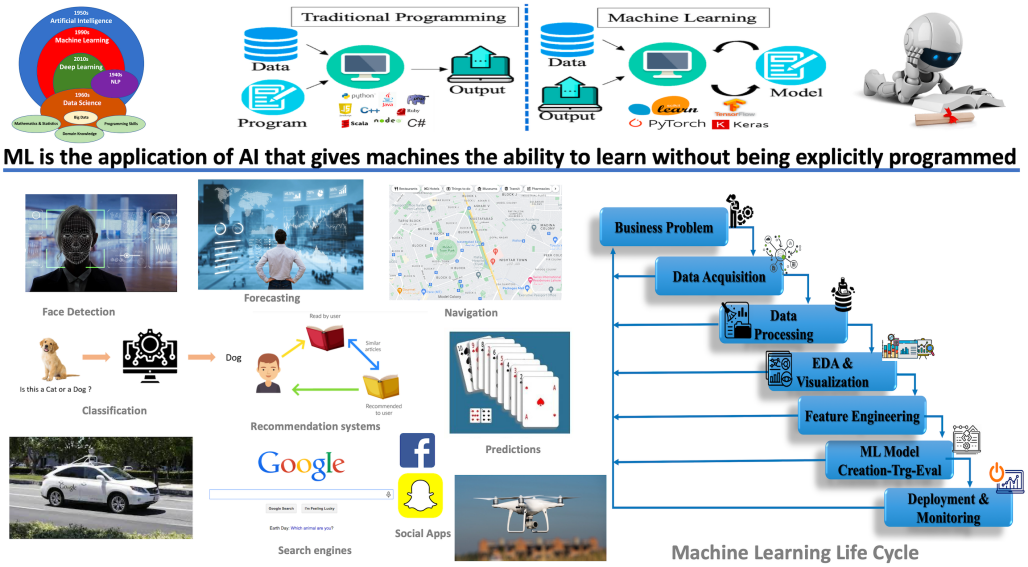
Department of Data Science

Course: Tools and Techniques for Data Science

Instructor: Muhammad Arif Butt, Ph.D.

Lecture 6.28 (Support Vector Machines Part-I)

Open in Colab
(https://colab.research.google.com/github/arifpucit/data-science/blob/master/Section-4-Mathematics-for-Data-Science/Lec-4.1(Descriptive-Statistics).ipynb)



Learning agenda of this notebook

- Support Vector Machine (A Big Picture)
  - An Overview
  - Linearly Separable Data
  - Hyperplanes as Decision Boundries
  - How to Select the Best Hyperplane?
  - How does SVM Work (A Geometric Intuition)?
- Math Behind SVM
  - Equation of a Line/Hyperplane
  - Classifying a New Data Point
  - Positive and Negative Hyperplanes
  - Decision Rule for SVM (Optimization Function and its Constraints)
  - Formula to Calculate the Margin
- Example: (Hard Margin SVM)

In [ ]:

In [ ]:

In [ ]:

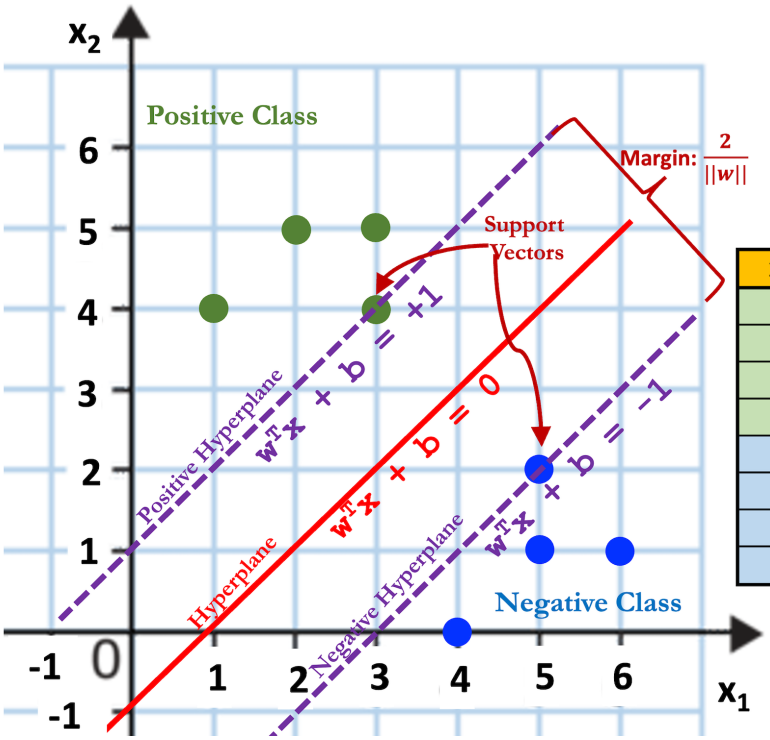
- Equation of a Line/Hyperplane
- Classifying a New Data Point
- Positive and Negative Hyperplanes
- Decision Rule for SVM (Optimization Function and its Constraints)
- Formula to Calculate the Margin

- Example: (Hard Margin SVM)

1. Support Vector Machine (A Big Picture)

a. An Overview

Support Vector Machine (SVM) is a supervised machine learning algorithm, which can be used for both classification and regression problems.



x1	x2	Class
1	4	1
2	5	1
3	5	1
3	4	1
6	1	-1
4	0	-1
5	2	-1
5	1	-1

Logistic Regression	Support Vector Machine
Used for classification alone	Used for classification as well as regression
`Probabilistic classifier` that o/p probability estimates for k classes and assigns the label of the class with the highest probability	`Deterministic classifier` that o/p a hard label without providing the probability estimate for k classes
Use statistical properties of data	Use geometric properties of data and is also called maximum margin classifier
Identifies a single hyperplane by using the idea of perceptron logic	Identifies three hyperplanes (extends the idea of perceptron logic)
Logistic Regression is a linear model and work on linear data only	SVM is a linear model but can work on non-linear data as well (kernel trick)
Sensitive to outliers and prone to overfitting	Robust to outliers and risk of overfitting is relatively lower

