Tutorial – SCS 4104 (Data Analytics)

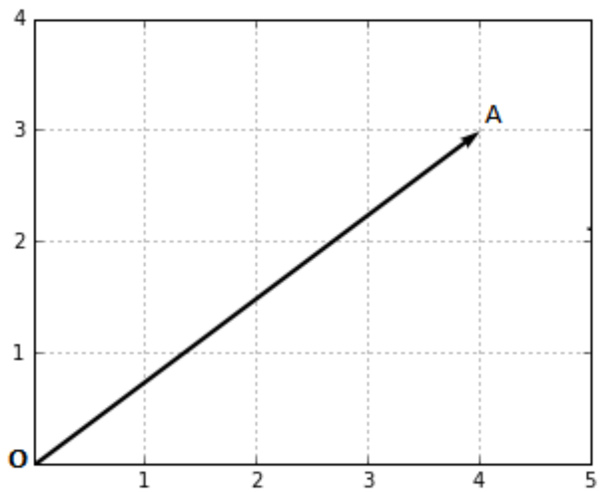
Support Vector Machines

- One of the most performant off-the-shelf supervised machine learning algorithms.

- In real life, SVMs have been successfully used in three main areas:
    - Text categorization - classifying news stories
    - Image recognition - handwritten digit recognition
    - Bioinformatics - cancer tissue samples

Vectors, Linear separability, and hyperplanes

- *A **vector** is an object that has both a magnitude and a direction.*



OA = **a** = (4,3)

- ***Norm*** of a vector is the magnitude, or length, of the vector.
- What is the norm of vector **OA**?
- Norm of a vector x = $(x_1, x_2, \ldots, x_n)$ is obtained using Euclidean norm formula.

$$\|\mathbf{x}\| := \sqrt{x_1^2 + \cdots + x_n^2}$$

- The ***direction*** of a vector is a new vector for which the coordinates are the initial coordinates of the vector divided by its norm.
- The direction of a vector $u=(u_1, u_2)$ is the vector,

$$\mathbf{w} = \left(\frac{u_1}{\|u\|}, \frac{u_2}{\|u\|}\right)$$

Task 1

- Write the following code snippet which returns the norm and direction in Python.

```
import numpy as np
x = [3,4]
np.linalg.norm(x) # 5.0

# Compute the direction of a vector x.
def direction(x):
      return x/np.linalg.norm(x)
```

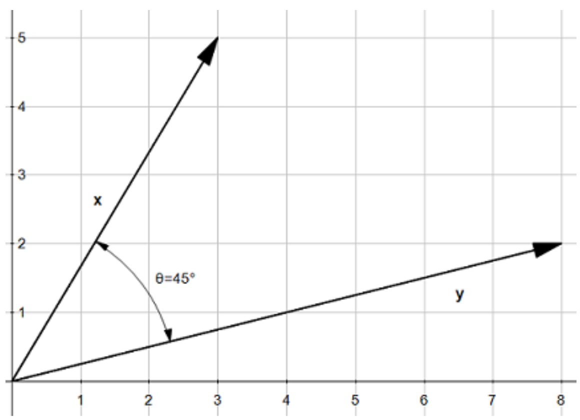
- Check the above code by executing the following.

```
u = np.array([3,4])
w = direction(u)
print(w) # [0.6 , 0.8]
```

- Obtain the direction of 2 vectors **u_1**=[3,4] and **u_2**=[30,40] and compare the answers.
- What is the norm of a direction vector? Obtain it using a Python code. unit vector. divide vector by norm.

- ***Dimensions*** of a vector
  - What are the dimensions of the vectors w=(0.6,0.8) and u=(5,3,2)?

- ***Dot product*** of two vectors



$$\mathbf{x} \cdot \mathbf{y} = \|x\| \, \|y\| \, cos(\theta)$$

Task 2

- Test the following Python code snippet which computes the dot product.

```
import math
import numpy as np
def geometric_dot_product(x,y, theta):
      x_norm = np.linalg.norm(x)
      y_norm = np.linalg.norm(y)
      return x_norm * y_norm * math.cos(math.radians(theta))
```

- Check the above code by executing the following.

```
theta = 45
x = [3,5] y = [8,2]
print(geometric_dot_product(x,y,theta)) # 34.0
```

