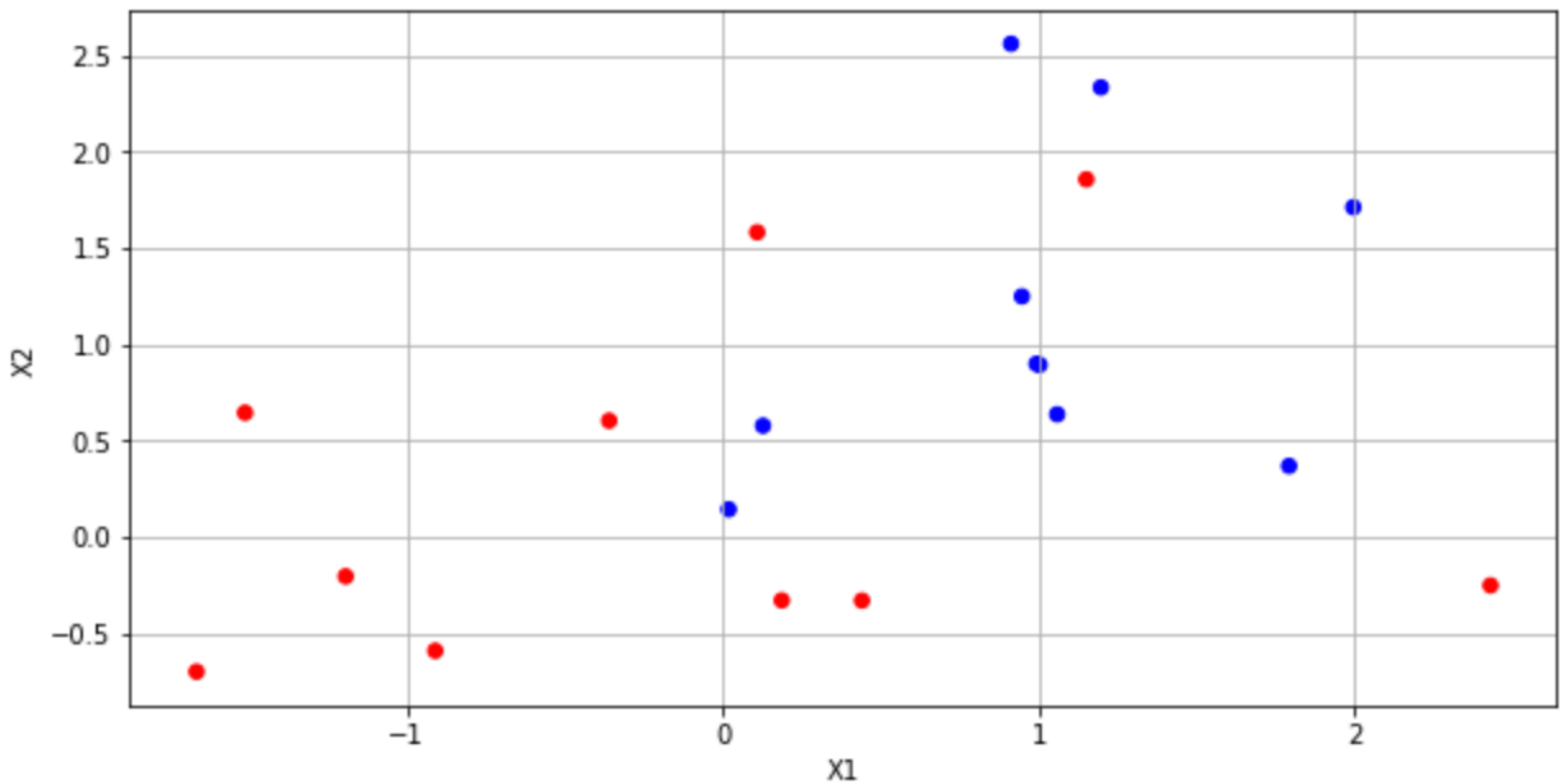
```
y_train = df_train.y
y_test = df_test.y
y_train.value_counts()
```

```
 1    10
-1    10
Name: y, dtype: int64
```

```
X_train = df_train.drop(['y'],axis=1)
X_train = X_train.values
X_test = df_test.drop(['y'],axis=1)
X_test = X_test.values
```

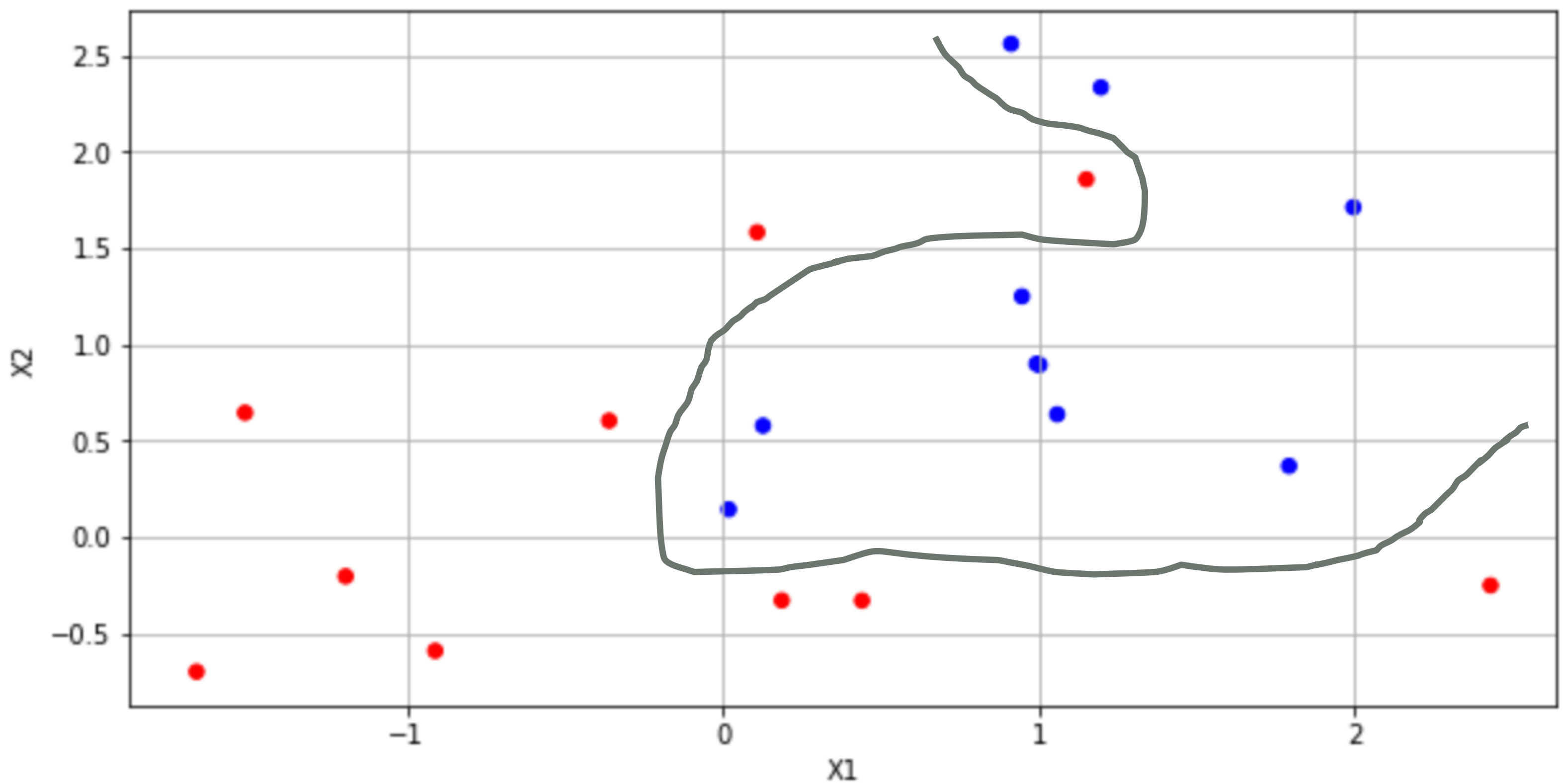
Example – nonlinearly separable data

```
colors = np.where(y_train > 0, 'r', 'b')
plt.figure(figsize=(10,5))
plt.scatter(df_train.x1,df_train.x2,s=30,c=colors)
plt.xlabel('X1')
plt.ylabel('X2')
plt.grid()
```



Example – nonlinearly separable data

```
colors = np.where(y_train > 0, 'r', 'b')
plt.figure(figsize=(10,5))
plt.scatter(df_train.x1,df_train.x2,s=30,c=colors)
plt.xlabel('X1')
plt.ylabel('X2')
plt.grid()
```

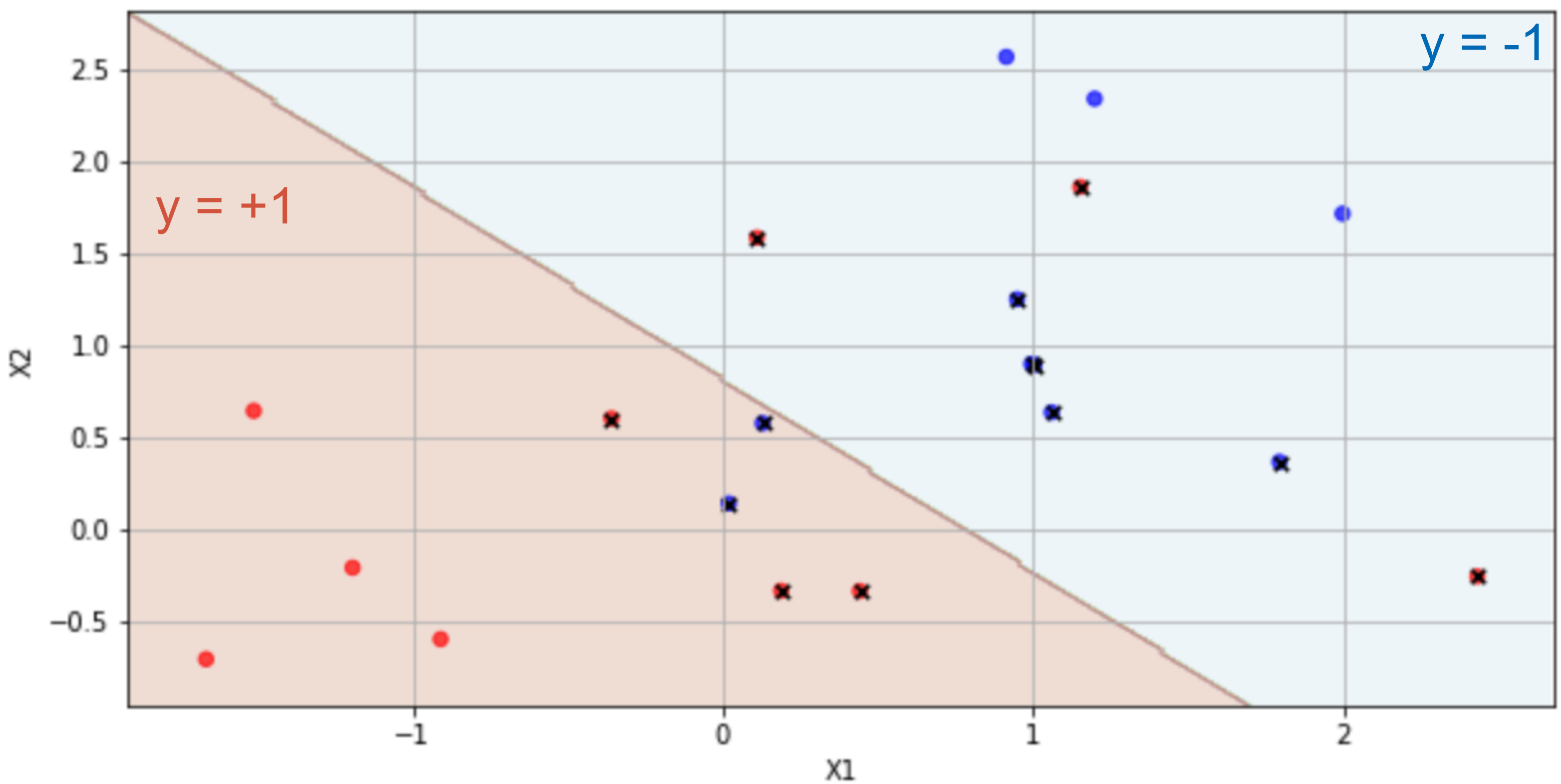


Example - SVC with linear boundary and C=1

```
svc = SVC(C=1,kernel='linear')
svc.fit(X_train,y_train)
plot_svc(svc,X_train,y_train)
```

support vectors in the margin area are shown with x

```
Number of support vectors:  13
```



Example – Identify the 13 support vectors

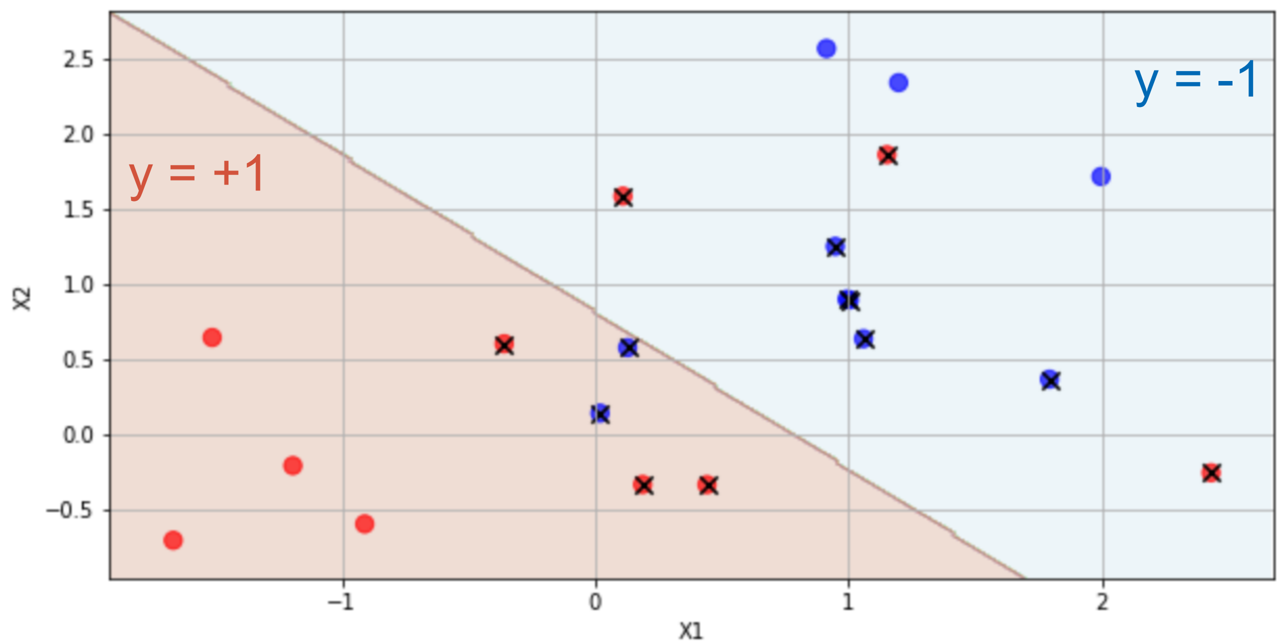
```
svc.support_
```

these are the observations in the margin area

```
array([10, 11, 13, 14, 15, 16, 17,  0,  1,  2,  4,  6,  8], dtype=int32)
```

```
df_train.iloc[list(svc.support_)]
```

	x1	x2	y
10	0.019392	0.143147	-1
11	0.128121	0.577492	-1
13	1.059144	0.636689	-1
14	1.003289	0.894070	-1
15	1.793053	0.368428	-1
16	0.993805	0.898932	-1
17	0.947692	1.249218	-1
0	0.441227	-0.330870	1
1	2.430771	-0.252092	1