Identifies a single hyperplane by using the idea of perceptron logic	Identifies three hyperplanes (extends the idea of perceptron logic)
Logistic Regression is a linear model and work on linear data only	SVM is a linear model but can work on non-linear data as well (kernel trick)
Sensitive to outliers and prone to overfitting	Robust to outliers and risk of overfitting is relatively lower

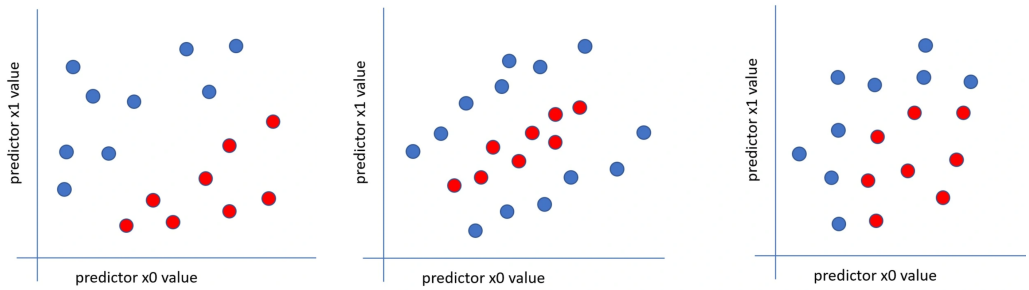
In [ ]:

In [ ]:

In [ ]:

b. Linearly Separable Data

SVM is a linear model, however, work on non-linear datasets as well by using techniques like kernel tricks.



- **Linear SVM:** When the data is perfectly linearly separable only then we can use Linear SVM. Perfectly linearly separable means that the data points can be classified into 2 classes by using a single straight line (if 2D).
- **Non-Linear SVM:** When the data is not linearly separable then we can use Non-Linear SVM, which means when the data points cannot be separated into 2 classes by using a straight line (if 2D) then we use some advanced techniques like kernel tricks to classify them. In most real-world applications we do not find linearly separable datapoints hence we use kernel trick to solve them.

In [ ]:

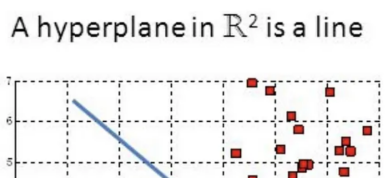
In [ ]:

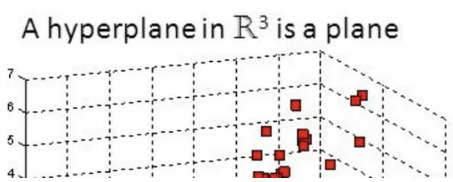
In [ ]:

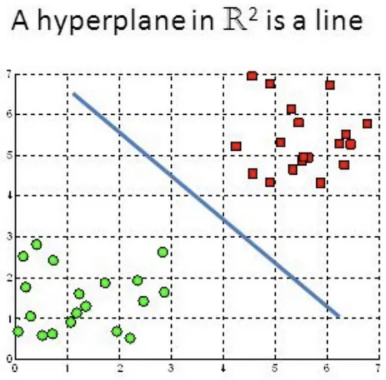
In [ ]:

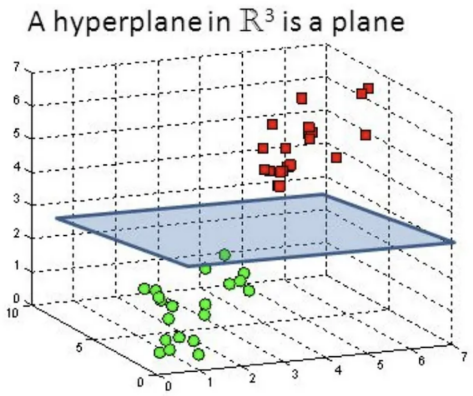
c. Hyperplanes as Decision Boundries

Hyperplanes are decision boundaries that help classify the data points









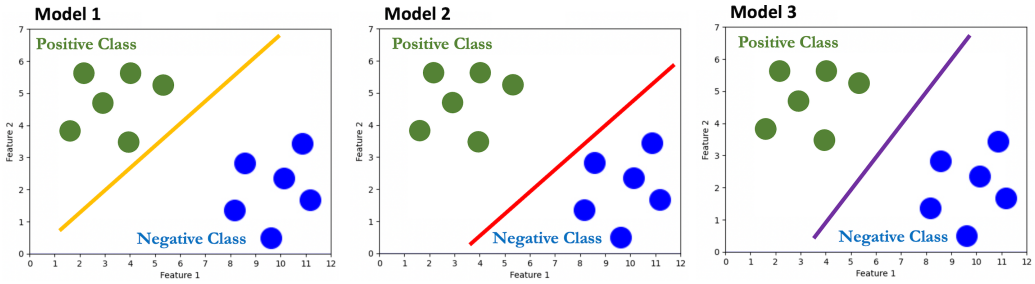
In [ ]:

In [ ]:

In [ ]:

In [ ]:

d. How to Select the Best Hyperplane?



- If the hyperplane lies very close to either the positive or negative data points, a new test point may get mis-classified

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

e. How does SVM Work (A Geometric Intuition)?