- The dot product can also be written as

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{2}(x_i y_i)$$

or

$$\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + x_2 y_2$$

or in a more general way, for n-dimensional vectors as,

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n}(x_i y_i)$$

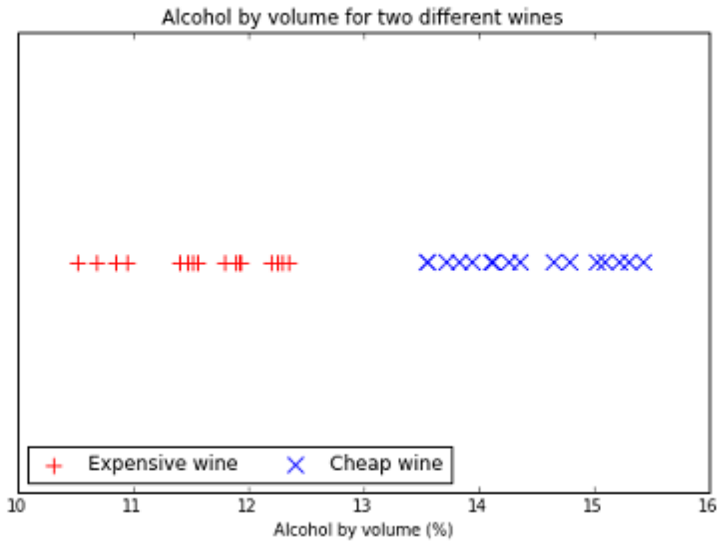
Task 3

- Test the following Python code snippet which computes the algebraic dot product.

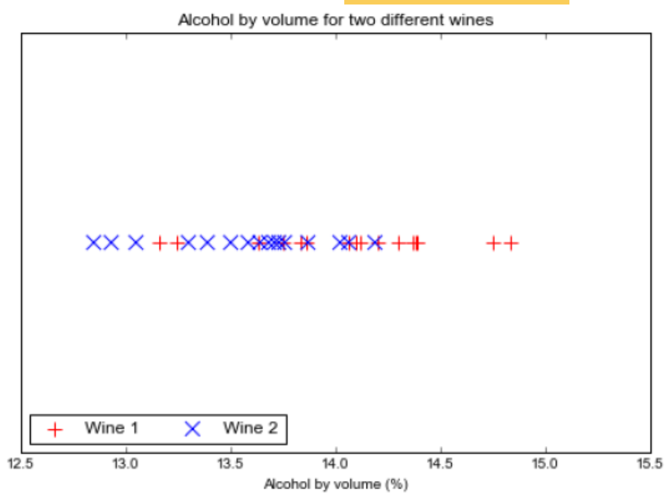
```
def dot_product(x,y):
      result = 0
      for i in range(len(x)):
            result = result + x[i]*y[i]
      return result
```

- Obtain the dot product of the two vector x=[3,5] and y=[8,2] using the above function.

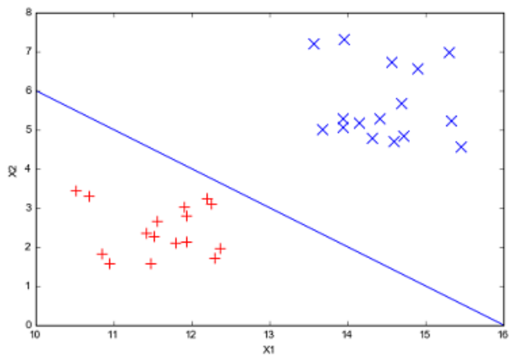
- ***Linear Separability***
  - Example on using alcohol-by-volume to classify wine.



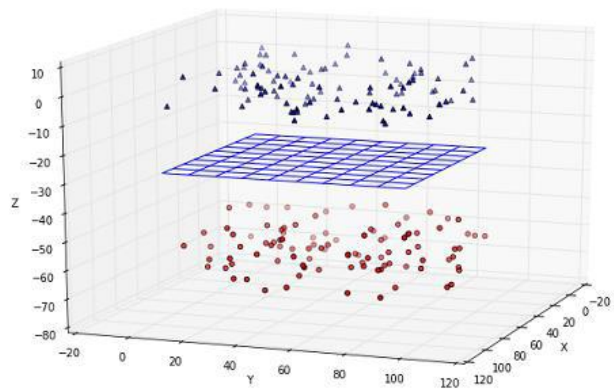
  - Notice that the above data is linearly separable.