Example - Support vectors

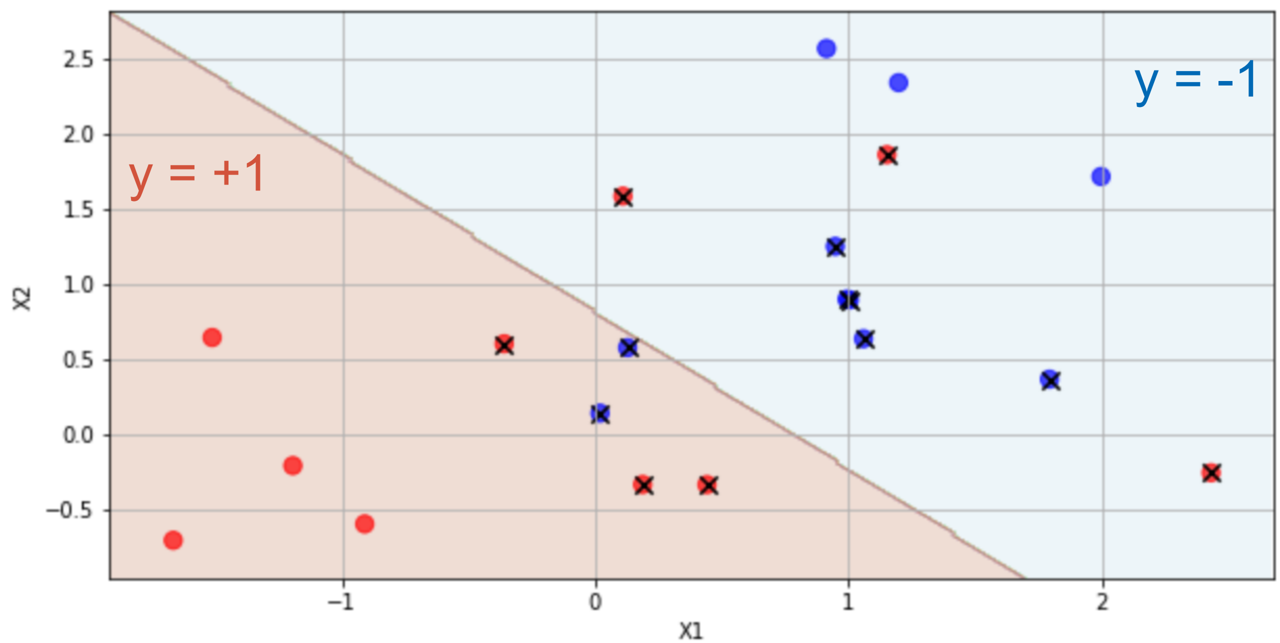
```
svc.support_
```

these are the observations in the margin

```
array([10, 11, 13, 14, 15, 16, 17,  0,  1,  2,  4,  6,  8], dtype=int32)
```

```
df_train.iloc[list(svc.support_)]
```

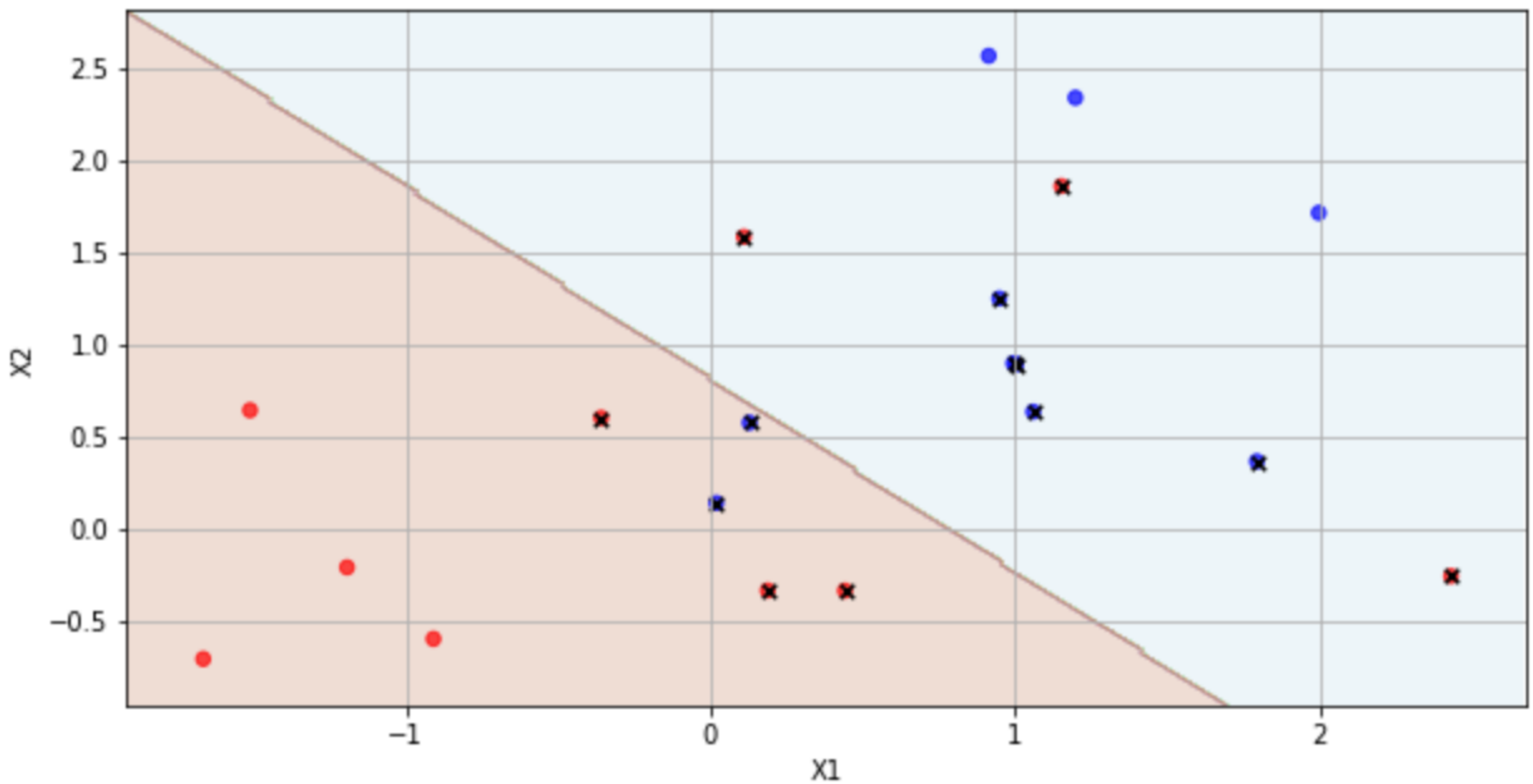
	x1	x2	y
10	0.019392	0.143147	-1
11	0.128121	0.577492	-1
13	1.059144	0.636689	-1
14	1.003289	0.894070	-1
15	1.793053	0.368428	-1
16	0.993805	0.898932	-1
17	0.947692	1.249218	-1
0	0.441227	-0.330870	1
1	2.430771	-0.252092	1



Example - SVC with linear boundary and **C=1**

```
svc = SVC(C=1,kernel='linear')
svc.fit(X_train,y_train)
plot_svc(svc,X_train,y_train)
```

```
Number of support vectors:  13
```

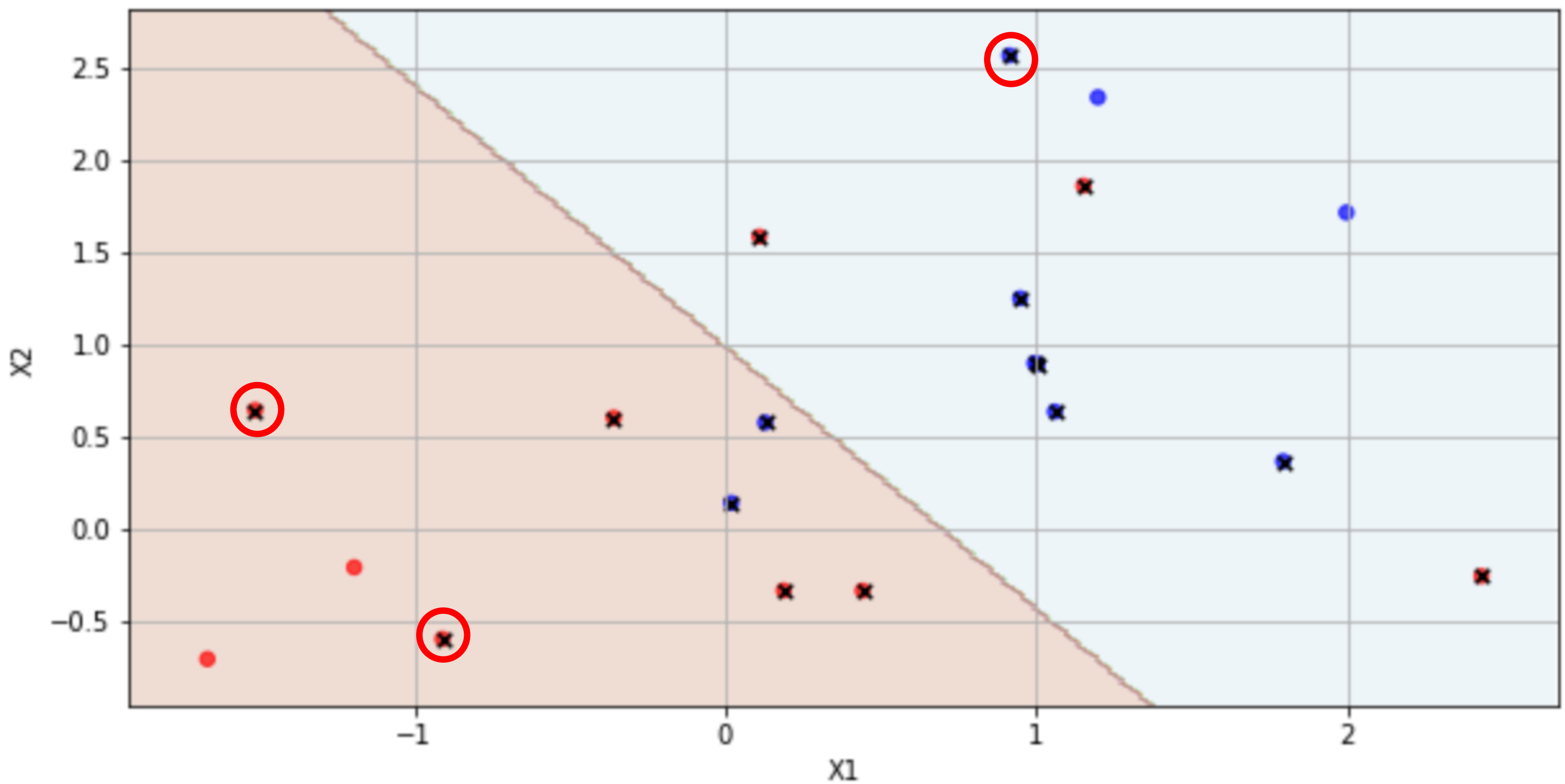


Example - SVC with linear boundary and **C=0.1**

```
svc2 = SVC(C=0.1,kernel='linear')
svc2.fit(X_train,y_train)
plot_svc(svc2,X_train,y_train)
```

```
Number of support vectors:  16
```

smaller C → wider margin



Example – GridSearchCV to find best C

```
param_grid={'C':[0.001,0.01,0.1,5,10,100]}
```

```
kfold = StratifiedKFold(n_splits=5,shuffle = True, random_state = 1)
```

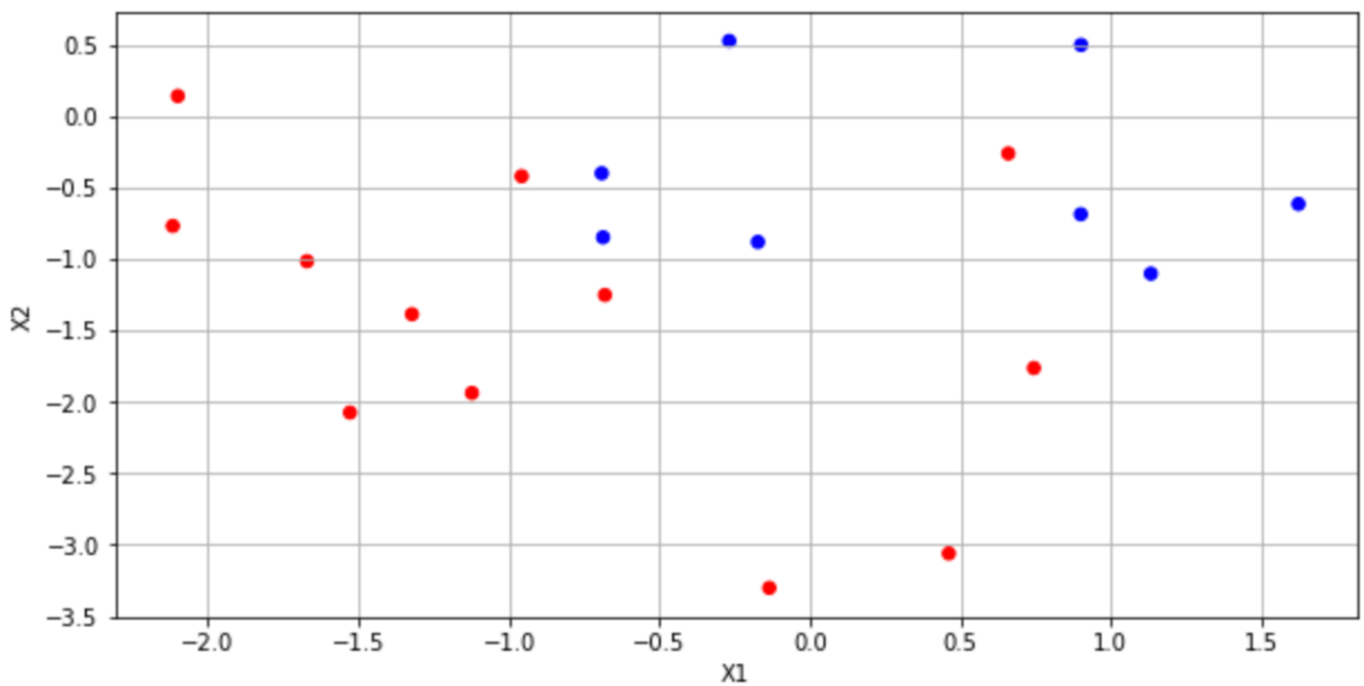
```
clf = GridSearchCV(SVC(kernel='linear'),param_grid,
                   cv=kfold, scoring = 'accuracy')
clf.fit(X_train,y_train);
```

```
# best C
clf.best_params_
```

```
{'C': 0.001}
```