In [ ]:

In [ ]:

SVC is also called Maximum Margin Classifier as it tries to maximize the margin that separates the two classes

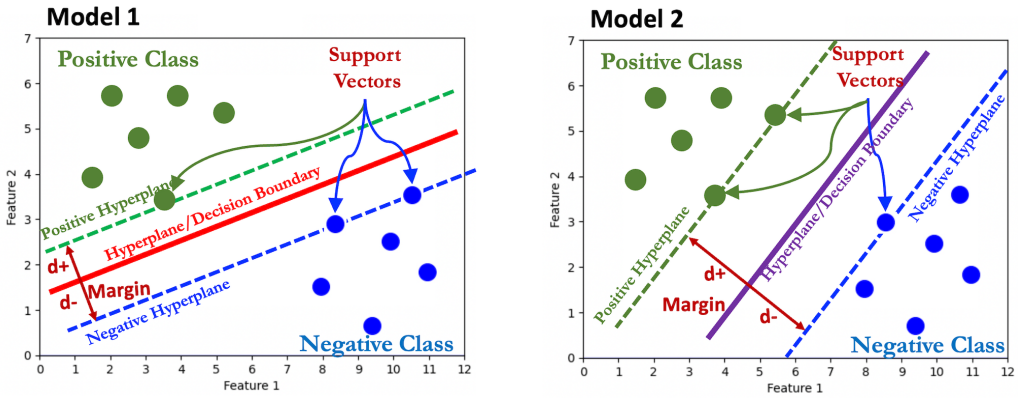
In [ ]:

e. How does SVM Work (A Geometric Intuition)?

In [ ]:

In [ ]:

SVC is also called Maximum Margin Classifier as it tries to maximize the margin that separates the two classes



Support Vectors:

- Support vectors are the data points that are `closest` to the `hyperplane` and have the greatest impact on the position of the hyperplane. These points are the most critical points in determining the position of the hyperplane and thats why the algorithm is named as Support Vector Machine.
- It's the support vectors that are used to define the hyperplane.
- SVM algorithm finds the hyperplane that `maximizes` the distance between the support vectors of the different classes.

Margin:

- The margin in SVM is the distance between the hyperplane and the closest data points from each class. It is a measure of how well the hyperplane separates the different classes in the data.
- The goal of the SVM algorithm is to find the hyperplane that maximizes the margin, as a large margin generally indicates a better separation of the classes and a more robust model.
- The margin is a key concept in SVM, as it helps to increase the model's generalization ability by maximizing the distance between the classes. This helps to reduce the chances of overfitting and also increase the robustness of the model.

Characteristics of Best Decision Boundary:

1. **Maximizes the margin.**
2. **Correctly classifies majority of the data points.**
3. **Robust to outliers and noise.**
4. **High generalization ability.**

In [ ]:

In [ ]:

In [ ]:

2. Math Behind SVM

In [ ]:

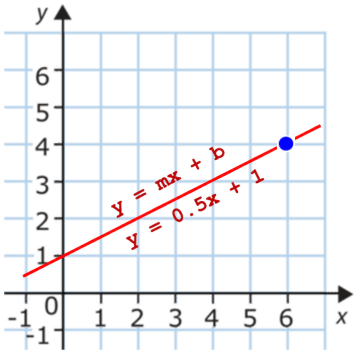
a. Equation of a Line/Hyperplane

In [ ]:

In [ ]:
2. Math Behind SVM

In [ ]:

a. Equation of a Line/Hyperplane



$$y = 0.5x + 1$$

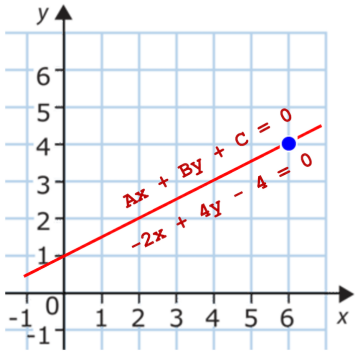
$$-0.5x + y - 1 = 0$$

$$-2x + 4y - 4 = 0$$

$$A=-2,\ B=4,\ C=-4$$

$$y = -\frac{A}{B}x - \frac{C}{B}$$

$$y = -\frac{-2}{4}x - \frac{-4}{4}$$



$$Ax_1 + Bx_2 + C = 0$$

$$w_1 x_1 + w_2 x_2 + w_0 = 0$$

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0 = 0$$

$$w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots + w_m x_m + w_0 = 0$$

$$\vec{w}\,\vec{x} + w_0 = 0$$

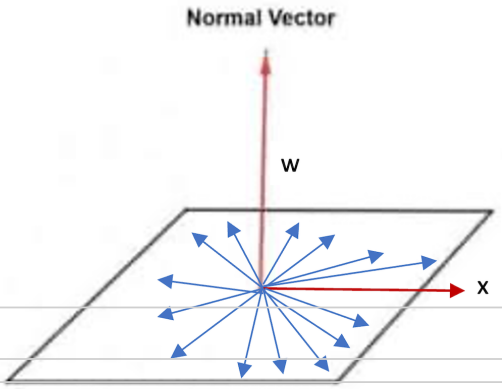
$$\vec{w}\,\vec{x} = 0$$

$$w^T x = 0$$

$$\begin{bmatrix} w_1 & w_2 & w_3 & \cdots & w_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = 0$$

Interpretation of Equation of Vector $\vec{w}$

- Since the dot product of $\vec{w}$ and $\vec{x}$ is zero, that means the angle between $\vec{w}$ and $\vec{x}$ is 90 degrees, i.e., they are orthogonal vectors
- Therefore, $\vec{w}$ is a vector which will always be perpendicular to the hyperplane (decision boundry).

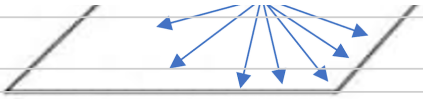


In [ ]:

In [ ]:
$$w^T x = w \cdot x = |w|\ |x|\ cos\theta = 0$$

In [ ]:

hyperplane (decision boundry).



