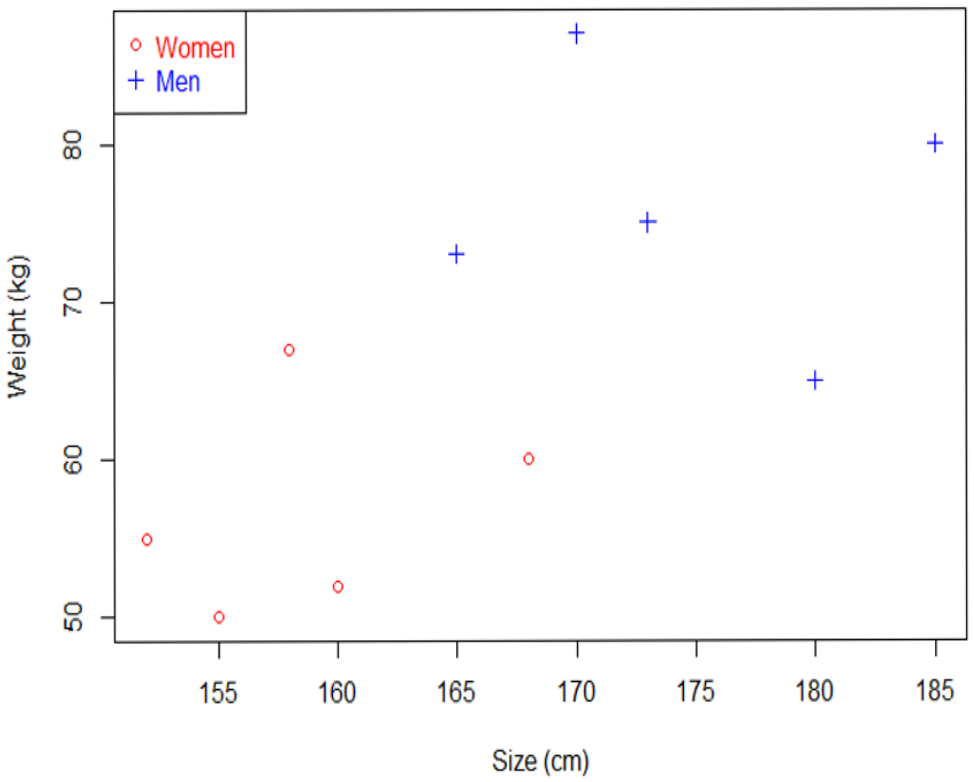
Support Vector Machines Internals

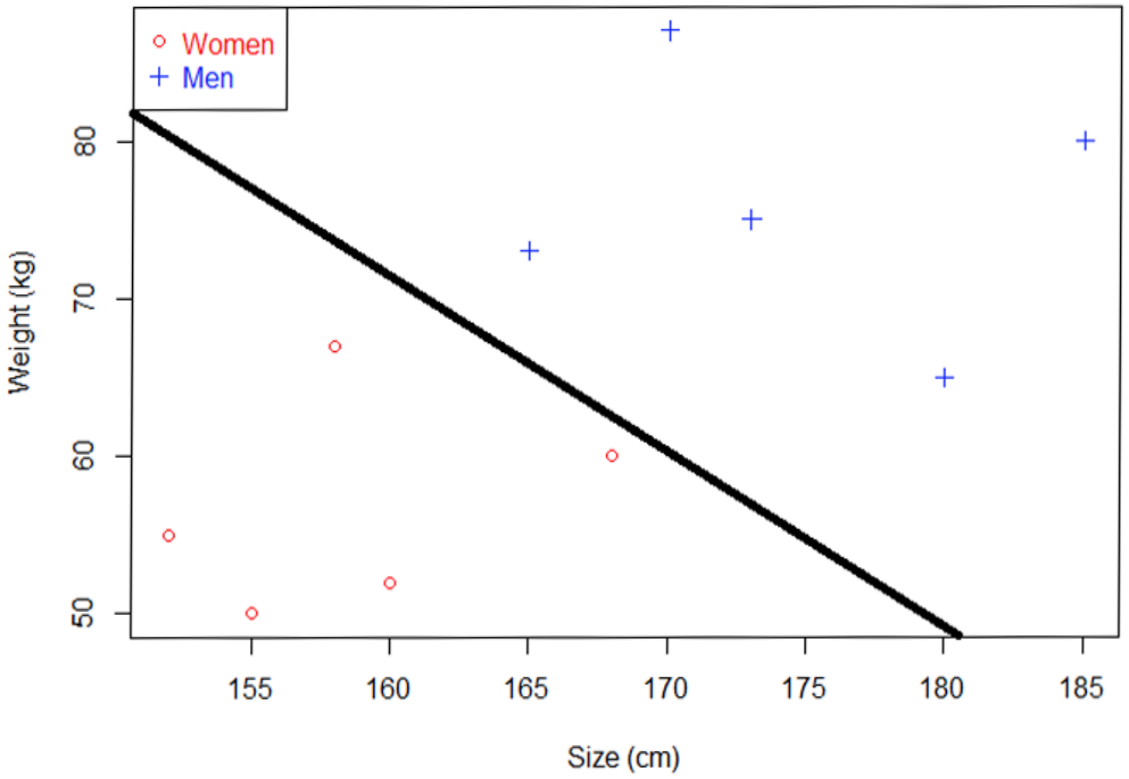
By Mohit Kumar

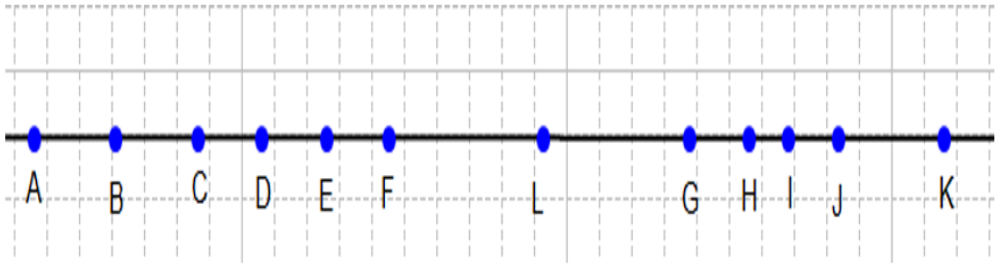
Support Vector Machine:Goal



- SVM is primarily a classification algorithm
- The goal of SVM is to find the optimal seperating hyperplane.