Example – Plotting the Test data

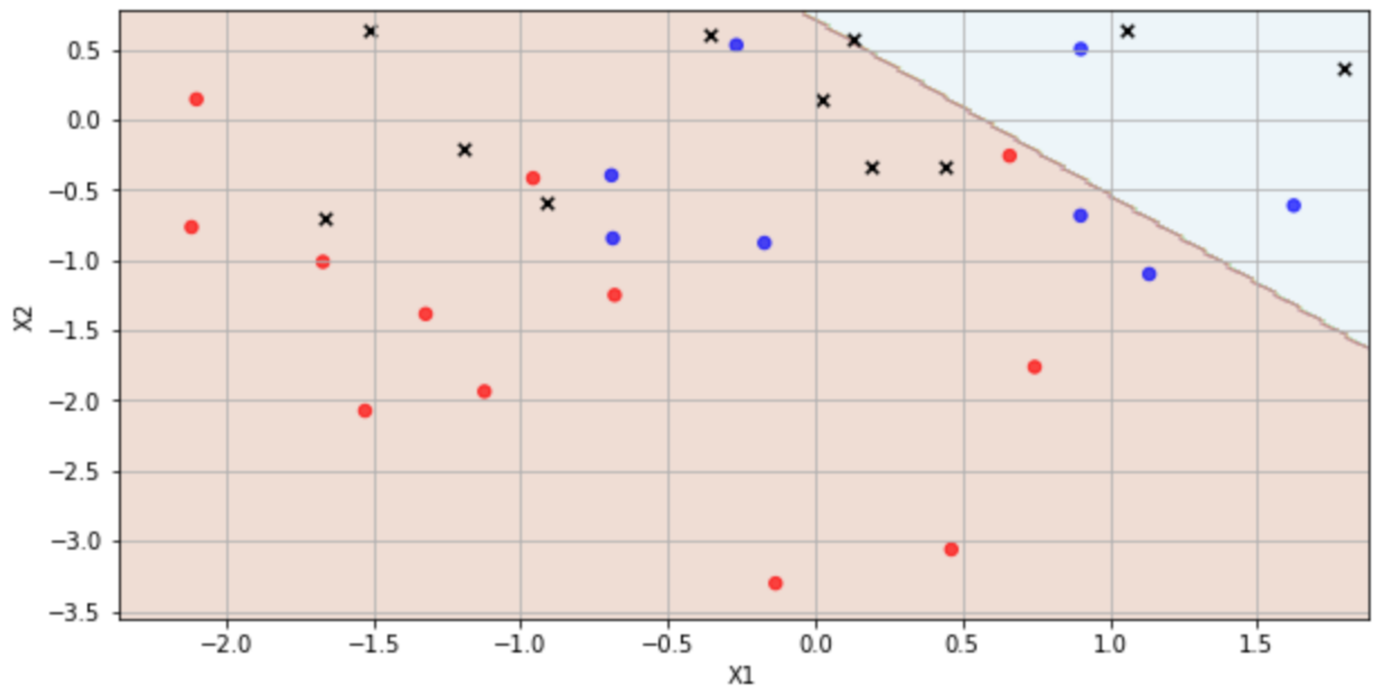
```
# Scatterplot of test data
colors = np.where(y_test > 0, 'r', 'b')
plt.figure(figsize=(10,5))
plt.scatter(df_test.x1,df_test.x2,s=30,c=colors)
plt.xlabel('X1')
plt.ylabel('X2')
plt.grid()
```



Example - SVC with best C = 0.001

```
svc2 = SVC(C=0.001,kernel='linear')
svc2.fit(X_train,y_train);
plot_svc(svc2,X_test,y_test)
```

```
Number of support vectors:  20
```



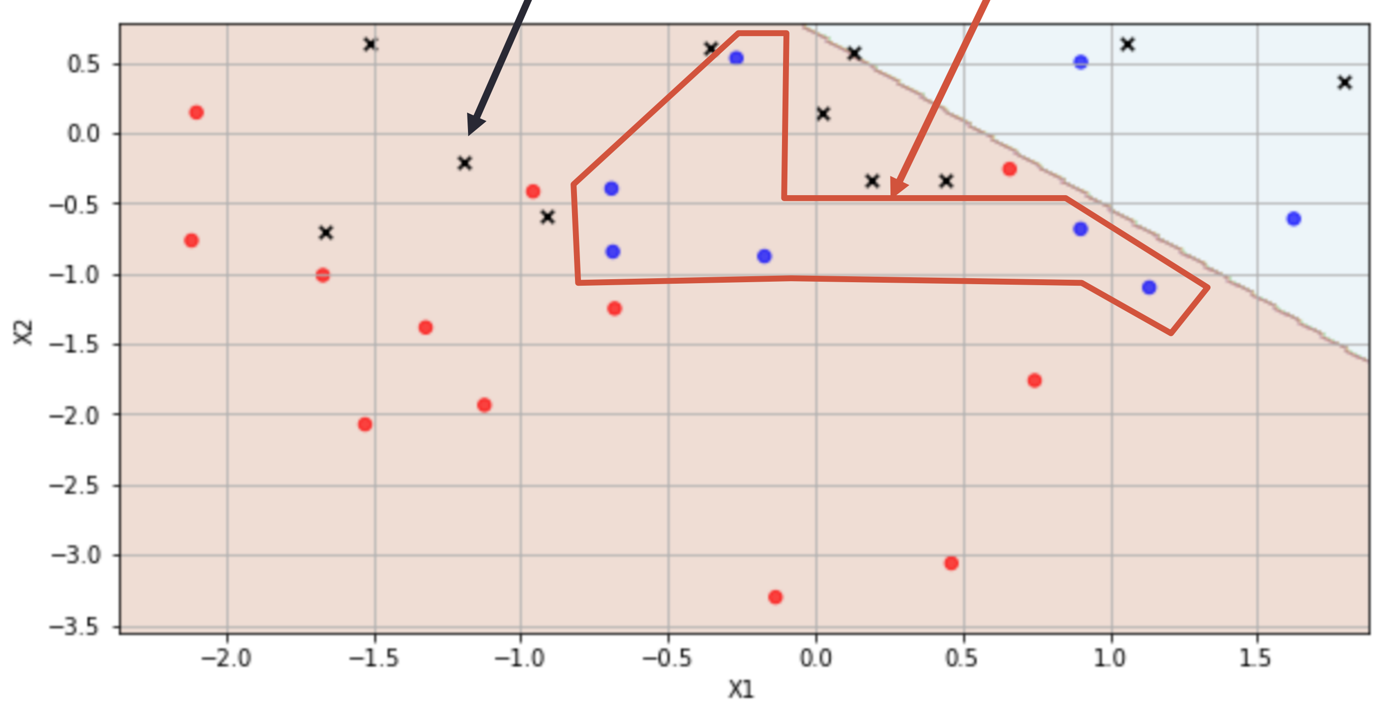
Example - SVC with best C = 0.001

```
svc2 = SVC(C=0.001,kernel='linear')
svc2.fit(X_train,y_train);
plot_svc(svc2,X_test,y_test)
```

```
Number of support vectors:  20
```

support vectors from the train set with C = 0.001

6 misclassifications



Example - SVC with best C = 0.001

```
y_pred = svc2.predict(X_test)
pd.crosstab(y_test,y_pred,
            rownames = ['y'],
            colnames = ['y_pred'])
```

y_pred	-1	1
y		
-1	2	6
1	0	12

```
# misclassified rows
```

```
df1[y_test != y_pred]
```

	y_test	y_pred
27	-1	1
28	-1	1
32	-1	1
34	-1	1
35	-1	1
36	-1	1

```
df1 = pd.DataFrame()
df1['y_test']=y_test
df1['y_pred']=y_pred
df1
```

	y_test	y_pred
20	-1	-1
21	1	1
22	1	1
23	1	1
24	1	1
25	1	1
26	1	1
27	-1	1
28	-1	1
29	1	1
30	1	1

Example – SVM nonlinear boundary

```
# data with a nonlinear boundary
```

```
df3 = pd.read_csv('data3.csv')
df3.shape
```

```
(200, 3)
```

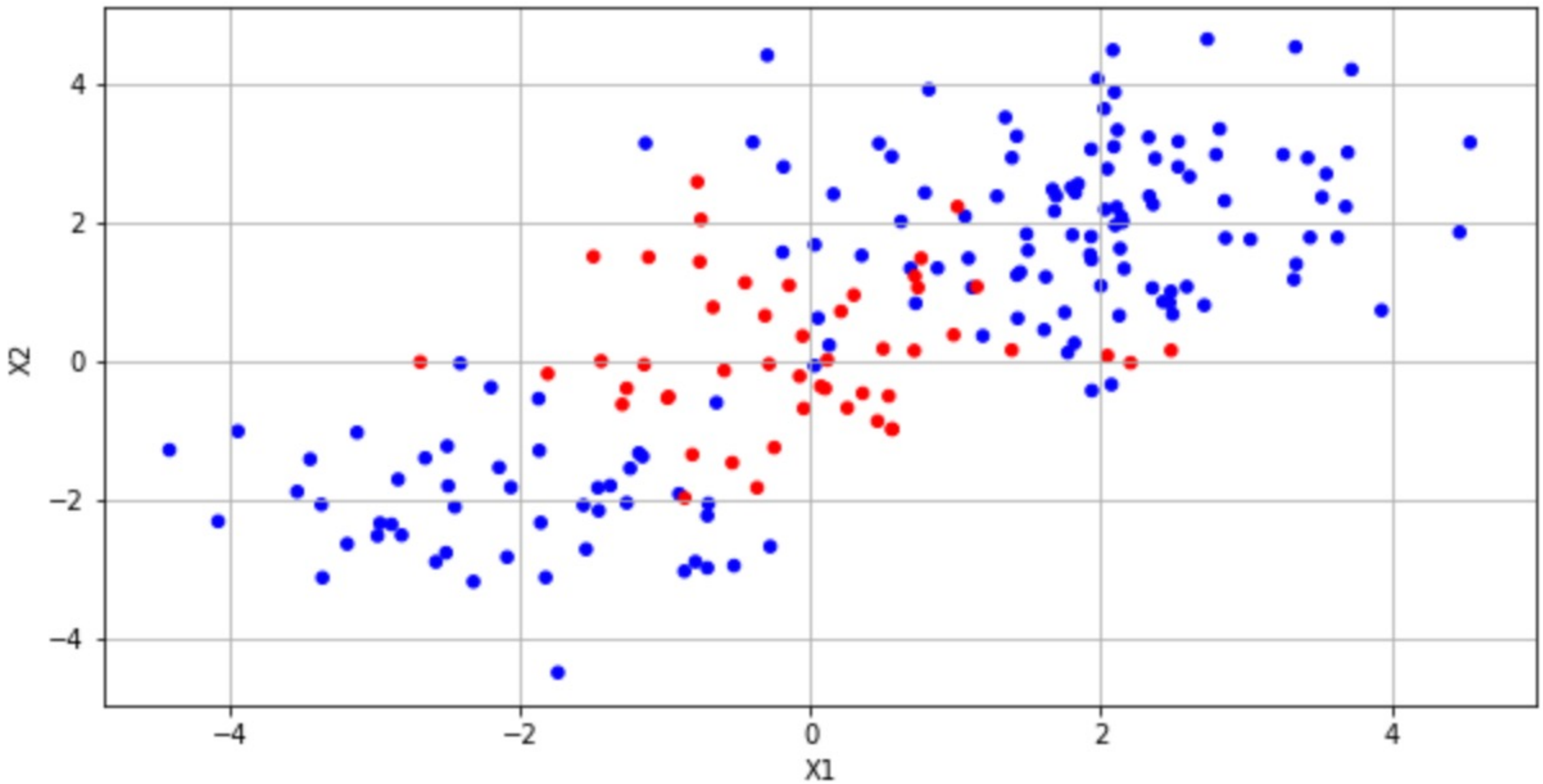
```
y = df3.y
X = df3.drop(['y'],axis=1)
```

```
y.value_counts()
```

```
-1    150
 1     50
Name: y, dtype: int64
```