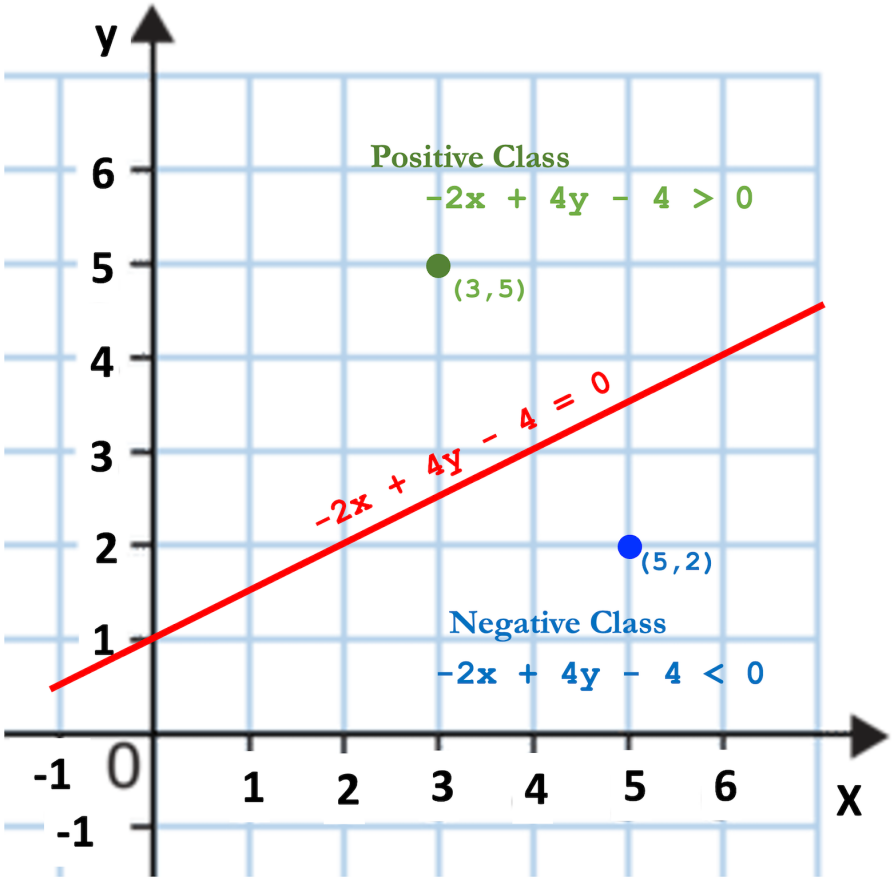
In [ ]:

In [ ]: $$w^T x = w \cdot x = |w|\ |x|\ cos\theta = 0$$

In [ ]:

In [ ]:

b.



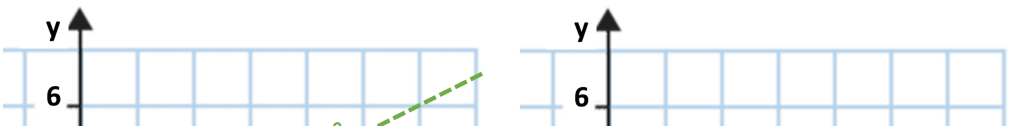
Classifying a New Data Point

- For any vector $\vec{x}$ in this entire vector space, it's corresponding predicted label $\hat{y}$ is $+1$ if $w^T x + b \geq 0$ and -1 if $w^T x + b < 0$
- This can formally be written as follows:

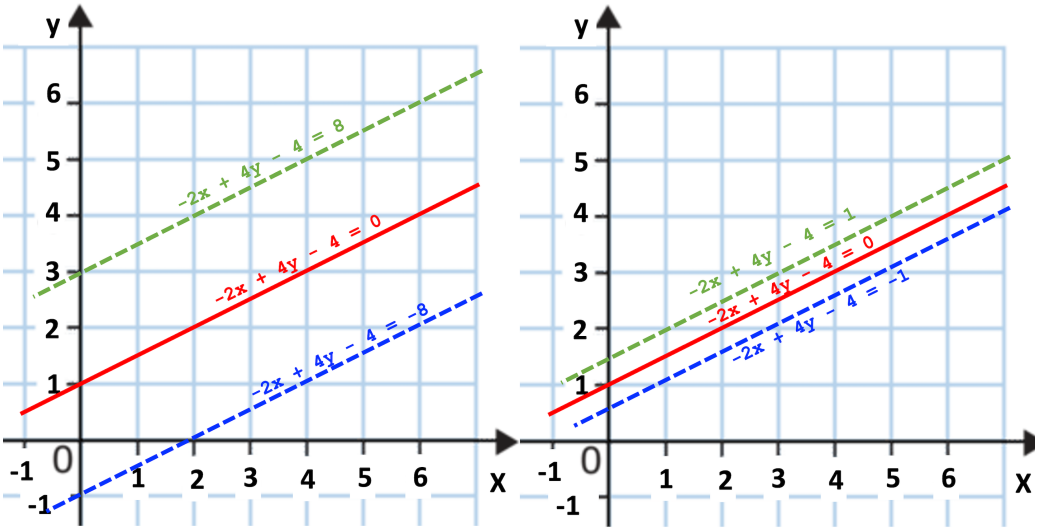
$$\hat{y} = \begin{cases} +1, & \text{if} \quad w^T x + b \geq 0 \\ -1, & \text{if} \quad w^T x + b < 0 \end{cases}$$