Support Vector Machine:Hyperplane

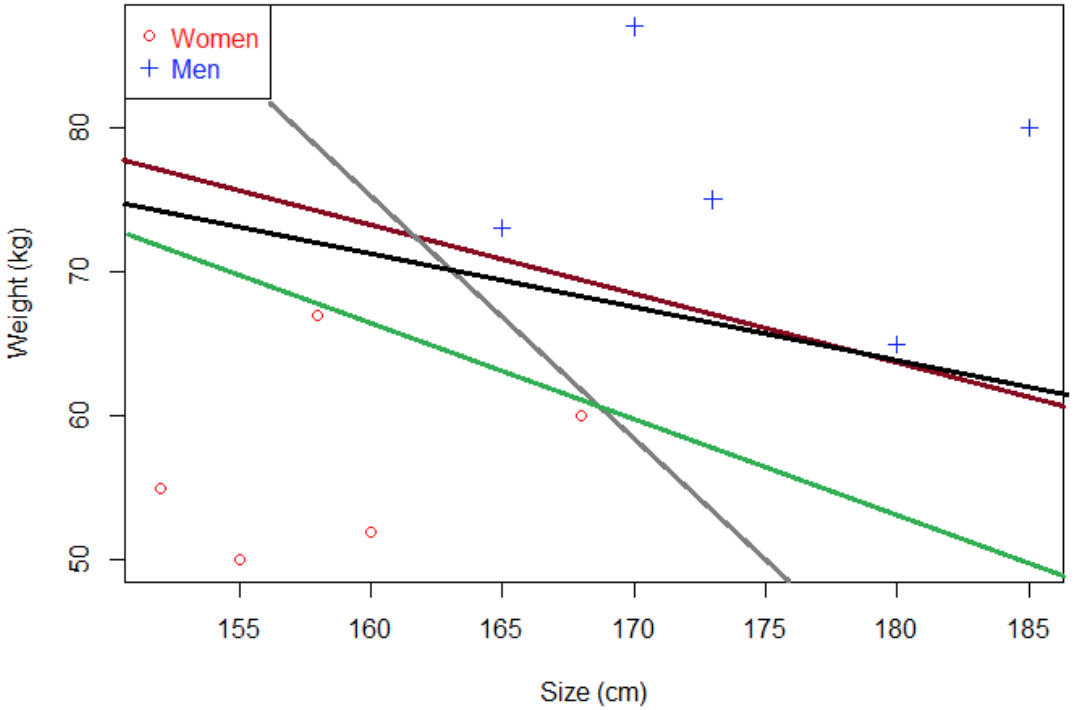


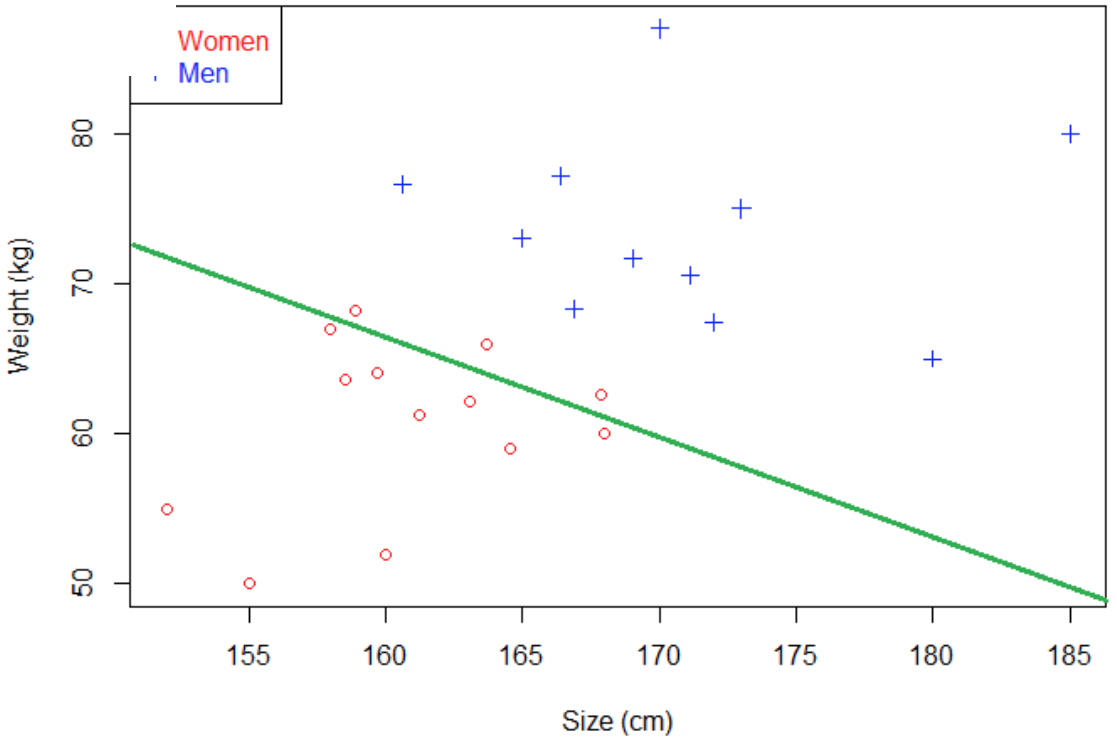