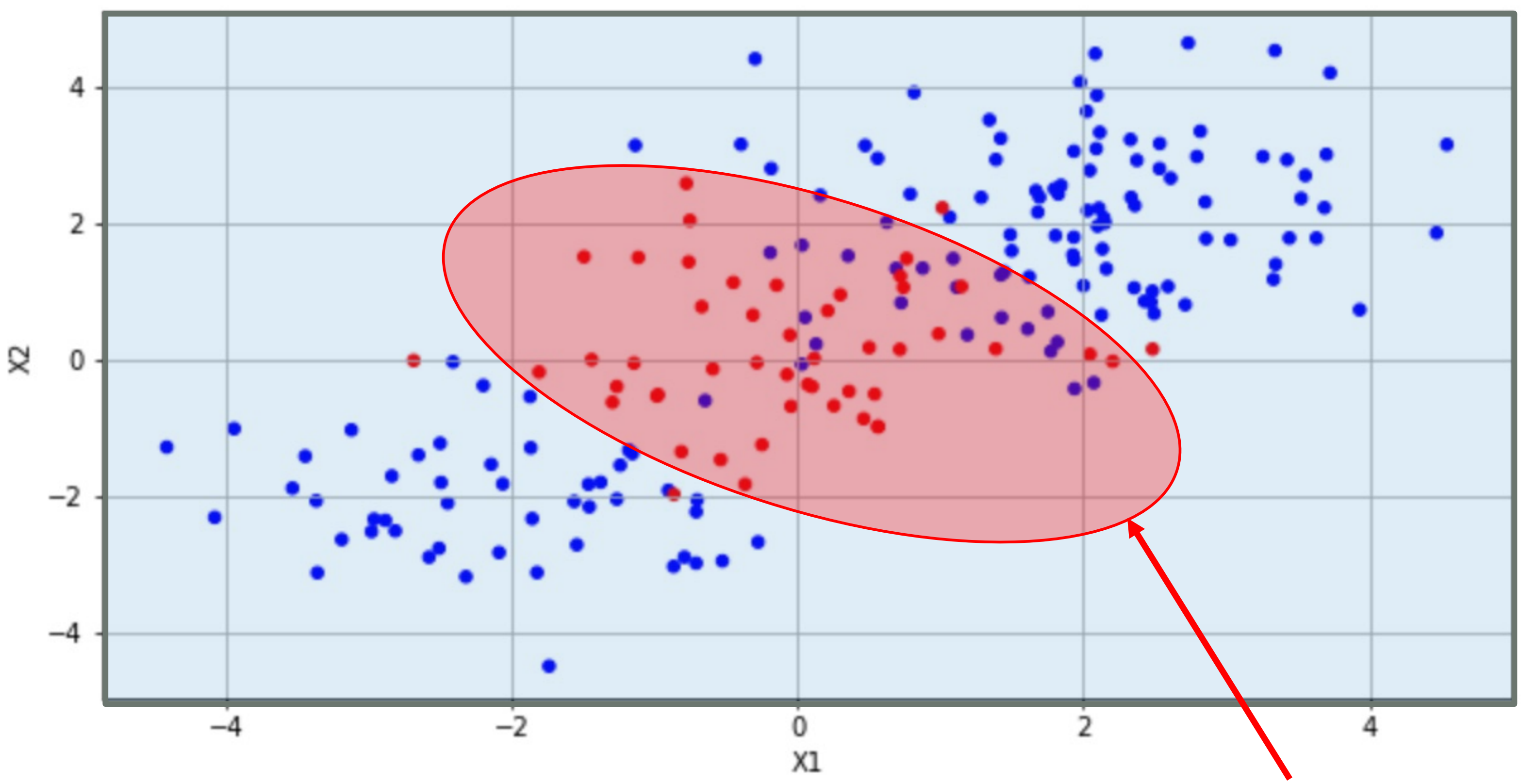
Example – SVM nonlinear boundary

```
colors = np.where(y_test > 0, 'r', 'b')
plt.figure(figsize=(10,5))
plt.scatter(df3.x1,df3.x2,s=20,c=colors)
plt.xlabel('X1')
plt.ylabel('X2')
plt.grid()
```



Example – SVM nonlinear boundary

```
plt.figure(figsize=(10,5))
plt.scatter(df3.x1,df3.x2,s=20,c=colors)
plt.xlabel('X1')
plt.ylabel('X2')
plt.grid()
```



Try nonlinear kernels to find nonlinear boundaries

Example – Nonlinear kernels

- To fit a polynomial kernel use *kernel = 'poly'* selecting appropriate degree
- To fit a radial kernel use *kernel = 'rbf'* selecting appropriate gamma

Example – SVM nonlinear kernel

.

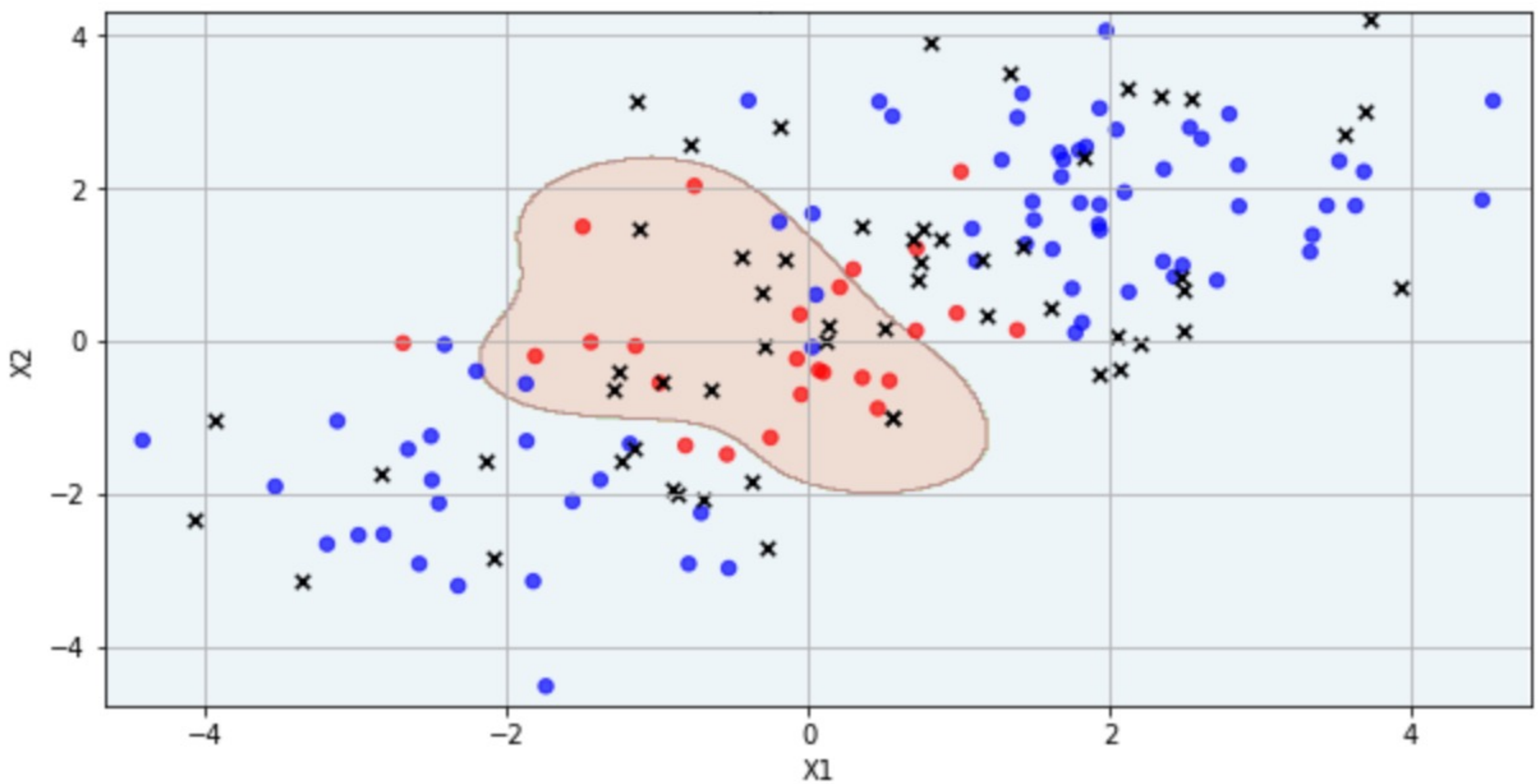
```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,
                                                 test_size=0.50,
                                                 random_state=2)
```

```
# transform df to arrays
X_train = X_train.values
X_test = X_test.values
```

Example – SVM nonlinear kernel

```
svm = SVC(C=1,kernel='rbf',gamma=1.0)
svm.fit(X_train,y_train);
plot_svc(svm,X_test,y_test)
```

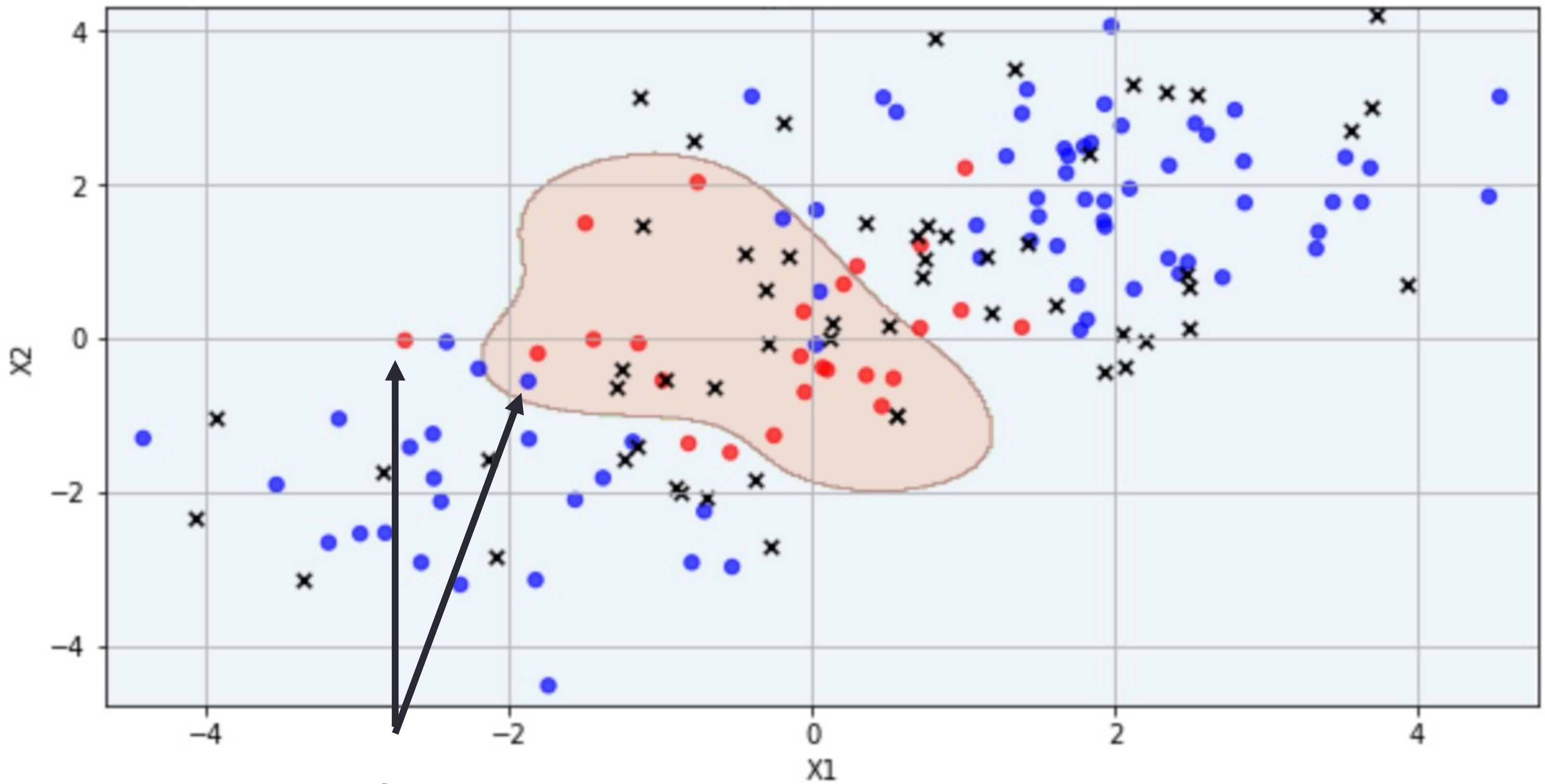
```
Number of support vectors:  61
```



Example – SVM nonlinear kernel

```
svm = SVC(C=1,kernel='rbf',gamma=1.0)
svm.fit(X_train,y_train);
plot_svc(svm,X_test,y_test)
```

```
Number of support vectors:  61
```



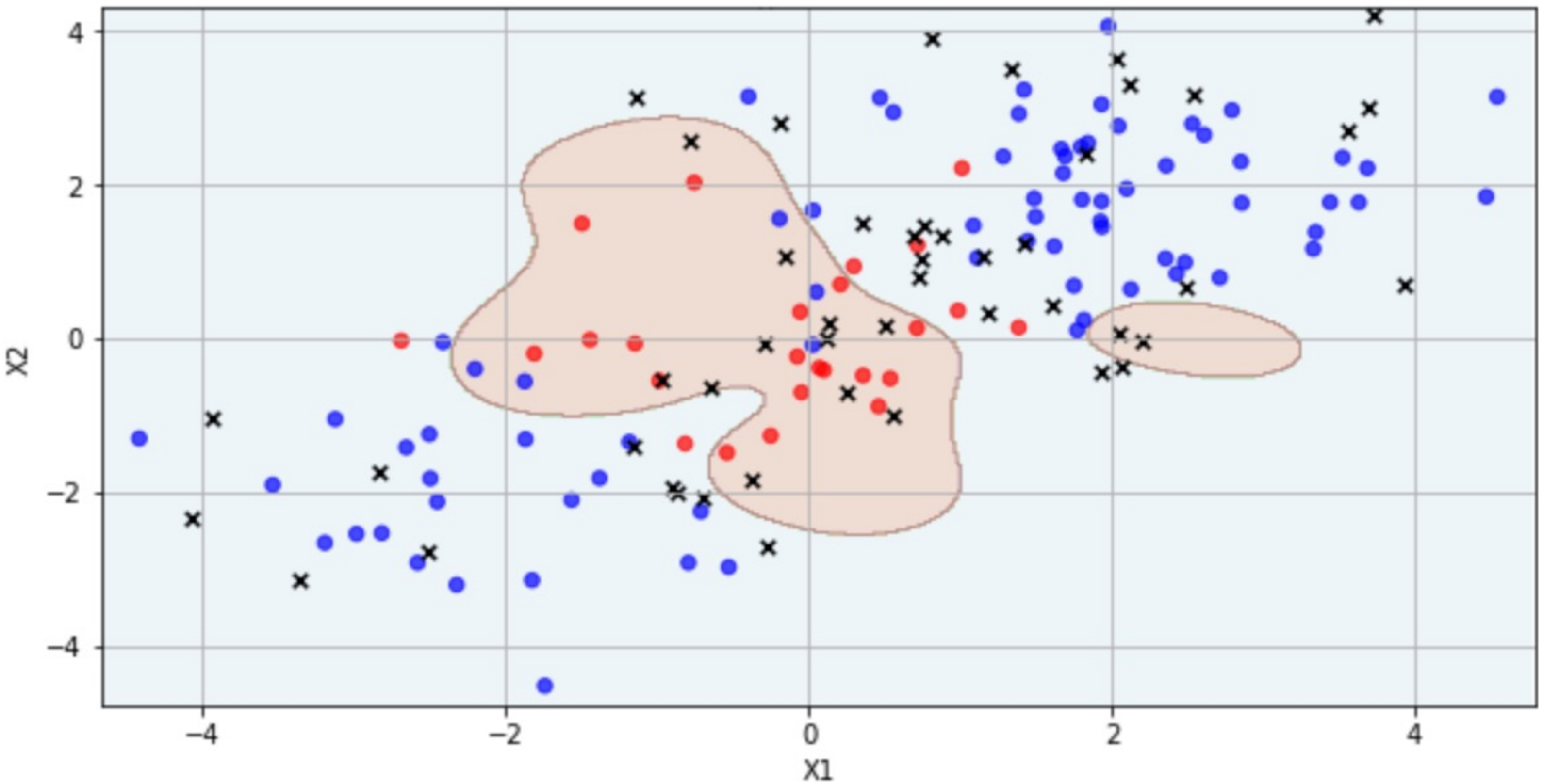
misclassifications

Example – Increase cost to decrease margin

```
svm2 = SVC(C=10,kernel='rbf',gamma=1)
svm2.fit(X_train,y_train)
plot_svc(svm2,X_test,y_test)
```

```
Number of support vectors:  52
```

number of support vectors decreases



Increasing C gives a more irregular decision boundary

Example – Search for the best cost and gamma

```
param_grid= {'C':[0.01,0.1,1,10],
             'gamma':[0.5,1,2,3]}
```

```
kfold = StratifiedKFold(n_splits=5,shuffle = True, random_state = 1)
```

```
clf = GridSearchCV(SVC(kernel='rbf'),param_grid,
                   cv=kfold, scoring='accuracy')
clf.fit(X_train,y_train);
```

```
clf.best_params_
```

```
{'C': 1, 'gamma': 0.5}
```