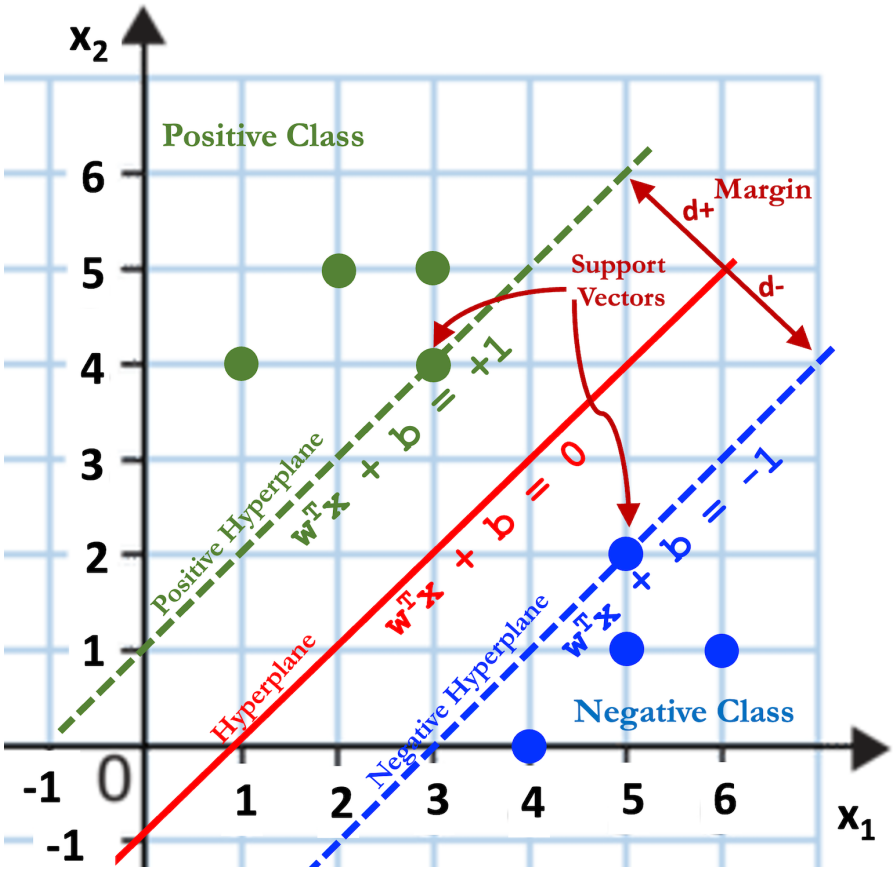
https://www.desmos.com/calculator (https://www.desmos.com/calculator)

c. Positive and Negative Hyperplanes



c. Positive and Negative Hyperplanes





https://www.desmos.com/calculator (https://www.desmos.com/calculator)

In [ ]:

In [ ]:

In [ ]:

d. Decision Rule for

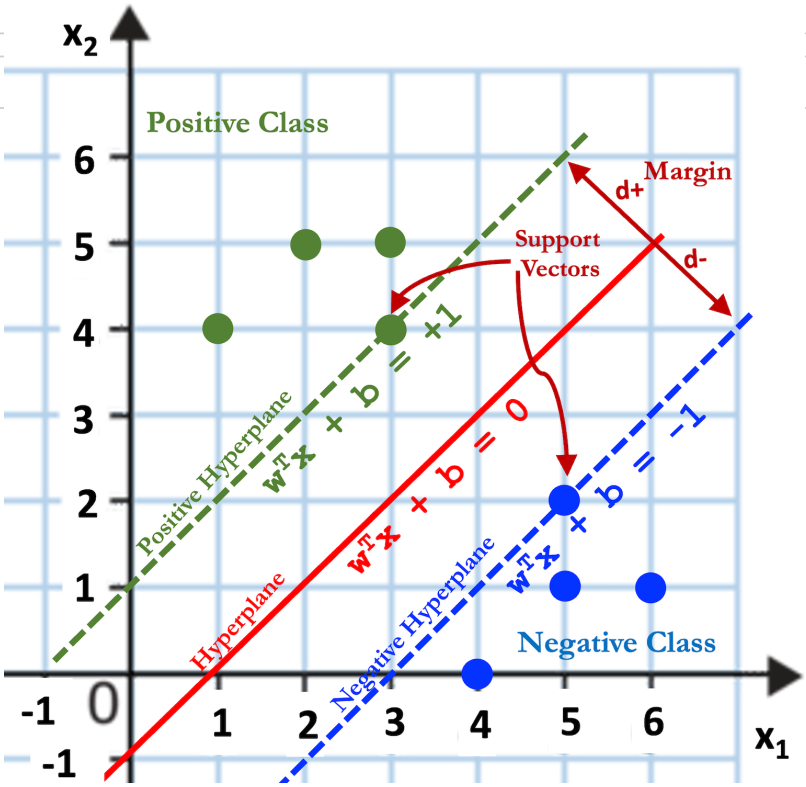
In [ ]:

In [ ]:

In [ ]:

d. Decision Rule for SVM

- General Rule to



`Classify/Predict` a New Data Point:

$$\hat{y}_i = \begin{cases} +1, & \text{if} \quad \vec{w}\vec{x_i} + b \geq 0 \\ -1, & \text{if} \quad \vec{w}\vec{x_i} + b < 0 \end{cases}$$

- **Constraint for `Hard SVM`:**
  - We need to maximize the margin, such that
    - NO training point is `below` the `Positive Hyperplane`:
      $$\vec{w}\vec{x} + b \geq 1$$
    - NO training point is be `above` the `Negative Hyperplane`:
      $$\vec{w}\vec{x} + b < -1$$
  - Simplify the above two equation to one, assuming that the label of all the positiv training points is $+1$ and the label of all the negative training points is -1:

$$\hat{y}_i = y_i(\vec{w}\vec{x_i} + b) \geq 1$$

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

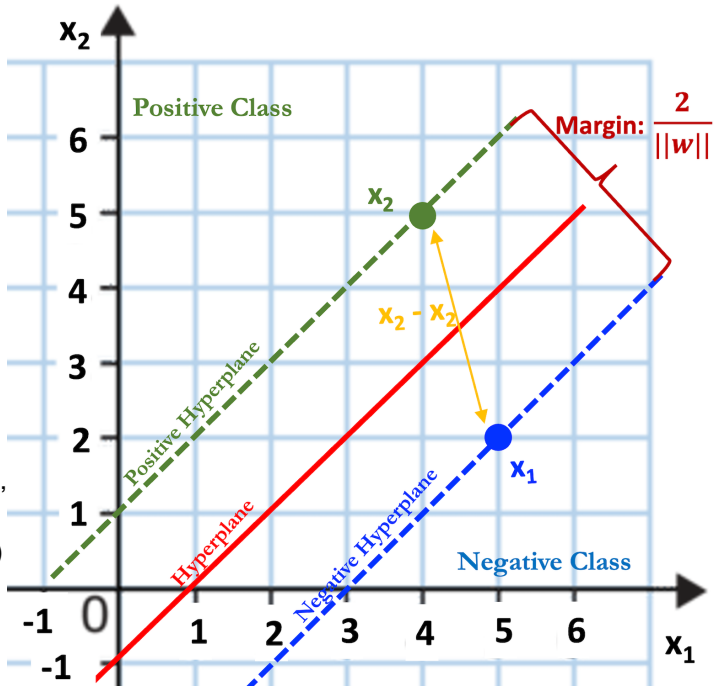
In [ ]:

In [ ]:

In [ ]:

e. Formula to Calculate the Margin

- The shortest distance between x_1 and x_2 will be perpendicular to both the positive and negative hyperplanes.
- Now we do have a $\vec{w}$, which is perpendicular to both rather all the three hyperplanes, whose unit vector (vector having a magnitude of one) can be obtained by dividing $\vec{w}$ by its magnitude, i.e., $\|w\|$.
- Moreover the margin (d) is actually the projection of $(\vec{x_2} - \vec{x_1})$ on $\vec{w}$, So



$$d = (x_2 - x_1).\frac{\vec{w}}{\|w\|}$$

$$d = \frac{x_2\vec{w} - x_1\vec{w}}{\|w\|}$$

$$d = \frac{\vec{w}x_2 - \vec{w}x_1}{\|w\|} \qquad \cdots \qquad (i)$$

- We already know that for all support vectors $y_i(\vec{w}\vec{x_i} + b) = 1$ holds.
  - Since x_2 is a positive support vector having y_i of $+1$, so $\vec{w}\vec{x_2} = 1 - b$
  - Since x_1 is a negative support vector having y_i of -1, so $\vec{w}\vec{x_1} = -1 - b$

- Substituting these values in equation (i):

$$d = \frac{1 - b - (-1 - b)}{\|w\|}$$

$$d = \frac{2}{\|w\|} \qquad \cdots \qquad (ii)$$

In [ ]:

Margin between the positive and negative hyperplane is two divided by magnitude of $\vec{w}$, i.e. $\|w\|$

In [ ]:

In [ ]:

$||w||$

In [ ]:

Margin between the positive and negative hyperplane is two divided by magnitude of $\vec{w}$, i.e. $||w||$

In [ ]:

In [ ]:

In [ ]:

f. Mathematical Formulation of SVM

$$argmax_{(w^*,b^*)} \frac{2}{||w||}, \text{such that} \quad y_i(\vec{w}\vec{x_i} + b) \geq$$

- Since 2 is a constant, so we can ignore it and the above expression can be writen as:

$$argmax_{(w^*,b^*)} \frac{1}{||w||}, \text{such that} \quad y_i(\vec{w}\vec{x_i} + b$$

- To change above into a minimization problem, we can write as:

$$argmin_{(w^*,b^*)} ||w||, \text{such that} \quad y_i(\vec{w}\vec{x_i} + b)$$

- For further ease of calculation, we can write above as:

$$argmin_{(w^*,b^*)} \frac{1}{2}||w||^2, \text{such that} \quad y_i(\vec{w}\vec{x_i} +$$

100$ Question:

How to find the values of $\vec{w}$ and b for the best decision boundry that maximizes the distance between the positive and negative hyperplanes?

¶

One of the way to solve the above constraint optimisation problem is to add one more variable (called Lagrange multiplier) for each constraint and solve it.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
In [ ]:
In [ ]:
In [ ]:
In [ ]:

Langrange Multipliers:

- In mathematical optimization, the method of Lagrange multipliers is a strategy for finding the local maxima and minima of a function subject to equation constraints (i.e., subject to the condition that one or more equations have to be satisfied exactly by the chosen values of the variables). It is named after the mathematician Joseph-Louis Lagrange.
- In the general case, for two functions $f(x)$ and $g(x)$ and Lagrange multiplier (α or λ), the Langrangian function is defined as:

$$L(x, \alpha) = f(x) + \alpha . g(x)$$

- Where, λ or α is called the Langrange multiplier
- So the above function with constraint can be written as a Langrangian function as shown below:

$$L = \frac{1}{2}\|w\|^2 - \sum \alpha_i [y_i(\vec{w}\vec{x_i} + b) - 1]$$

- In order to find the maximum or minimum of a constrained function all partial derivatives should be zero.

$$\frac{\partial L}{\partial w} = \vec{w} - \sum \alpha_i y_i \vec{x_i} \qquad \frac{\partial L}{\partial b} = -\sum \alpha_i y_i$$

$$\vec{w} - \sum \alpha_i y_i \vec{x_i} = 0 \qquad -\sum \alpha_i y_i = 0$$

$$\vec{w} = \sum \alpha_i y_i \vec{x_i} \qquad \sum \alpha_i y_i = 0$$

- Plug in the value of $\vec{w}$ in above Loss function, shown in green:

$$L = \frac{1}{2}(\sum \alpha_i y_i x_i).(\sum \alpha_i y_i x_i) - (\sum \alpha_i y_i x_i).(\sum \alpha$$

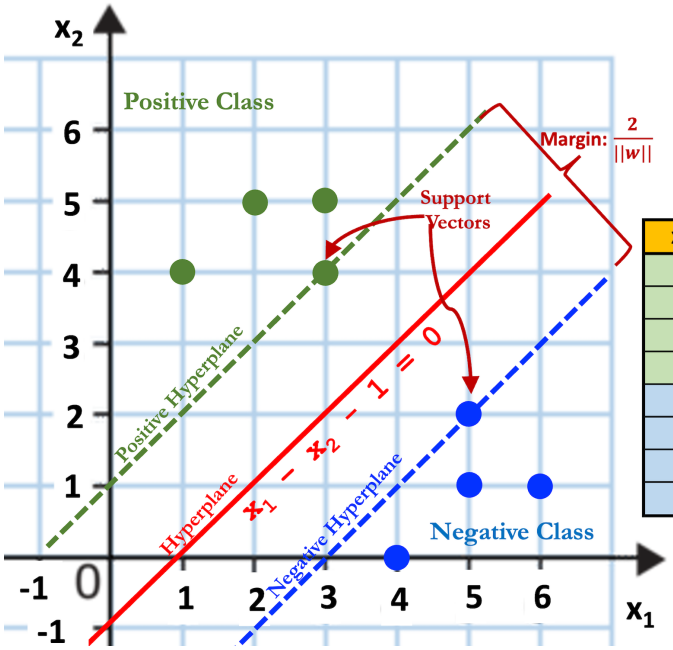
$$L = \sum \alpha_i - \frac{1}{2}\sum\sum \alpha_i \alpha_j y_i y_j x_i . x_j$$