. Hyperplane is generalization of a plane

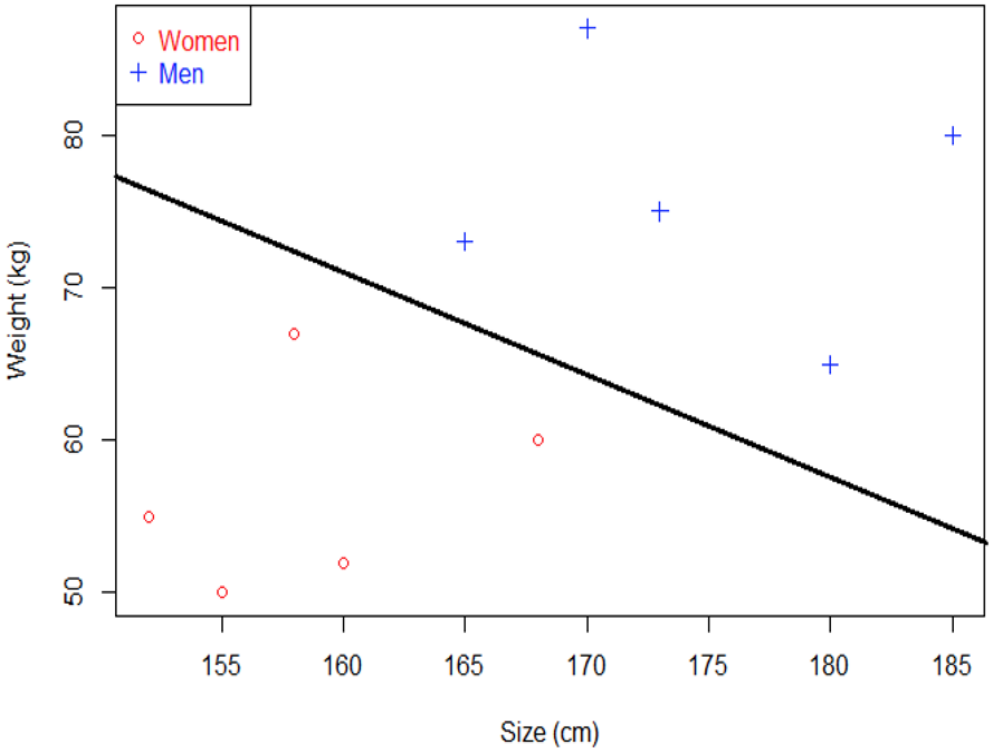
- In one dimension, it is a point.
- In 2 dimensions, it is a line and so on.