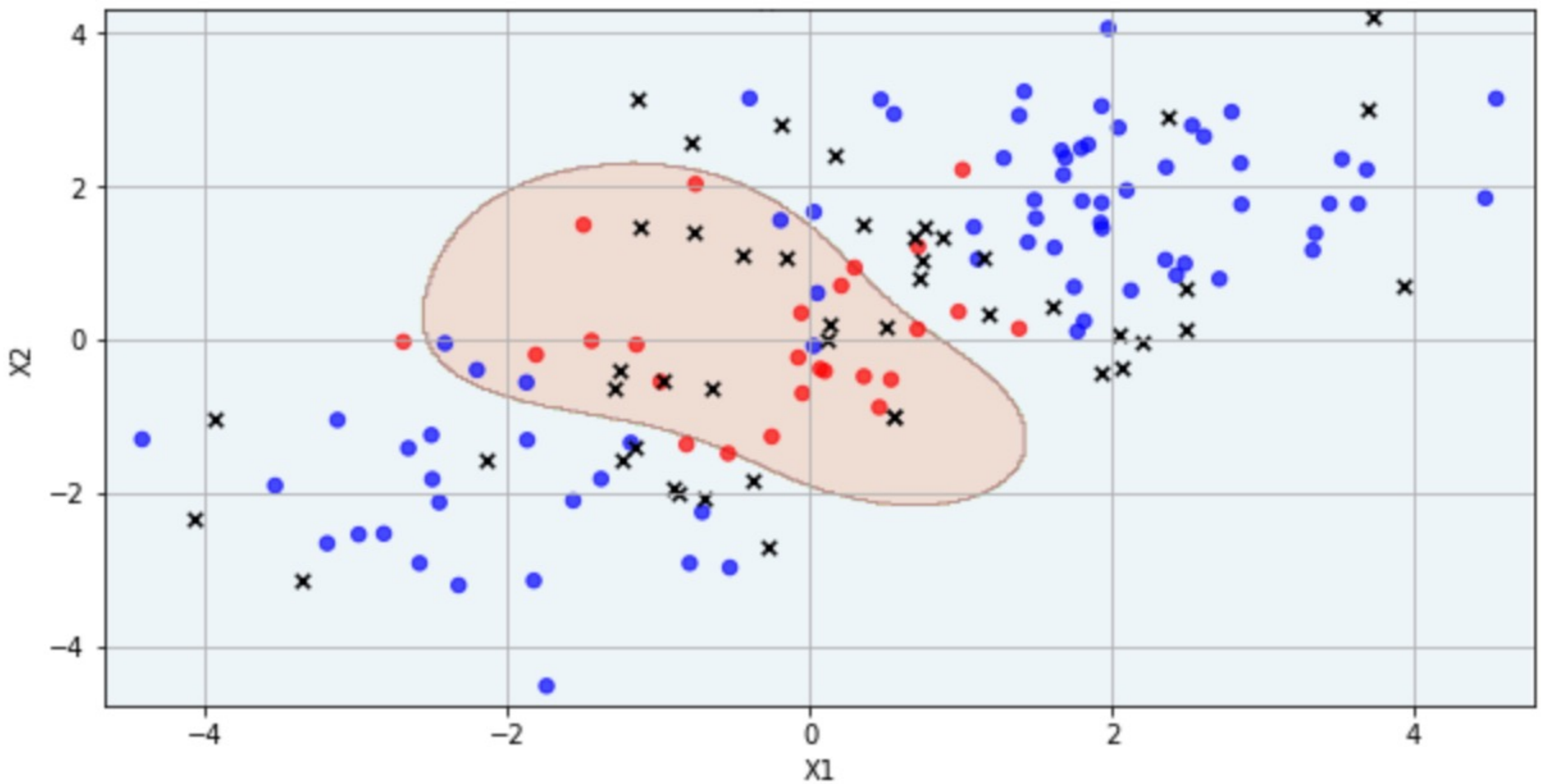
```
plot_svc(clf.best_estimator_,X_test,y_test)
```

```
Number of support vectors:  50
```

Example – Search for the best cost and gamma

```
plot_svc(clf.best_estimator_,X_test,y_test)
```

```
Number of support vectors:  50
```



Example – test accuracy rate with best C, gamma

```
y_pred = clf.best_estimator_.predict(X_test)
pd.crosstab(y_test,y_pred,
                rownames = ['y'],
                colnames = ['y_pred'])
```

y_pred	-1	1
y		
-1	69	6
1	7	18

```
clf.best_estimator_.score(X_test,y_test)
```

0.87

Example – More than 2 categories

```
df4 = pd.read_csv('data4.csv')
df4[:5]
```

	x1	x2	y	color
0	2.091205	3.091283	-1	r
1	0.053030	0.613650	-1	r
2	-0.296492	4.409834	-1	r
3	3.727836	4.204556	-1	r
4	2.794828	2.976421	-1	r

```
df4.shape
```

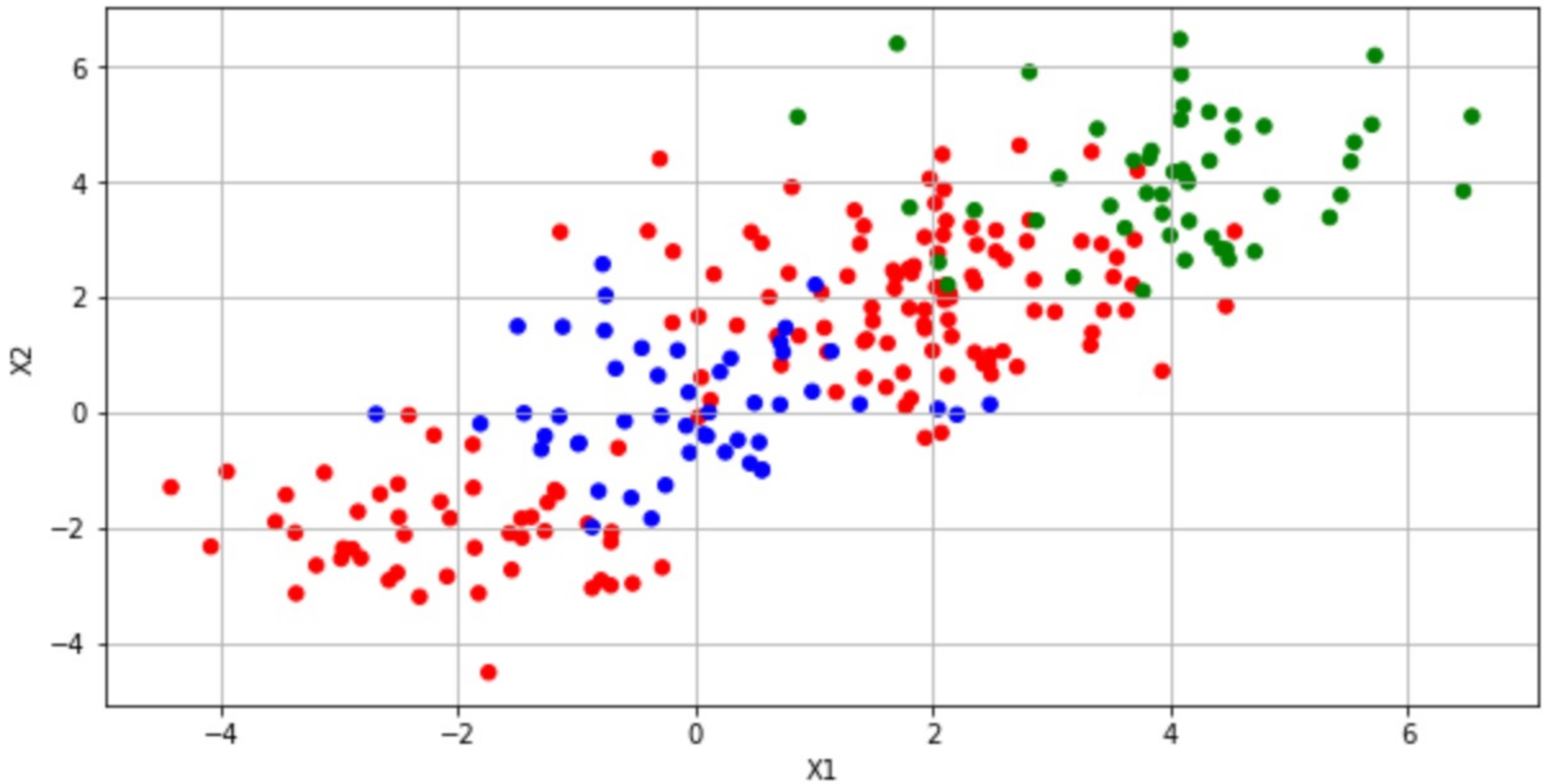
```
(250, 4)
```

```
y = df4.y
y.value_counts()
```

```
-1    150
 0     50
 1     50
```

Example – More than 2 categories

```
colors = np.where(y_test > 0, 'r', 'b')
plt.figure(figsize=(10,5))
plt.scatter(df4.x1,df4.x2,s=30,c=colors)
plt.xlabel('X1')
plt.ylabel('X2')
plt.grid()
```



Example – More than 2 categories

.

```
list1 = ['y','color']
X = df4.drop(list1,axis=1)
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,stratify = y,
                                                 test_size=0.50,
                                                 random_state=2)
```

```
type(X_train)
```

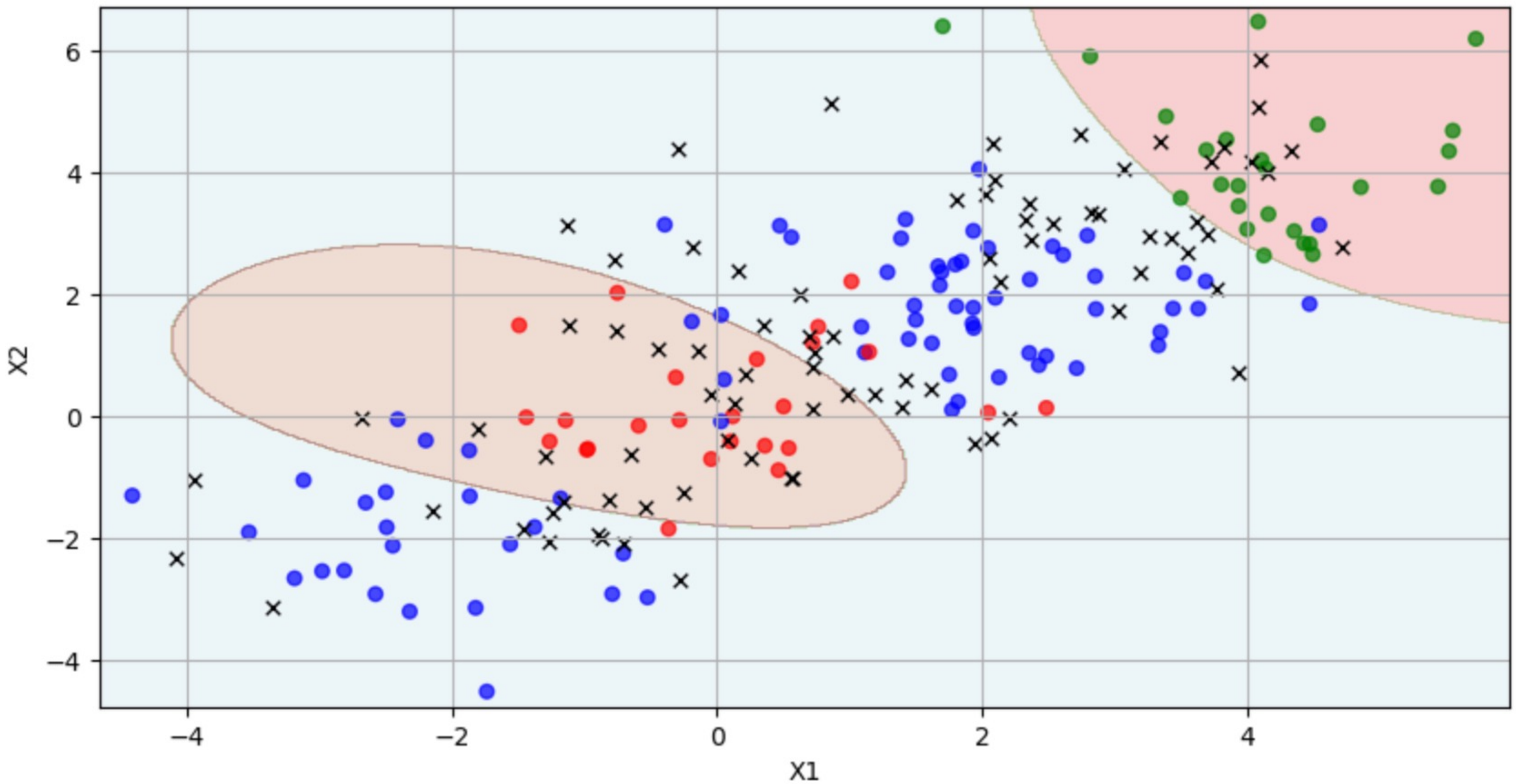
```
pandas.core.frame.DataFrame
```

```
# transform to arrays
X_train = X_train.values
X_test = X_test.values
```

Example – Plot test set on SVC regions

```
svm5 = SVC(C=1, kernel='rbf',gamma = 'scale')
svm5.fit(X_train,y_train)
plot_svc(svm5,X_test,y_test)
```

```
Number of support vectors:  84
```




```
svm5._gamma
```

```
0.0902933060590302
```

Parameters:

C : *float, default=1.0*

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'*

Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`. For an intuitive visualization of different kernel types see Plot classification boundaries with different SVM Kernels.

degree : *int, default=3*

Degree of the polynomial kernel function ('poly'). Must be non-negative. Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