  - In most cases, we will start from the linearly separable case (because it is the simpler) and then derive the non-separable case.
  - In real world examples we do not deal with only one/ two dimensions, but rather with thousands of dimensions.
  - Data is linearly separable when,
    - In one dimension, you can find a **point** separating the data.
    - In two dimensions, you can find a **line** separating the data.



- In three dimensions, you can find a **plane** separating the data.



- A **hyperplane** is a subspace of one dimension less than its ambient space.
  - Equation of Hyperplane: w.x+b=0
  - The equation of a line is derivable using the equation of a Hyperplane because a line is also a hyperplane.
  - Classifying data with a hyperplane is performed using a **hypothesis function** as follows.

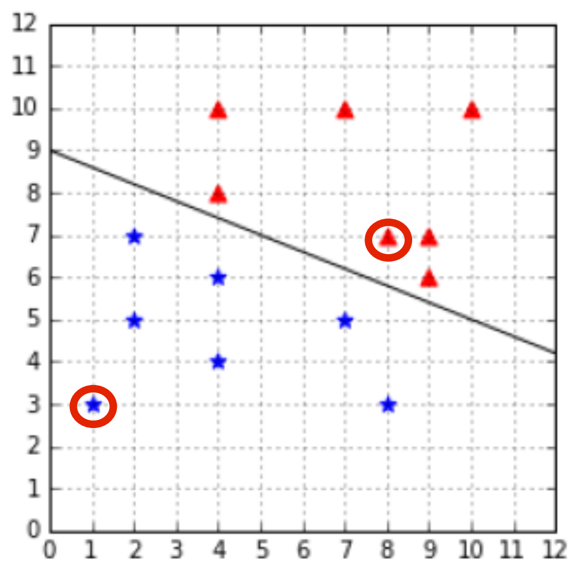
$$h(\mathbf{x}_i) = \begin{cases} +1 & \text{if} \quad \mathbf{w}\cdot\mathbf{x}_i + b \geq 0 \\ -1 & \text{if} \quad \mathbf{w}\cdot\mathbf{x}_i + b < 0 \end{cases}$$

which is equivalent to:

$$h(\mathbf{x}_i) = \text{sign}(\mathbf{w}\cdot\mathbf{x}_i + b)$$

Task 4

- Predict a value for the label y, for the x values, (8,7) and (1,3) based on the following figure.

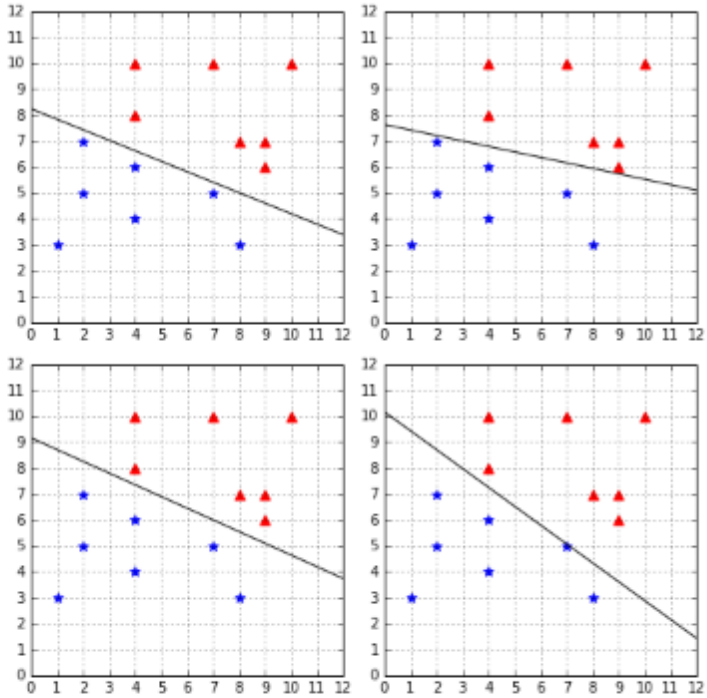


- Therefore, a hyperplane could be considered as a ***linear classifier***.
- The goal of a learning algorithm trying to learn a linear classifier is to find a hyperplane separating the data. Finding that hyperplane is equivalent to finding a vector **w**.