Support Vector Machine:Optimally separating

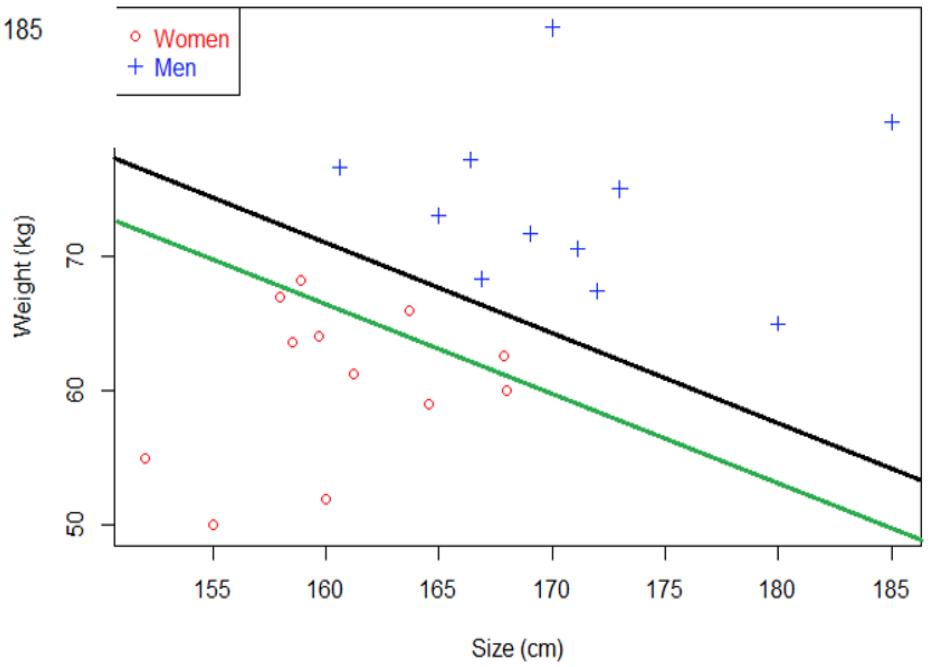




Support Vector Machine:Optimally separating



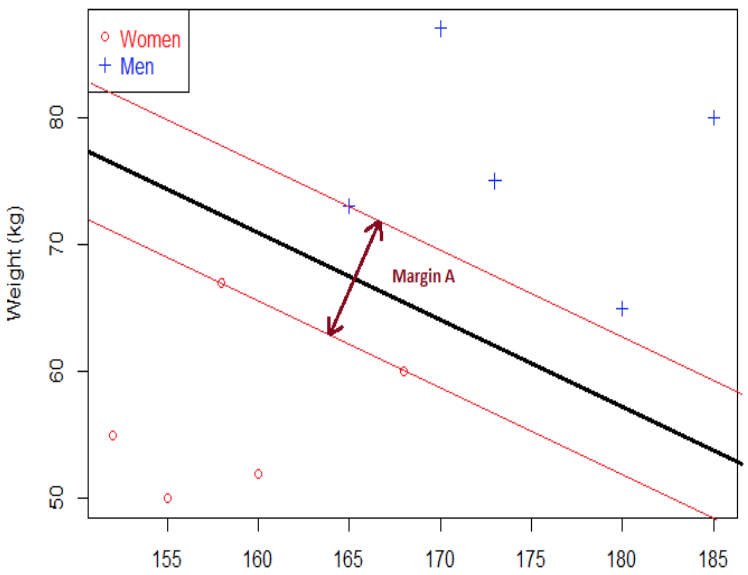
- Try and select a hyperplane as far as possible from the data points in each category

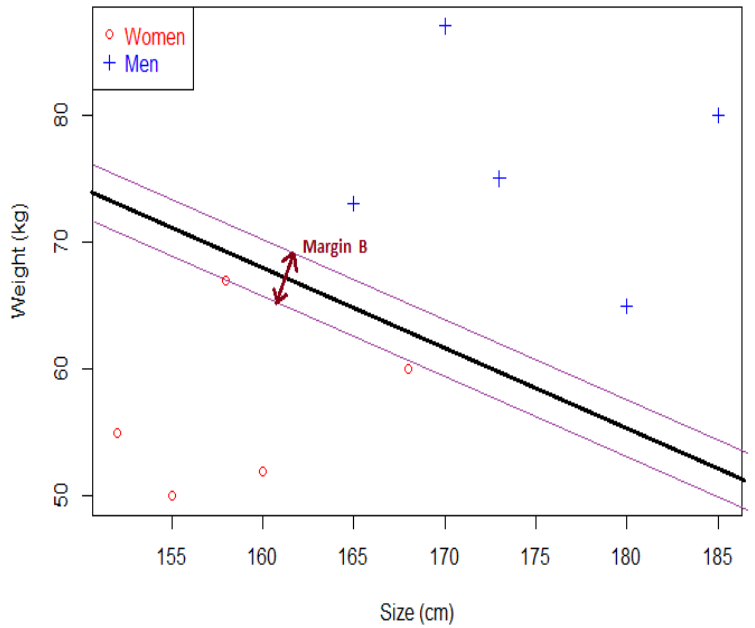


Support Vector Machine:Optimally separating

- That's why the objective of a SVM is to find the optimal separating hyperplane:
  - because it correctly classifies the training data
  - and because it is the one which will generalize better with unseen data