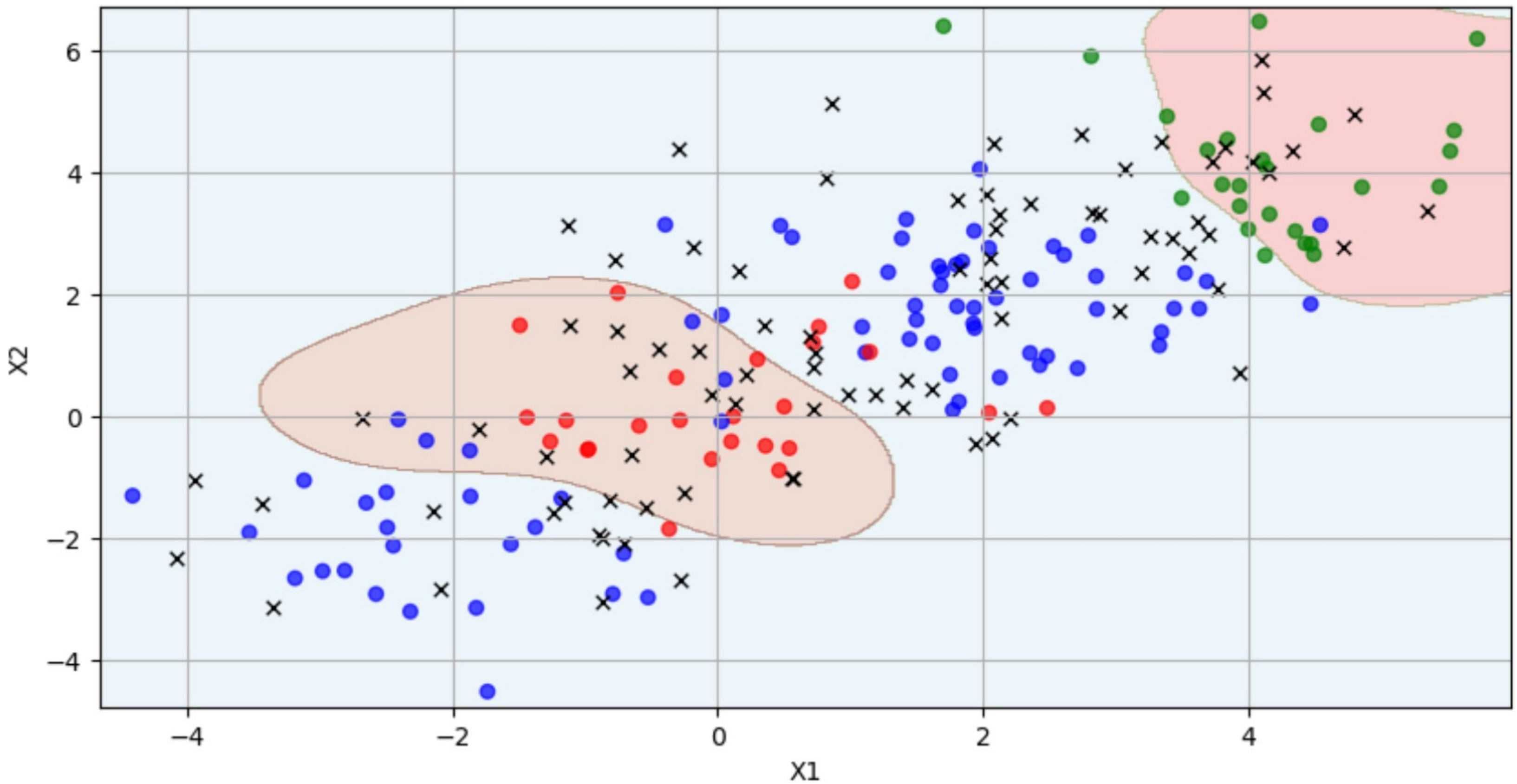
- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features
- if float, must be non-negative.

Changed in version 0.22: The default value of `gamma` changed from 'auto' to 'scale'.

Example – Plot test set on SVC regions

```
svm5 = SVC(C=1, kernel='rbf',gamma = 'auto')
svm5.fit(X_train,y_train)
plot_svc(svm5,X_test,y_test)
```

```
Number of support vectors:  87
```




```
svm5._gamma
```

```
0.5
```

Example – Test accuracy rate

```
y_pred = svm5.predict(X_test)
pd.crosstab(y_test,y_pred,
            rownames = ['y'],
            colnames = ['y_pred'])
```

y_pred	-1	0	1
y			
-1	68	1	6
0	5	20	0
1	7	0	18

```
# number of accurate predictions (main diagonal sum)
accuracy_score(y_test, y_pred,normalize=False)
```

```
106
```

```
# test accuracy rate (out of 125 test observations)
accuracy_score(y_test,y_pred)
```

```
0.848
```

Example – GridSearchCV

```
params = {'C': [0.01, 0.1,0.5,1,2,3,5],
          'gamma': [0.01,0.05,0.1,0.2,0.3]}
```

```
grid = GridSearchCV(SVC(kernel = 'rbf'),
                    params, cv=kfold)
grid.fit(X_train, y_train)
grid.best_params_
```

```
{'C': 2, 'gamma': 0.05}
```

```
y_pred = grid.best_estimator_.predict(X_test)
pd.crosstab(y_test,y_pred,
             rownames = ['y'],
             colnames = ['y_pred'])
```

y_pred	-1	0	1
y			
-1	66	1	8
0	1	24	0
1	7	0	18

```
# number of accurate predictions (main diagonal sum)
accuracy_score(y_test, y_pred,normalize=False)
```

```
108
```

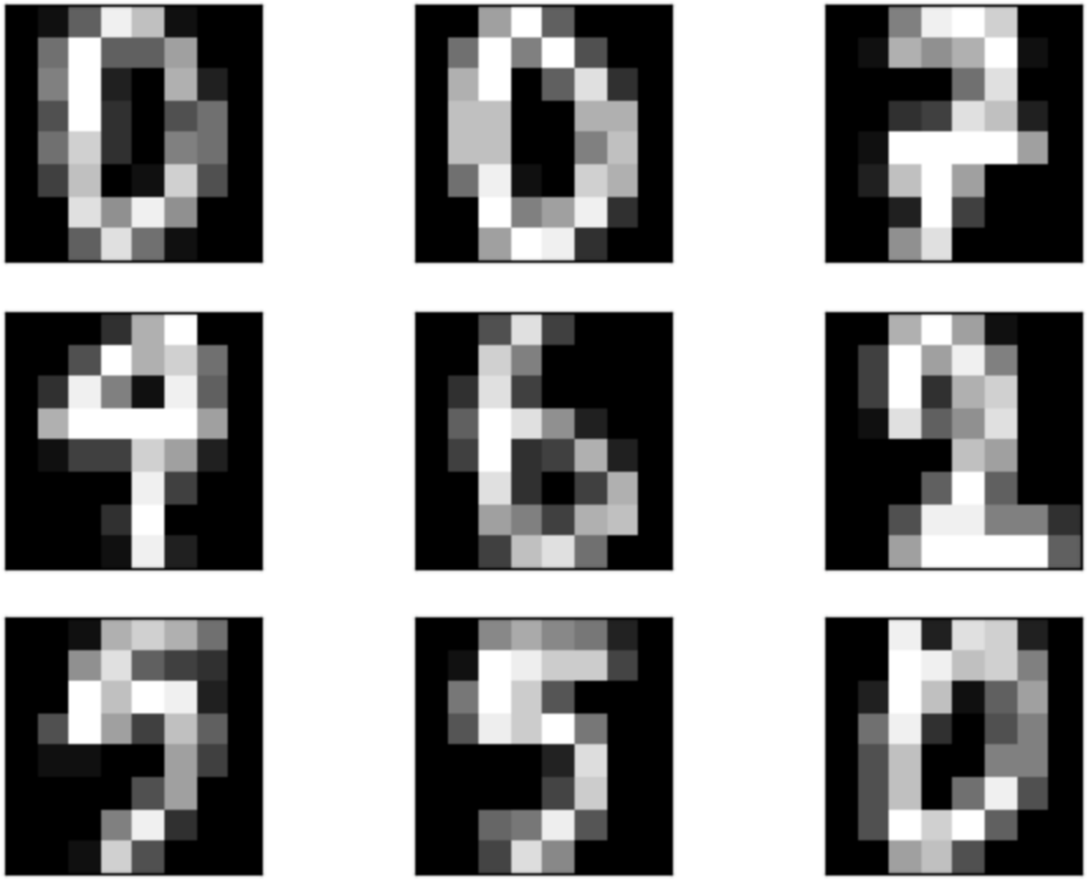
```
# test accuracy rate (out of 125 test observations)
accuracy_score(y_test,y_pred)
```

```
0.864
```

EXAMPLE 2 – Digits dataset

Example 2 – Digits recognition

Identify handwritten digits (0,…,9)



Example 2 – Digits recognition

Datasets available at

archive.ics.uci.edu/ml/machine-learning-databases/optdigits

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
```

Example 2 – Digits recognition

```
X_train = pd.read_csv('optdigits.tra',header=None)
X_test  = pd.read_csv('optdigits.tes',header=None)
```

```
# target variable in column 64
```

```
y_train = X_train[64]
y_test  = X_test[64]
```

```
X_train = X_train.drop(X_train.columns[64],axis=1)
X_test  = X_test.drop(X_test.columns[64],axis=1)
```

```
X_train.shape
```

```
(3823, 64)
```

3823 images in the train set

```
X_test.shape
```

```
(1797, 64)
```

1797 images in the test set

Example 2 – Digits recognition

```
# see digit in row 4
```

```
y_train[4]
```

```
6
```

```
X_train.values[4]
```

each digit is stored in a single row with 64 columns

```
array([ 0,  0,  5, 14,  4,  0,  0,  0,  0,  0, 13,  8,  0,  0,  0,  0,  0,
        3, 14,  4,  0,  0,  0,  0,  0,  6, 16, 14,  9,  2,  0,  0,  0,  4,
       16,  3,  4, 11,  2,  0,  0,  0, 14,  3,  0,  4, 11,  0,  0,  0, 10,
        8,  4, 11, 12,  0,  0,  0,  4, 12, 14,  7,  0,  0])
```

Example 2 – Digits recognition

```
# see digit in row 4
```

```
y_train[4]
```

```
6
```

```
X_train.values[4]
```

```
array([ 0,  0,  5, 14,  4,  0,  0,  0,  0,  0, 13,  8,  0,  0,  0,  0,  0,
        3, 14,  4,  0,  0,  0,  0,  0,  6, 16, 14,  9,  2,  0,  0,  0,  4,
       16,  3,  4, 11,  2,  0,  0,  0, 14,  3,  0,  4, 11,  0,  0,  0, 10,
        8,  4, 11, 12,  0,  0,  0,  4, 12, 14,  7,  0,  0])
```

```
X_train.values[4].reshape(8,8)
```

```
array([[ 0,  0,  5, 14,  4,  0,  0,  0],
       [ 0,  0, 13,  8,  0,  0,  0,  0],
       [ 0,  3, 14,  4,  0,  0,  0,  0],
       [ 0,  6, 16, 14,  9,  2,  0,  0],
       [ 0,  4, 16,  3,  4, 11,  2,  0],
       [ 0,  0, 14,  3,  0,  4, 11,  0],
       [ 0,  0, 10,  8,  4, 11, 12,  0],
       [ 0,  0,  4, 12, 14,  7,  0,  0]])
```

each value is
for the digit darknes,
where 0 is a black pixel
and 16 is a white pixel

Example 2 – Digits recognition

```
# see digit in row 4
```

```
y_train[4]
```

```
6
```

```
X_train.values[4]
```

```
array([ 0,  0,  5, 14,  4,  0,  0,  0,  0,  0, 13,  8,  0,  0,  0,  0,  0,
        3, 14,  4,  0,  0,  0,  0,  0,  6, 16, 14,  9,  2,  0,  0,  0,  4,
       16,  3,  4, 11,  2,  0,  0,  0, 14,  3,  0,  4, 11,  0,  0,  0, 10,
        8,  4, 11, 12,  0,  0,  0,  4, 12, 14,  7,  0,  0])
```

```
X_train.values[4].reshape(8,8)
```

```
array([[ 0,  0,  5, 14,  4,  0,  0,  0],
       [ 0,  0, 13,  8,  0,  0,  0,  0],
       [ 0,  3, 14,  4,  0,  0,  0,  0],
       [ 0,  6, 16, 14,  9,  2,  0,  0],
       [ 0,  4, 16,  3,  4, 11,  2,  0],
       [ 0,  0, 14,  3,  0,  4, 11,  0],
       [ 0,  0, 10,  8,  4, 11, 12,  0],
       [ 0,  0,  4, 12, 14,  7,  0,  0]])
```

Example 2 – Digits recognition

```
# see digit in row 4
```

```
y_train[4]
```

```
6
```

```
X_train.values[4]
```

```
array([ 0,  0,  5, 14,  4,  0,  0,  0,  0,  0, 13,  8,  0,  0,  0,  0,  0,
        3, 14,  4,  0,  0,  0,  0,  6, 16, 14,  9,  2,  0,  0,  0,  4,
       16,  3,  4, 11,  2,  0,  0,  0, 14,  3,  0,  4, 11,  0,  0,  0, 10,
        8,  4, 11, 12,  0,  0,  0,  4, 12, 14,  7,  0,  0])
```

```
X_train.values[4].reshape(8,8)
```

```
array([[ 0,  0,  5, 14,  4,  0,  0,  0],
       [ 0,  0, 13,  8,  0,  0,  0,  0],
       [ 0,  3, 14,  4,  0,  0,  0,  0],
       [ 0,  6, 16, 14,  9,  2,  0,  0],
       [ 0,  4, 16,  3,  4, 11,  2,  0],
       [ 0,  0, 14,  3,  0,  4, 11,  0],
       [ 0,  0, 10,  8,  4, 11, 12,  0],
       [ 0,  0,  4, 12, 14,  7,  0,  0]])
```

Example 2 – Digits recognition

```
# see digit in row 4
```

```
y_train[4]
```

```
6
```

```
X_train.values[4]
```

```
array([ 0,  0,  5, 14,  4,  0,  0,  0,  0,  0, 13,  8,  0,  0,  0,  0,  0,
        3, 14,  4,  0,  0,  0,  0,  0,  6, 16, 14,  9,  2,  0,  0,  0,  4,
       16,  3,  4, 11,  2,  0,  0,  0, 14,  3,  0,  4, 11,  0,  0,  0, 10,
        8,  4, 11, 12,  0,  0,  0,  4, 12, 14,  7,  0,  0])
```

```
X_train.values[4].reshape(8,8)
```

```
array([[ 0,  0,  5, 14,  4,  0,  0,  0],
       [ 0,  0, 13,  8,  0,  0,  0,  0],
       [ 0,  3, 14,  4,  0,  0,  0,  0],
       [ 0,  6, 16, 14,  9,  2,  0,  0],
       [ 0,  4, 16,  3,  4, 11,  2,  0],
       [ 0,  0, 14,  3,  0,  4, 11,  0],
       [ 0,  0, 10,  8,  4, 11, 12,  0],
       [ 0,  0,  4, 12, 14,  7,  0,  0]])
```

Example 2 – Digits recognition