- **The nice thing about the above objective function is that we have an expression for $\vec{w}$ in terms of Lagrange multipliers involving no $\vec{w}$ term.**

Example:

$\vec{w}$ term.

Example:



x1	x2	Class
1	4	1
2	5	1
3	5	1
3	4	1
6	1	-1
4	0	-1
5	2	-1
5	1	-1

Load the Dataset:

In [1]:
```
import numpy as np
import pandas as pd

X = np.array([[1, 4], [2, 5], [3, 5], [3, 4], [6,1], [4,0], [5,2], [5,
y = np.array([1, 1, 1, 1, -1, -1, -1, -1])
df = pd.concat([pd.DataFrame(data = X, columns = ['x1', 'x2']),    pd.
df
```

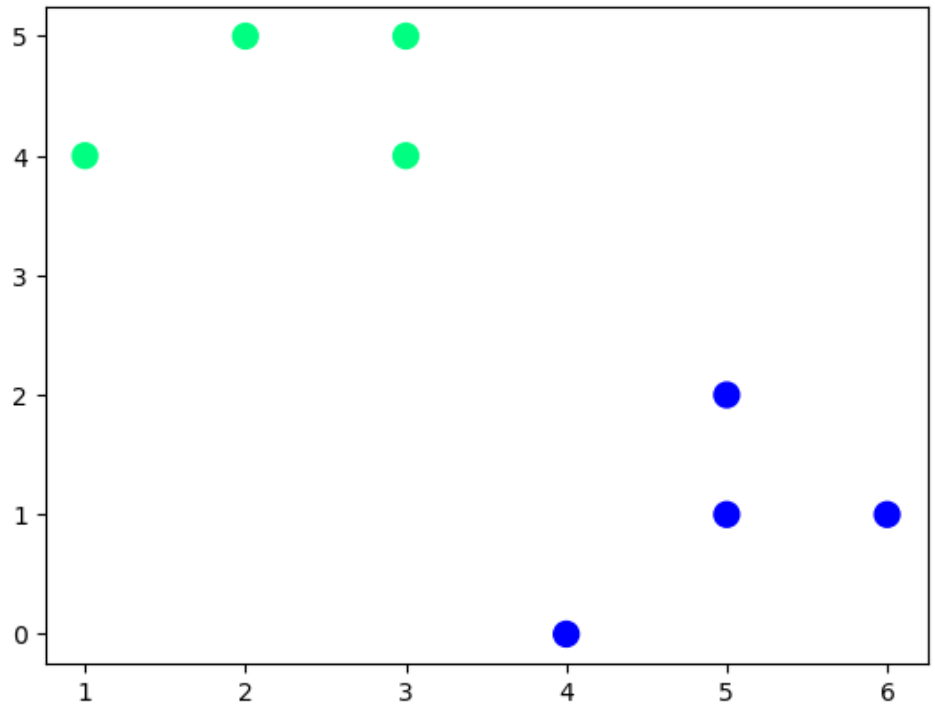
Out[1]:

	x1	x2	y
0	1	4	1
1	2	5	1
2	3	5	1
3	3	4	1
4	6	1	-1
5	4	0	-1
6	5	2	-1
7	5	1	-1

Visualize Dataset using Scatter Chart:

In [2]:
```
import matplotlib.pyplot as plt
plt.scatter(X[:, 0], X[:, 1], c=y, s=100, cmap='winter');
```

In [2]:
```
import matplotlib.pyplot as plt
plt.scatter(X[:, 0], X[:, 1], c=y, s=100, cmap='winter');
```



Create an Instance of SVC Model:

In [3]:
```
from sklearn.svm import SVC
model = SVC(kernel='linear', C=1)
```

Train the Model:

In [4]:
```
model.fit(X, y)
```

Out[4]:
```
SVC(C=1, kernel='linear')
```

Check values of different attributes:

In [5]:
```
# Get the support vectors using the support_vectors_ attribute of mode
model.support_vectors_
```

Out[5]:
```
array([[5., 2.],
       [3., 4.]])
```

In [6]:
```
# get indices of support vectors
model.support_
```

Out[6]:
```
array([6, 3], dtype=int32)
```

In [7]:
```
# get number of support vectors for each class
model.n_support_
```

Out[7]:
```
array([1, 1], dtype=int32)
```

In [8]:
```
# use the coef_ attribute to access the normal vector (w) of the hyper
# use the intercept_ attribute to access the bias (b)
w = model.coef_[0]
b = model.intercept_[0]

print('Vector (w) = ', w)
print('Bias (b) = ', b)
```

In [8]:
```
# use the coef_ attribute to access the normal vector (w) of the hyper
# use the intercept_ attribute to access the bias (b)
w = model.coef_[0]
b = model.intercept_[0]

print('Vector (w) = ', w)
print('Bias (b) = ', b)
```

```
Vector (w) =  [-0.5  0.5]
Bias (b) =  0.5
```

Create the Equation of Decision Boundry:

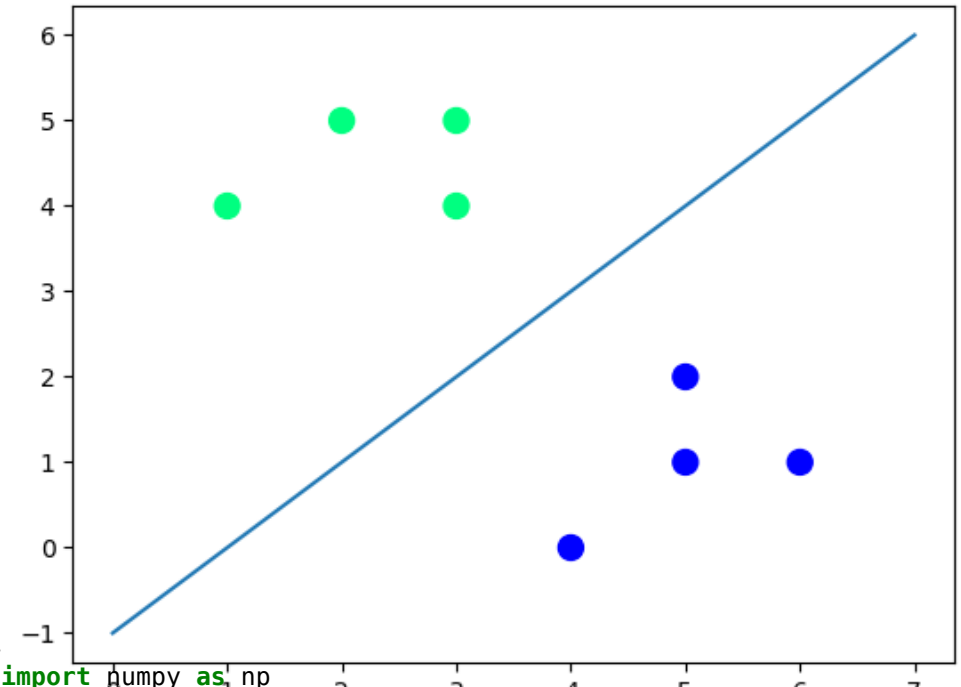
- $Ax + By + C = 0$
- $y = mx + c$

In [9]:
```
# For the general form of the equation Ax+By+C = 0, calculate the slop
A = w[0]
B = w[1]
C = b

m = -A/B # slope m = -A/B
c = -C/B # y-intercept is c = -C/B
print("Slope: ", m)

print("y-intercept: ", c)
```

```
Slope:  0.9999999999999998
y-intercept:  -0.9999999999999998
```

Plot the Hyperplane/Decision Boundry:

In [10]:
```
xx = np.linspace(0, 7, 50) # Generate 50 evenly spaced numbers betwee
yy = m * xx + c            # Calculate corresponding 50 points for out
plt.scatter(X[:, 0], X[:, 1], c=y, s=100, cmap='winter'); # use scatte
plt.plot(xx, yy);          # use plot() method to draw the line (Decis
```



In [11]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Plot the Positive, Negative Hyperplanes and Decision Boundry:

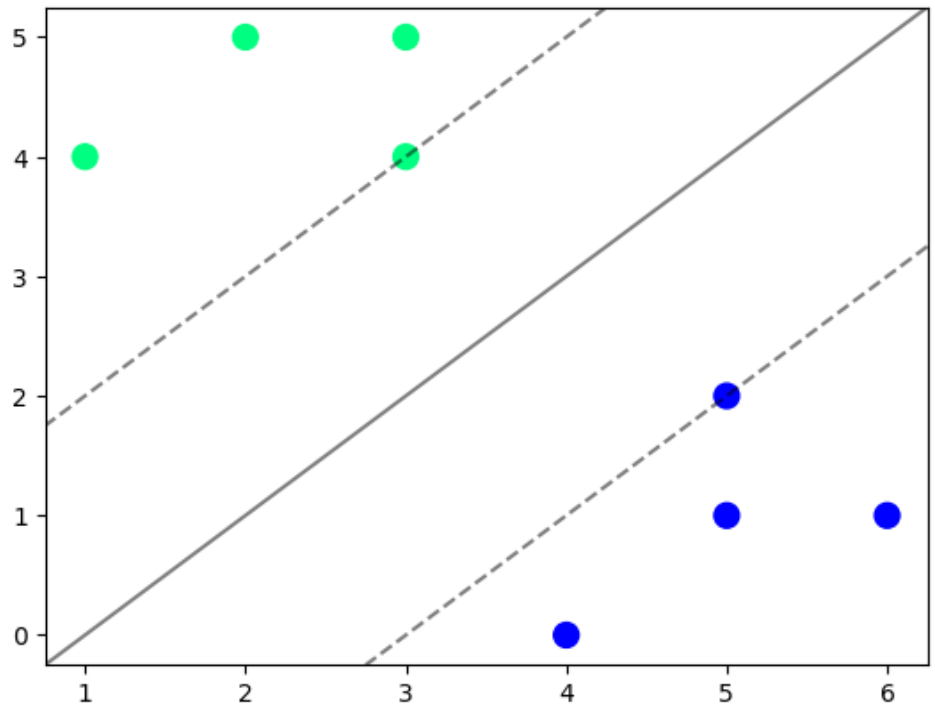
```
# User defined plotting function
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
```

In [11]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Plot the Positive, Negative Hyperplanes and Decision Boundry:

```
# User defined plotting function
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
```

In [12]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=100, cmap='winter');
plot_svc_decision_function(model);
```



In [13]:

Carry out Prediction on New Data Points:

```
point1 = [2,3]
point2 = [4,1]

print(f'Point {point1} belong to class: {model.predict([point1])[0]}')
print(f'Point {point2} belong to class: {model.predict([point2])[0]}')
```

```
Point [2, 3] belong to class: 1
```

Carry out Prediction on New Data Points:

In [13]:
```
point1 = [2,3]
point2 = [4,1]

print(f'Point {point1} belong to class: {model.predict([point1])[0]}')
print(f'Point {point2} belong to class: {model.predict([point2])[0]}')
```

```
Point [2, 3] belong to class: 1
Point [4, 1] belong to class: -1
```