Support Vector Machine:Optimally separating

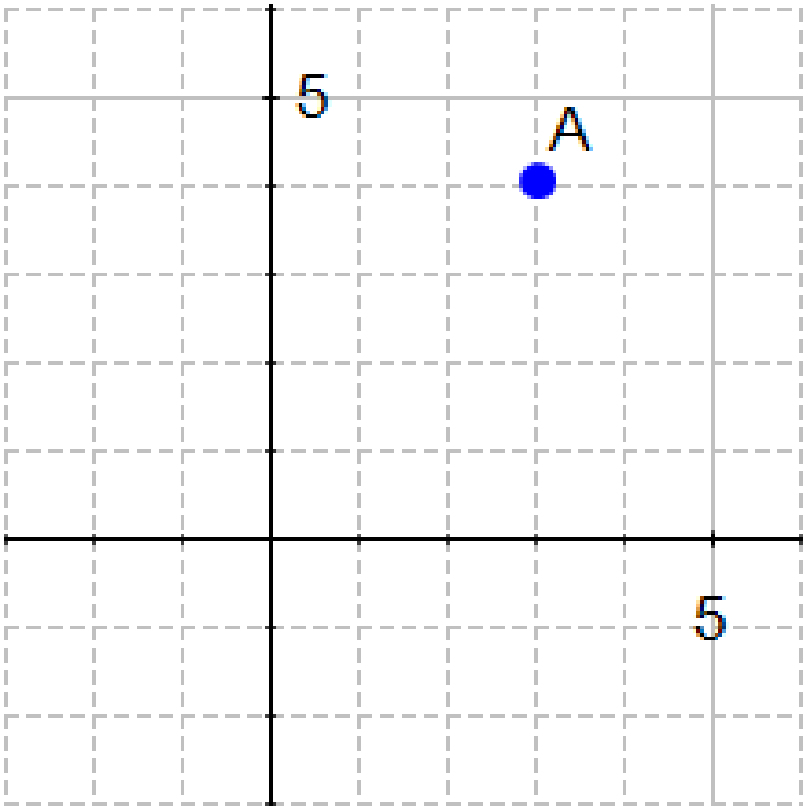


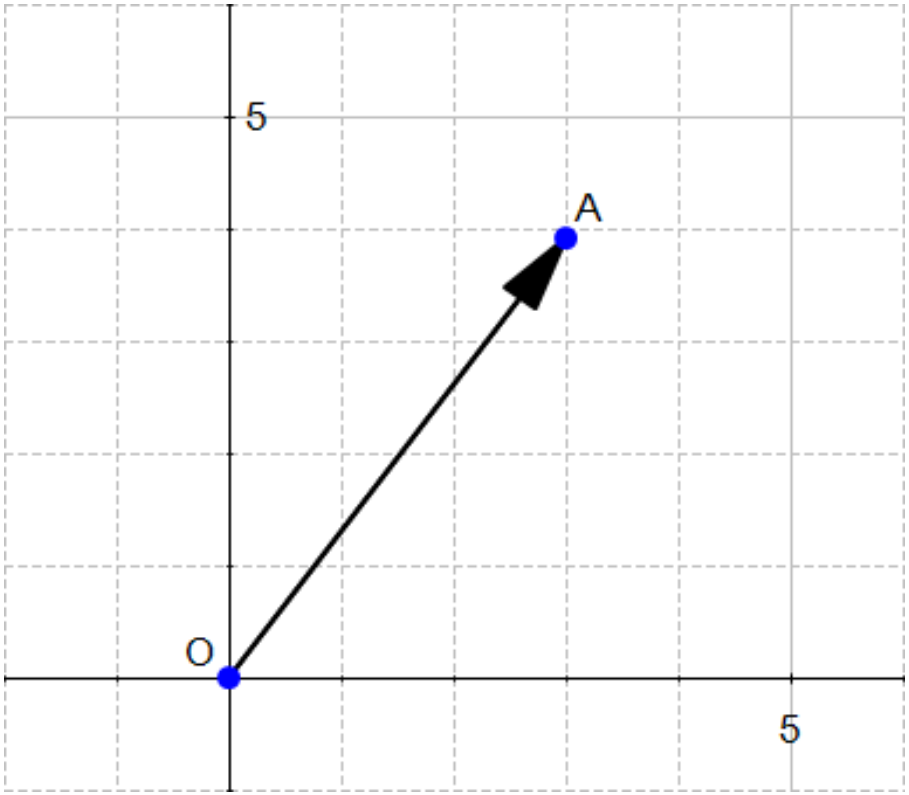


Margin - Given a hyperplane, margin is twice the distance the plane and the nearest data point.