```
# see digit in row 4
```

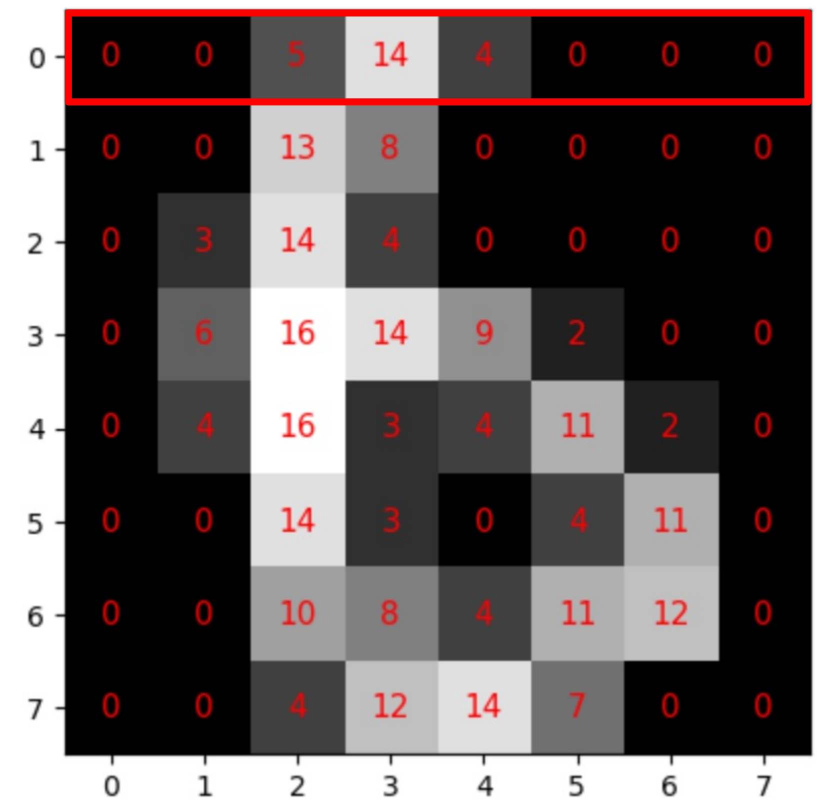
```
y_train[4]
```

```
6
```

```
X_train.values[4].reshape(8,8)
```

```
array([[ 0,  0,  5, 14,  4,  0,  0,  0],
       [ 0,  0, 13,  8,  0,  0,  0,  0],
       [ 0,  3, 14,  4,  0,  0,  0,  0],
       [ 0,  6, 16, 14,  9,  2,  0,  0],
       [ 0,  4, 16,  3,  4, 11,  2,  0],
       [ 0,  0, 14,  3,  0,  4, 11,  0],
       [ 0,  0, 10,  8,  4, 11, 12,  0],
       [ 0,  0,  4, 12, 14,  7,  0,  0]])
```

```
# 0 is a black pixel, 16 is white
```



Example 2 – Digits recognition

```
# see digit in row 4
```

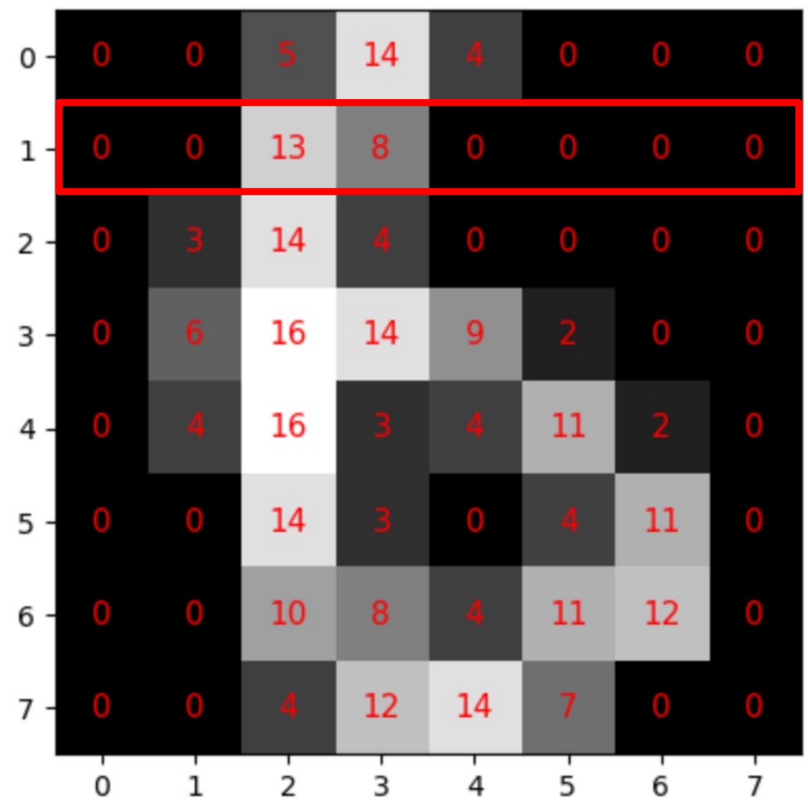
```
y_train[4]
```

```
6
```

```
X_train.values[4].reshape(8,8)
```

```
array([[ 0,  0,  5, 14,  4,  0,  0,  0],
       [ 0,  0, 13,  8,  0,  0,  0,  0],
       [ 0,  3, 14,  4,  0,  0,  0,  0],
       [ 0,  6, 16, 14,  9,  2,  0,  0],
       [ 0,  4, 16,  3,  4, 11,  2,  0],
       [ 0,  0, 14,  3,  0,  4, 11,  0],
       [ 0,  0, 10,  8,  4, 11, 12,  0],
       [ 0,  0,  4, 12, 14,  7,  0,  0]])
```

```
# 0 is a black pixel, 16 is white
```



Example 2 – Digits recognition

```
# see digit in row 4
```

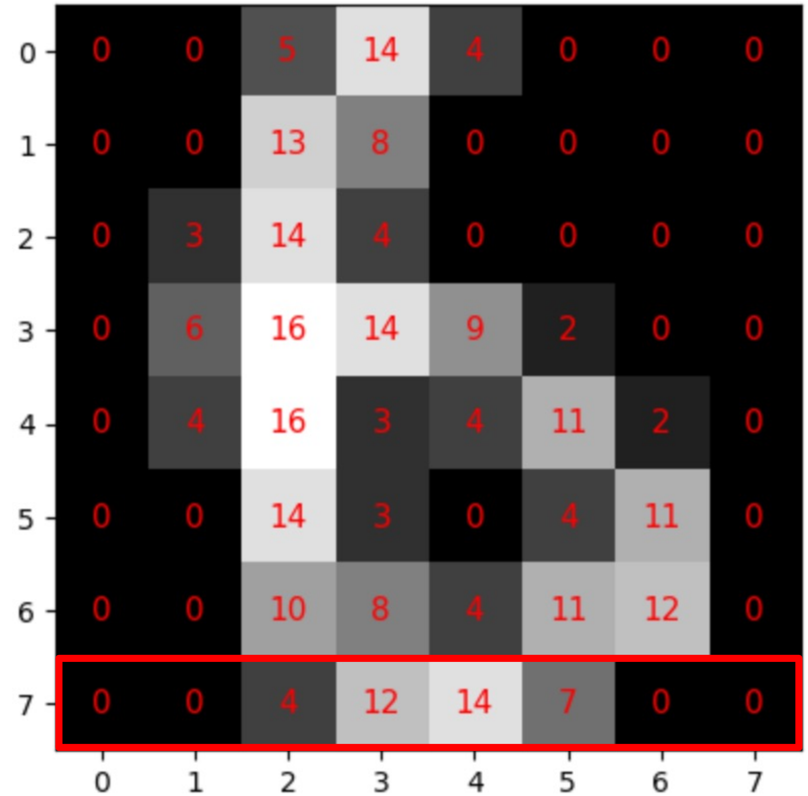
```
y_train[4]
```

```
6
```

```
X_train.values[4].reshape(8,8)
```

```
array([[ 0,  0,  5, 14,  4,  0,  0,  0],
       [ 0,  0, 13,  8,  0,  0,  0,  0],
       [ 0,  3, 14,  4,  0,  0,  0,  0],
       [ 0,  6, 16, 14,  9,  2,  0,  0],
       [ 0,  4, 16,  3,  4, 11,  2,  0],
       [ 0,  0, 14,  3,  0,  4, 11,  0],
       [ 0,  0, 10,  8,  4, 11, 12,  0],
       [ 0,  0,  4, 12, 14,  7,  0,  0]])
```

```
# 0 is a black pixel, 16 is white
```



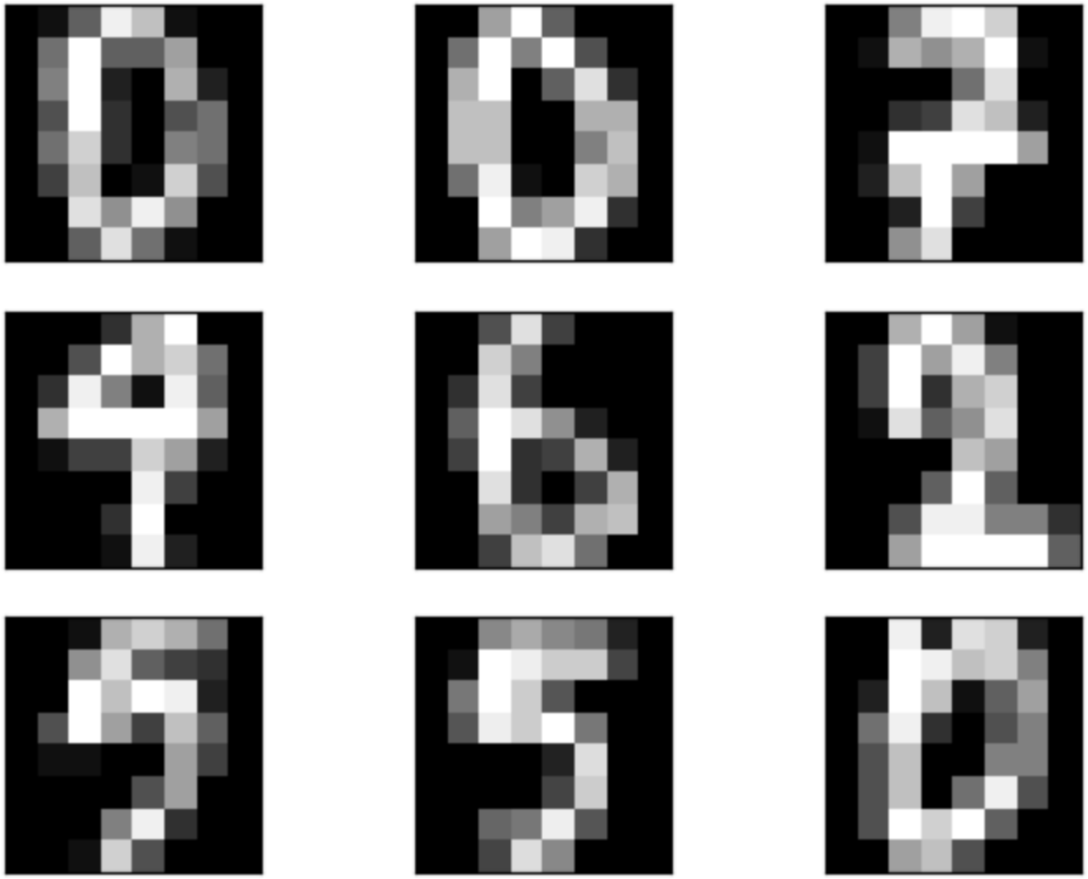
Example 2 – Digits recognition

```
# see first 9 digits in the train set
```

```
list(y_train[:9])
```

```
[0, 0, 7, 4, 6, 2, 5, 5, 0]
```

```
for k in range(0,9):
    image = X_train.values[k].reshape(8,8)
    ax = plt.subplot(3,3,k+1)
    ax.imshow(image,cmap='gray')
    plt.setp(ax,xticks=[], yticks=[])
```



Example 2 – SVC Linear kernel

```
svc = SVC(kernel = 'linear')
svc.fit(X_train,y_train);
```

```
# test accuracy rate
ypred = svc.predict(X_test)
accuracy_score(y_test,ypred)
```

0.9610461880912632

```
df1 = pd.crosstab(y_test,ypred,
                  rownames=['true labels'],
                  colnames=['predictions'])
```

predictions true labels	0	1	2	3	4	5	6	7	8	9
0	177	0	0	0	0	1	0	0	0	0
1	0	178	0	0	0	0	3	0	1	0
2	0	7	170	0	0	0	0	0	0	0
3	1	0	5	171	0	2	0	2	1	1
4	0	0	0	0	180	0	0	0	1	0
5	0	0	1	0	0	180	0	0	0	1
6	0	0	0	0	1	0	179	0	1	0
7	0	0	0	0	1	7	0	165	0	6
8	0	9	1	5	0	2	0	0	157	0
9	1	0	0	4	1	3	0	0	1	170

Example 2 – SVC Linear kernel

```
svc = SVC(kernel = 'linear')
svc.fit(X_train,y_train);
```

```
# test accuracy rate
ypred = svc.predict(X_test)
accuracy_score(y_test,ypred)
```

```
0.9610461880912632
```

```
df1 = pd.crosstab(y_test,ypred,
                  rownames=['true labels'],
                  colnames=['predictions'])
```

```
df2 = df1.copy()
df2['row sum'] = df2.sum(axis=1).values
```

predictions	0	1	2	3	4	5	6	7	8	9	row sum
true labels											
0	177	0	0	0	0	1	0	0	0	0	178
1	0	178	0	0	0	0	3	0	1	0	182
2	0	7	170	0	0	0	0	0	0	0	177
3	1	0	5	171	0	2	0	2	1	1	183
4	0	0	0	0	180	0	0	0	1	0	181
5	0	0	1	0	0	180	0	0	0	1	182
6	0	0	0	0	1	0	179	0	1	0	181
7	0	0	0	0	1	7	0	165	0	6	179
8	0	9	1	5	0	2	0	0	157	0	174
9	1	0	0	4	1	3	0	0	1	170	180

Example 2 – SVC Linear kernel