<u>Perceptron algorithm</u>

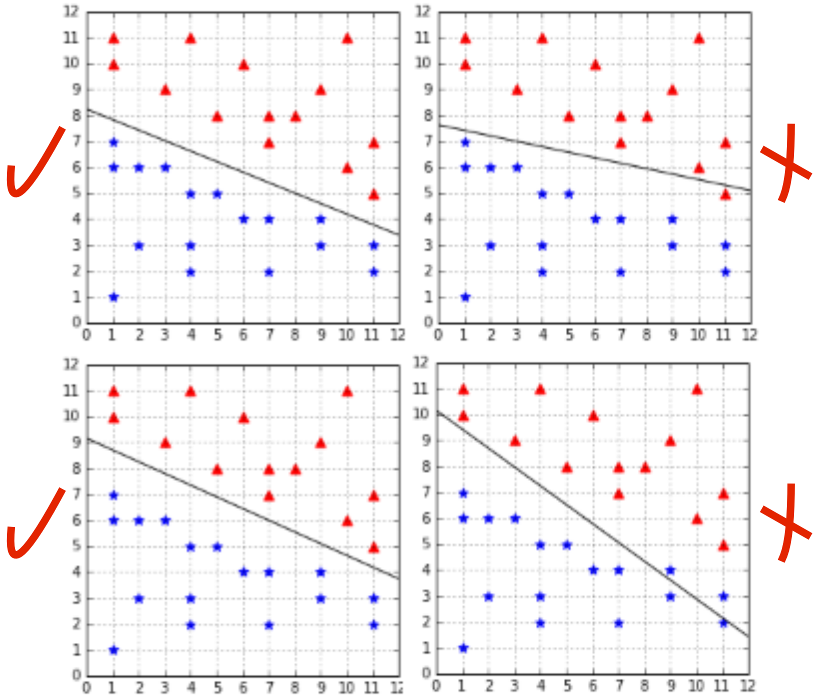
- Perceptron is the building block of a simple neural network.
- The goal of the Perceptron is to find a hyperplane that can separate a linearly separable data set.
- It means that the goal of the algorithm is to find a value for w.
- There is an infinite number of hyperplanes (you can give any value to w).
- Following is the ***PLA algorithm***
  1. Start with a random hyperplane (defined by a vector) and use it to classify the data.
  2. Pick a misclassified example and select another hyperplane by updating the value of, hoping it will work better at classifying this example (this is called the update rule).
  3. Classify the data with this new hyperplane.
  4. Repeat steps 2 and 3 until there is no misclassified example.

- How do we ensure that number of misclassified samples decrease with the update rule?
  - Perceptron convergence theorem guarantees that if the two sets P and N (of positive and negative examples respectively) are linearly separable, the vector is updated only a finite number of times.
  - The PLA finds a different hyperplane each time.



Hyperplanes for Training dataset

- Our goal is not to find a way to classify perfectly the data we have right now. Our goal is to find a way to correctly classify new data we will receive in the future.
- Terminology:
  - Training set -> In-sample error (training error)
  - Test set -> Out-of-sample error (generalization error) ; **Our goal is to have the smallest out-of-sample error**.
  - Not all hyperplanes provide a perfect out-of-sample error (consider the figure below).

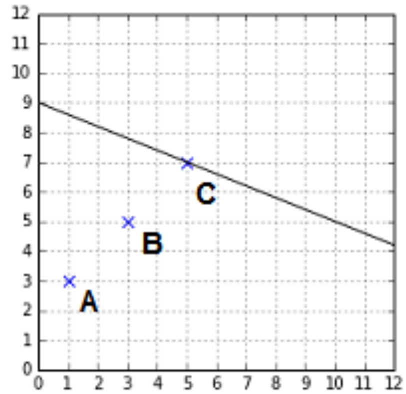


Hyperplanes for Testing dataset

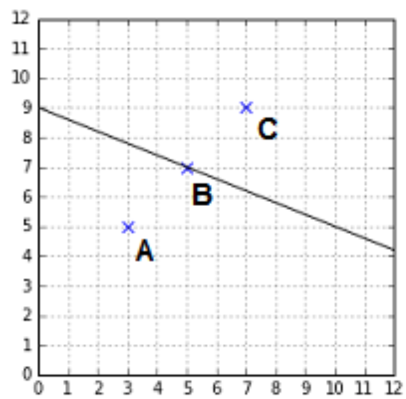
SVM optimization problem

- ***Advantages*** of Perceptron:
    - A simple model
    - Algorithm is easy to implement
    - It is theoretically proven that it will find a hyperplane that separates the data.
- ***Disadvantage*** of Perceptron is that it will not find the same hyperplane every time. Since not all hyperplanes are equal, it is important to find the optimal hyperplane.
- If the Perceptron gives you a hyperplane that is very close to all the data points from one class, you have a right to believe that it will generalize poorly when given new data.