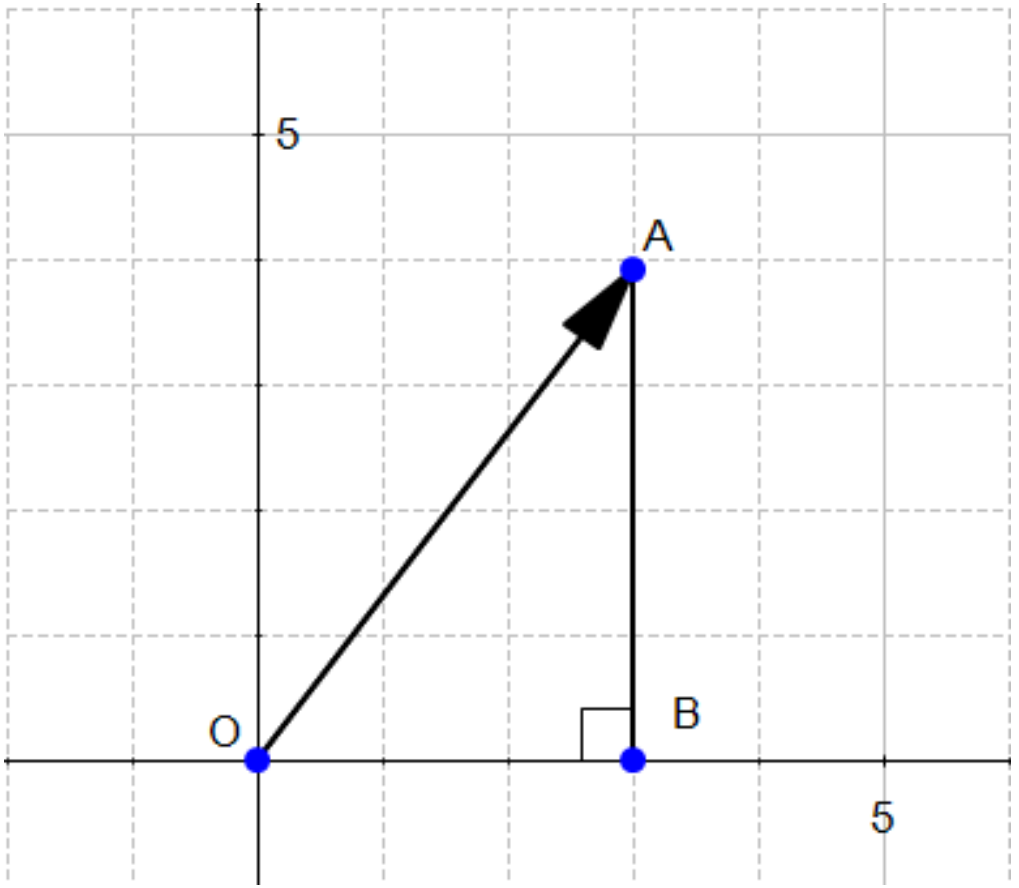
- The further the hyperplane is from the datapoint, bigger is the margin.