```
df2['error rate'] = 1-np.diag(df1)/np.sum(df1,axis=1)
del df2['row sum']
```

predictions true labels	0	1	2	3	4	5	6	7	8	9	error rate
0	177	0	0	0	0	1	0	0	0	0	0.005618
1	0	178	0	0	0	0	3	0	1	0	0.021978
2	0	7	170	0	0	0	0	0	0	0	0.039548
3	1	0	5	171	0	2	0	2	1	1	0.065574
4	0	0	0	0	180	0	0	0	1	0	0.005525
5	0	0	1	0	0	180	0	0	0	1	0.010989
6	0	0	0	0	1	0	179	0	1	0	0.011050
7	0	0	0	0	1	7	0	165	0	6	0.078212
8	0	9	1	5	0	2	0	0	157	0	0.097701
9	1	0	0	4	1	3	0	0	1	170	0.055556

```
svc = SVC(kernel = 'linear')
svc.fit(X_train,y_train);
```

```
# test accuracy rate
ypred = svc.predict(X_test)
accuracy_score(y_test,ypred)
```

```
0.9610461880912632
```

```
df1 = pd.crosstab(y_test,ypred,
                  rownames=['true labels'],
                  colnames=['predictions'])
```

```
df2 = df1.copy()
df2['row sum'] = df2.sum(axis=1).values
```

Example 2 – SVC Linear kernel

```
df2['error rate'] = 1-np.diag(df1)/np.sum(df1,axis=1)
del df2['row sum']
```

```
# digit with largest error rate
df2[ df2['error rate']==df2['error rate'].max()]
```

predictions	0	1	2	3	4	5	6	7	8	9	error rate
true labels											
8	0	9	1	5	0	2	0	0	157	0	0.097701

predictions	0	1	2	3	4	5	6	7	8	9	error rate
true labels											
0	177	0	0	0	0	1	0	0	0	0	0.005618
1	0	178	0	0	0	0	3	0	1	0	0.021978
2	0	7	170	0	0	0	0	0	0	0	0.039548
3	1	0	5	171	0	2	0	2	1	1	0.065574
4	0	0	0	0	180	0	0	0	1	0	0.005525
5	0	0	1	0	0	180	0	0	0	1	0.010989
6	0	0	0	0	1	0	179	0	1	0	0.011050
7	0	0	0	0	1	7	0	165	0	6	0.078212
8	0	9	1	5	0	2	0	0	157	0	0.097701
9	1	0	0	4	1	3	0	0	1	170	0.055556

```
svc = SVC(kernel = 'linear')
svc.fit(X_train,y_train);
```

```
# test accuracy rate
ypred = svc.predict(X_test)
accuracy_score(y_test,ypred)
```

```
0.9610461880912632
```

```
df1 = pd.crosstab(y_test,ypred,
                  rownames=['true labels'],
                  colnames=['predictions'])
```

```
df2 = df1.copy()
df2['row sum'] = df2.sum(axis=1).values
```

Example 2 – Best C and gamma

```
kfold = StratifiedKFold(n_splits=5,
                        shuffle = True,
                        random_state = 1)
```

```
params = {'C': [0.001, 0.01, 0.1, 1, 10],
          'gamma': [0.001, 0.01, 0.1, 1, 10]}
```

```
grid_search = GridSearchCV(SVC(), params, cv=kfold)
grid_search.fit(X_train, y_train);
```

```
grid_search.best_params_
```

```
{'C': 10, 'gamma': 0.001}
```

```
# Validation accuracy rate
grid_search.best_score_
```

```
0.991368442665024
```

```
# Test accuracy rate
grid_search.score(X_test, y_test)
```

```
0.9827490261547023
```

Example 2 – GridSearchCV functions

grid = GridSearchCV(model,params,cv,…)

grid.fit(X_train, y_train)

- grid.best_params_
- grid.best_score_ validation accuracy rate
- grid.score(X_test,y_test) test accuracy rate
- grid.best_estimator_ best model
- grid.cv_results_ CV validation acc. rates

Example 2 – Best C and gamma

```
kfold = StratifiedKFold(n_splits=5,
                        shuffle = True,
                        random_state = 1)
```

```
params = {'C': [0.001, 0.01, 0.1, 1, 10],
          'gamma': [0.001, 0.01, 0.1, 1, 10]}
```

```
grid_search = GridSearchCV(SVC(), params, cv=kfold)
grid_search.fit(X_train, y_train);
```

```
grid_search.best_params_
```

```
{'C': 10, 'gamma': 0.001}
```

```
# Validation accuracy rate
grid_search.best_score_
```

```
0.991368442665024
```

```
# Test accuracy rate
grid_search.score(X_test, y_test)
```

```
0.9827490261547023
```

Example 2 – Best C and gamma

- cv_results_ has the accuracy rates of each fold and their average in column mean_test_score

```
# store the results into a Dataframe
results = pd.DataFrame(grid_search.cv_results_)
results.dtypes
```

```
mean_fit_time         float64
std_fit_time          float64
mean_score_time       float64
std_score_time        float64
param_C                object    column 4
param_gamma            object    column 5
params                 object
split0_test_score     float64
split1_test_score     float64
split2_test_score     float64
split3_test_score     float64
split4_test_score     float64
mean_test_score       float64    column 12
std_test_score        float64
```

Example 2 – Best C and gamma

- cv_results_ has the accuracy rates of each fold and their average in column mean_test_score

```
# Create a dataframe with selected columns
list1 = list([4,5,12])
df9 = results.iloc[:,list1].copy()
df9.head()
```

	param_C	param_gamma	mean_test_score
0	0.001	0.001	0.160863
1	0.001	0.01	0.137594
2	0.001	0.1	0.102015
3	0.001	1	0.162433
4	0.001	10	0.103061

```
df9.columns = ['C','gamma','arate']
df9[:13]
```

	C	gamma	arate
0	0.001	0.001	0.160863
1	0.001	0.01	0.137594
2	0.001	0.1	0.102015
3	0.001	1	0.162433
4	0.001	10	0.103061
5	0.01	0.001	0.785507
6	0.01	0.01	0.137594
7	0.01	0.1	0.102015
8	0.01	1	0.162433
9	0.01	10	0.103061
10	0.1	0.001	0.974889
11	0.1	0.01	0.140995
12	0.1	0.1	0.102015

Example 2 – Best C and gamma

```
# Show validation accuracy rates in a two-way table
df1 = df9.pivot_table('arate',columns = 'C',index = 'gamma')
df1
```

C	0.001	0.010	0.100	1.000	10.000
gamma					
0.001	0.161125	0.789169	0.976986	0.989277	0.991368
0.010	0.147262	0.147262	0.150925	0.831550	0.844631
0.100	0.102276	0.102276	0.102276	0.106723	0.107247
1.000	0.162432	0.162432	0.162432	0.102014	0.102014
10.000	0.103322	0.103322	0.103322	0.103322	0.103322

Example 2 – Best C and gamma

```
# Show validation accuracy rates in a two-way table
df1 = df9.pivot_table('arate',columns = 'C',index = 'gamma')
df1
```

C gamma	0.001	0.010	0.100	1.000	10.000
0.001	0.161125	0.789169	0.976986	0.989277	0.991368
0.010	0.147262	0.147262	0.150925	0.831550	0.844631
0.100	0.102276	0.102276	0.102276	0.106723	0.107247
1.000	0.162432	0.162432	0.162432	0.102014	0.102014
10.000	0.103322	0.103322	0.103322	0.103322	0.103322

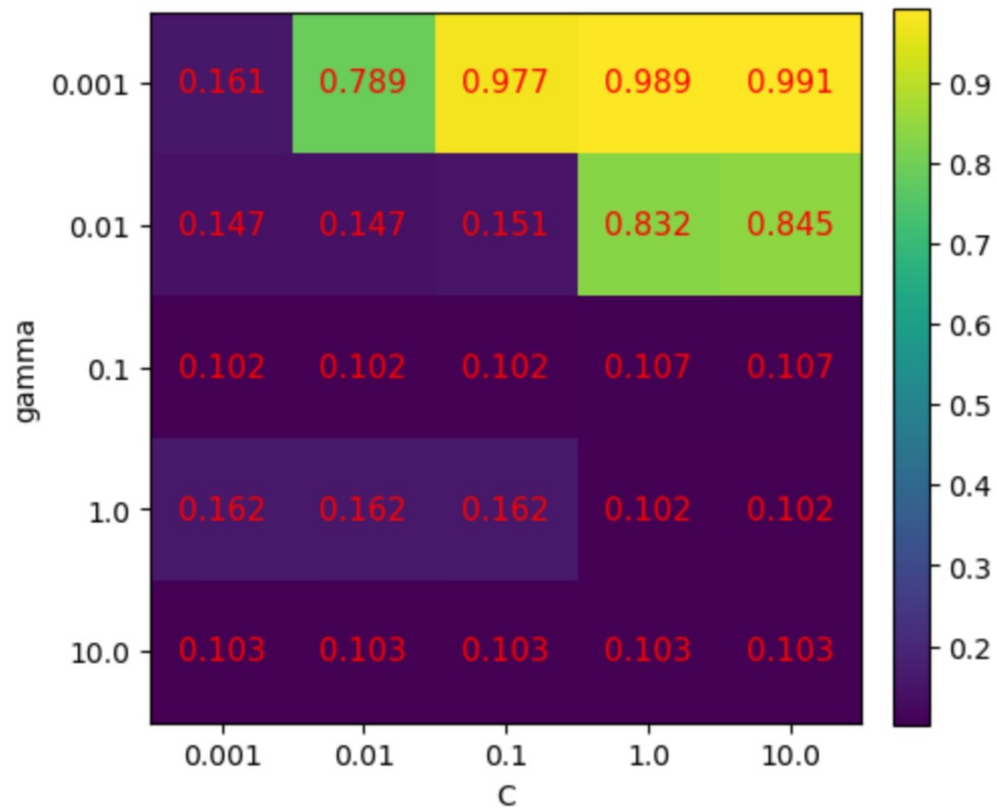
```
# Transform dataframe into array
arates = df1.values
arates = np.round(arates,3)
arates
```

```
array([[0.161, 0.789, 0.977, 0.989, 0.991],
       [0.147, 0.147, 0.151, 0.832, 0.845],
       [0.102, 0.102, 0.102, 0.107, 0.107],
       [0.162, 0.162, 0.162, 0.102, 0.102],
       [0.103, 0.103, 0.103, 0.103, 0.103]])
```

Example – Best C and gamma

```python
plt.figure(figsize = (5,5))
plt.yticks(range(5),df1.index)
plt.xticks(range(5),df1.columns)
plt.ylabel('gamma')
plt.xlabel('C')
plt.imshow(arates)
for i in range(5):
    for j in range(5):
        text = plt.text(j, i, arates[i,j],
                        ha="center",
                        va="center",
                        color="r",
                        fontsize = 11)
# resize colorbar
plt.colorbar(fraction=0.046, pad=0.04);
```

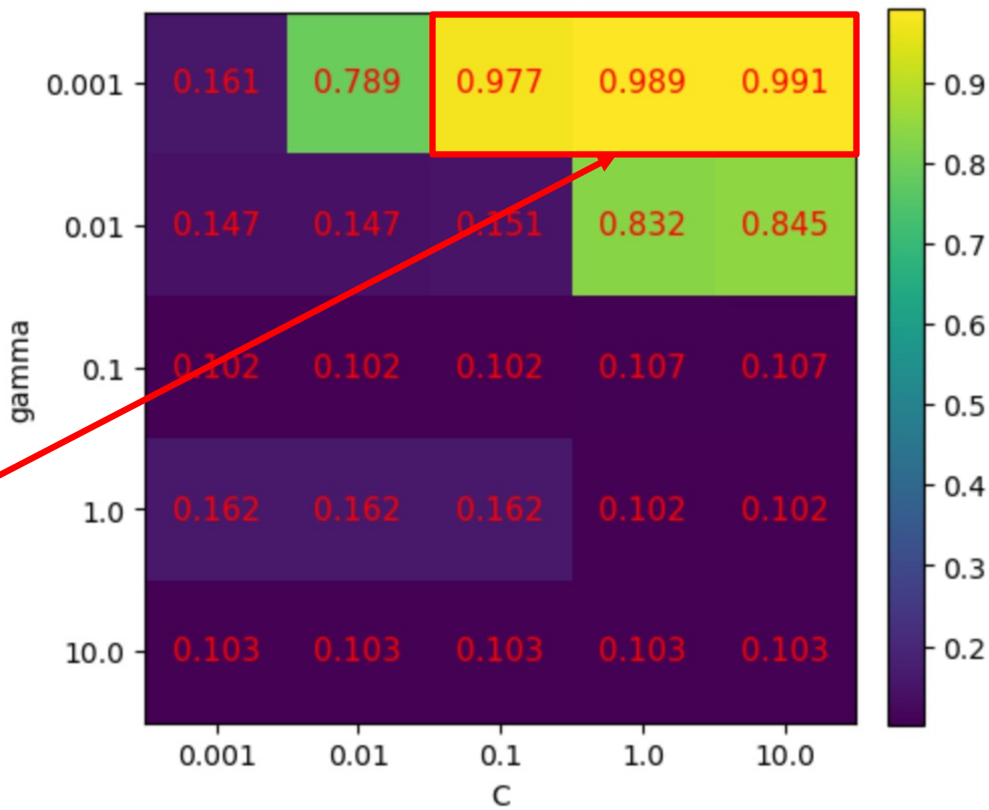
Validation Accuracy rates



Example – Best C and gamma

```
plt.figure(figsize = (5,5))
plt.yticks(range(5),df1.index)
plt.xticks(range(5),df1.columns)
plt.ylabel('gamma')
plt.xlabel('C')
plt.imshow(arates)
for i in range(5):
    for j in range(5):
        text = plt.text(j, i, arates[i,j],
                        ha="center",
                        va="center",
                        color="r",
                        fontsize = 11)
# resize colorbar
plt.colorbar(fraction=0.046, pad=0.04);
```

Validation Accuracy rates



focus on
- small gamma
- large C

Example 2 – 2nd GridSearchCV

```
kfold = StratifiedKFold(n_splits=5,
                        shuffle = True,
                        random_state = 1)
```

```
params = {'gamma': np.linspace(0.0005,0.001,5),
          'C': np.linspace(9,13,5)}
```

```
grid_search = GridSearchCV(SVC(), params, cv=kfold)
grid_search.fit(X_train, y_train);
```

```
grid_search.best_params_
```

```
{'C': 9.0, 'gamma': 0.00075}
```

```
# Validation accuracy rate
grid_search.best_score_
```

```
0.992675974403723
```

```
# Test accuracy rate
grid_search.score(X_test, y_test)
```

```
0.9833055091819699
```

Example – Best C and gamma

```
plt.figure(figsize = (5,5))
plt.yticks(range(5),df1.index)
plt.xticks(range(5),df1.columns)
plt.ylabel('gamma')
plt.xlabel('C')
plt.imshow(arates)
for i in range(5):
    for j in range(5):
        text = plt.text(j, i, arates[i,
                        ha="center",
                        va="center",
                        color="r",
                        fontsize = 11)
# resize colorbar
plt.colorbar(fraction=0.046, pad=0.04);
```

Validation Accuracy rates