- SVM is the solution to the above problem. SVM tries to find the ***optimal hyperplane***; the hyperplane that best separates the data.
- How can we find the optimal hyperplane from 2 given hyperplanes?
    - Consider the 3 points A,B and C, given below together with the Hyperplane (w=(-0.4,-1) and b=9).



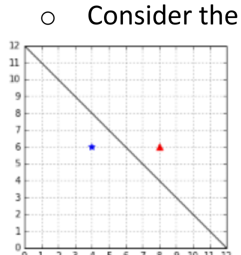
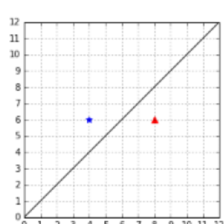
- Calculate w.x+b for the 3 points A, B and C.
- What are the observations for C (which is on the hyperplane), B (closer to the hyperplane) and A (far away from the hyperplane)?
- Consider the figure below and comment on the sign of the numbers obtained for A, B and C.



- So if, β= w.x.+b, compute β for each training example, and obtain the smallest β as number B.

$$B = \min_{i=1...m} \beta_i$$

- Out of 2 given hyperplanes select B which is the largest. So, if we have k hyperplanes, obtain, $\max_{i=1...k} B_i$ and select the hyperplane having this B_i.
- In order to avoid problems with examples on the negative side, consider B as the β having the smallest absolute value. $B = \min_{i=1...m} |\beta_i|$
- Consider the 2 hyperplanes given below. Value of B for both is 2.

- In order to determine the hyperplane which correctly classifies data, obtain a new number called f (also known as ***functional margin***),
  f = y * β
  f = y(w.x +b)

  y is the true value
  Beta is the predicted value
  if negative —> incorrect

- The sign of f will always be
  - Positive if the point is correctly classified.
  - Negative if the point is incorrectly classified.

- Given a dataset D, compute F using,

$$F = \min_{i=1,\ldots,m} f_i$$

$$F = \min_{i=1\ldots m} y_i(\mathbf{w} \cdot \mathbf{x}_i + b)$$

- Given two hyperplanes, select the one for which F is the largest.

- Following Python code snippet is used to obtain f (functional margin).

```
# Compute the functional margin of an example (x,y)
# with respect to a hyperplane defined by w and b.
def example_functional_margin(w, b, x, y):
      result = y * (np.dot(w, x) + b)
      return result
      # Compute the functional margin of a hyperplane # for examples X
with labels y.
def functional_margin(w, b, X, y):
      return np.min([example_functional_margin(w, b, x, y[i])
            for i, x in enumerate(X)])
```

- However, f is not scale invariant.

  NOT Scale Invariant since results are 8 and 80
  If you resize, you get different values

```
x = np.array([1, 1])
y = 1

b_1 = 5
w_1 = np.array([2, 1])

w_2 = w_1 * 10
b_2 = b_1 * 10

print(example_functional_margin(w_1, b_1, x, y)) # 8
print(example_functional_margin(w_2, b_2, x, y)) # 80
```

- F=y(w.x)+b

$$\gamma = y\left(\frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x} + \frac{b}{||\mathbf{w}||}\right)$$

$$M = \min_{i=1...m} \gamma_i$$

$$M = \min_{i=1...m} y_i\left(\frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x} + \frac{b}{||\mathbf{w}||}\right)$$

To make it Scale Invariant

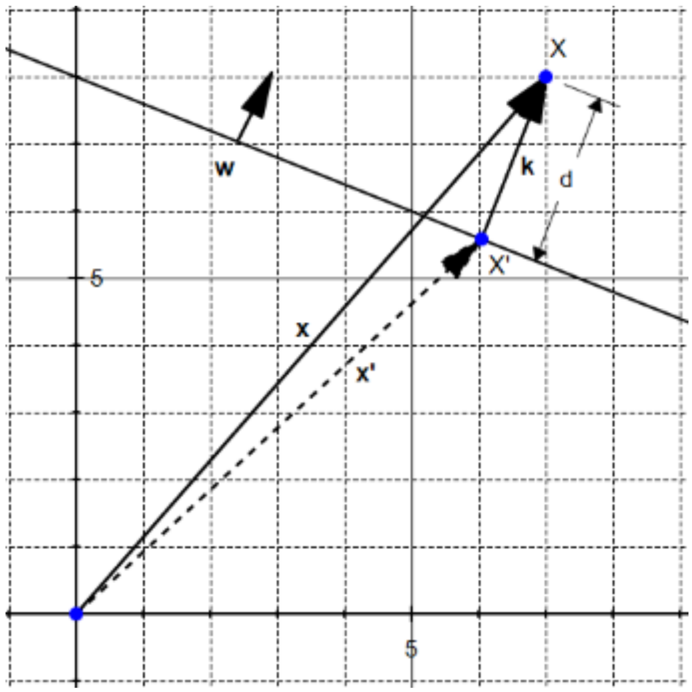
- Γ gives the same number no matter how large the vector w is. **Γ** is the **geometric margin of the training example**, while **M** is the **geometric margin of the dataset.**

```
# Compute the geometric margin of an example (x,y)
# with respect to a hyperplane defined by w and b.
def example_geometric_margin(w, b, x, y):
      norm = np.linalg.norm(w)
      result = y * (np.dot(w/norm, x) + b/norm)
      return result
# Compute the geometric margin of a hyperplane
# for examples X with labels y.
def geometric_margin(w, b, X, y):
      return np.min([example_geometric_margin(w, b, x, y[i])
            for i, x in enumerate(X)])
```

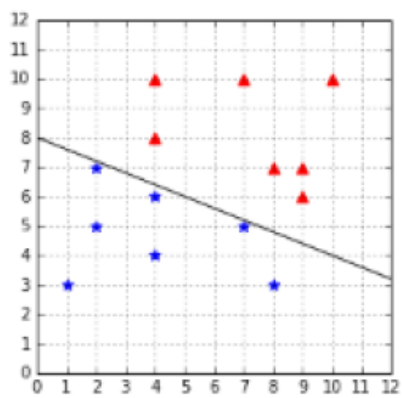
- Check the output for the two vectors w_1 and its rescaled version w_2.

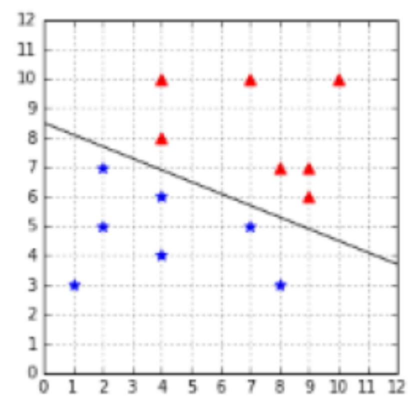
```
x = np.array([1,1])
y = 1
b_1 = 5
w_1 = np.array([2,1])
w_2 = w_1*10
b_2 = b_1*10                                     Same value even if you rescale
print(example_geometric_margin(w_1, b_1, x, y)) # 3.577708764
print(example_geometric_margin(w_2, b_2, x, y)) # 3.577708764
```

- The geometric margin measures the distance between x and the hyperplane.



- Consider the two hyperplanes given below. What can you observe about their geometric margins?





- Find the geometric margins of the two hyperplanes shown above, using the code snippet given below.

```
# Compare two hyperplanes using the geometrical margin.
positive_x = [[2,7],[8,3],[7,5],[4,4],[4,6],[1,3],[2,5]]
negative_x = [[8,7],[4,10],[9,7],[7,10],[9,6],[4,8],[10,10]]

X = np.vstack((positive_x, negative_x))
y = np.hstack((np.ones(len(positive_x)), -1*np.ones(len(negative_x))))

w = np.array([-0.4, -1])
b = 8
# change the value of b
print(geometric_margin(w, b, X, y)) # 0.185695338177
print(geometric_margin(w, 8.5, X, y)) # 0.64993368362
```

- Thus, what is the hyperplane that we should choose?
  Tip: Finding the optimal hyperplane is just a matter of finding the values of w and b for which we get the largest geometric margin.

- Therefore, we could conclude that SVMs are better at classifying data.

Task 5

- Derive the SVM formulation by using the above mentioned definitions.
- Obtain the quadratic programming formulation for SVM

Task 6

- By using the convex optimization software named CVX, optimize the above mentioned quadratic programing.