Support Vector Machine:Vector





Support Vector Machine:Vector Magnitude/Norm



$$OA^2 = OB^2 + AB^2$$

$$OA^2 = 3^2 + 4^2$$

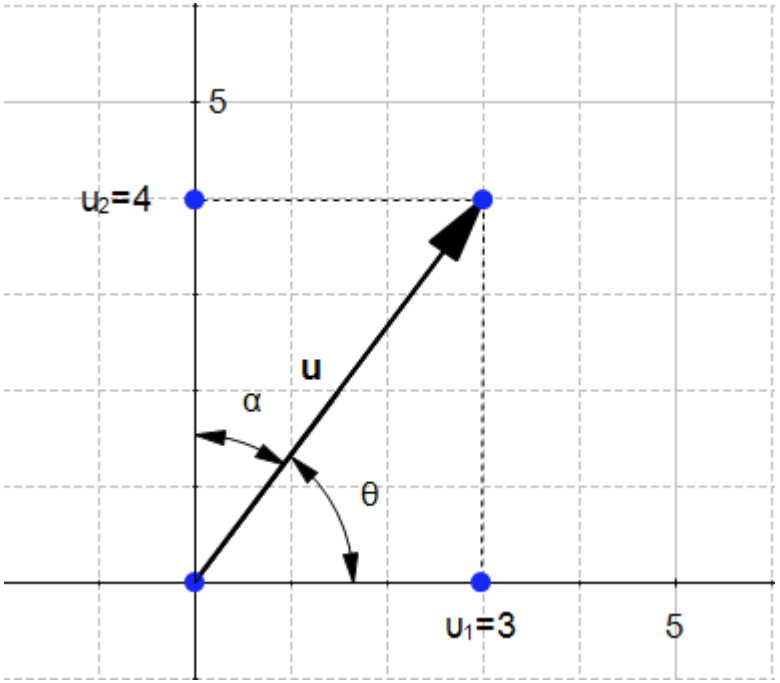
$$OA^2 = 25$$

$$OA = \sqrt{25}$$

$$\|OA\| = OA = 5$$

```python
import numpy as np

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

Support Vector Machine:Vector Direction



$$cos(\theta) = \frac{u_1}{\|u\|} = \frac{3}{5} = 0.6$$

$$cos(\alpha) = \frac{u_2}{\|u\|} = \frac{4}{5} = 0.8$$

*The **direction** of a vector* $\mathbf{u}(u_1, u_2)$ *is the vector* $\mathbf{w}(\frac{u_1}{\|u\|}, \frac{u_2}{\|u\|})$

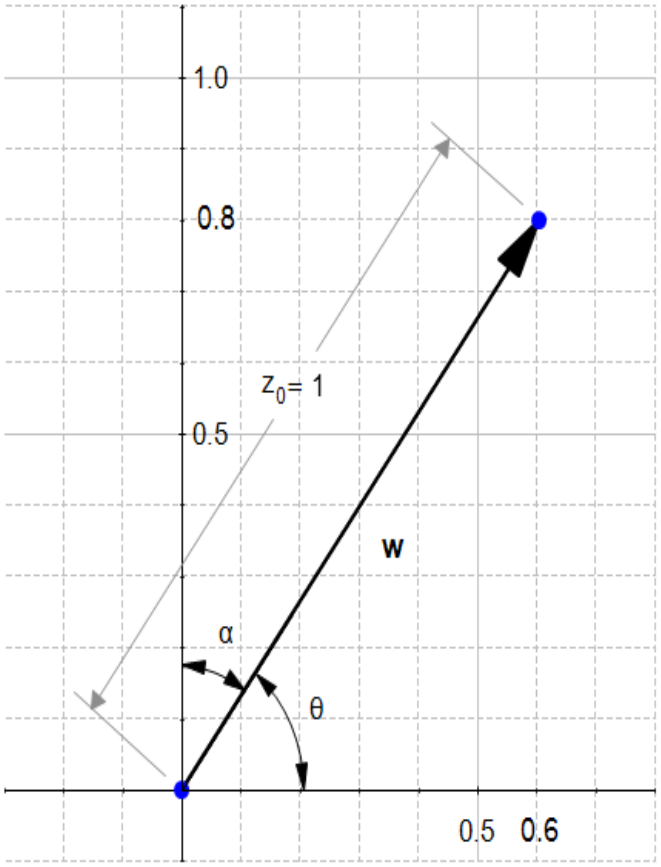
Support Vector Machine:Vector Direction

```python
import numpy as np

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

```python
u = np.array([3,4])
w = direction(u)

print(w) # [0.6 , 0.8]
```

```python
u_1 = np.array([3,4])
u_2 = np.array([30,40])

print(direction(u_1)) # [0.6 , 0.8]
print(direction(u_2)) # [0.6 , 0.8]
```

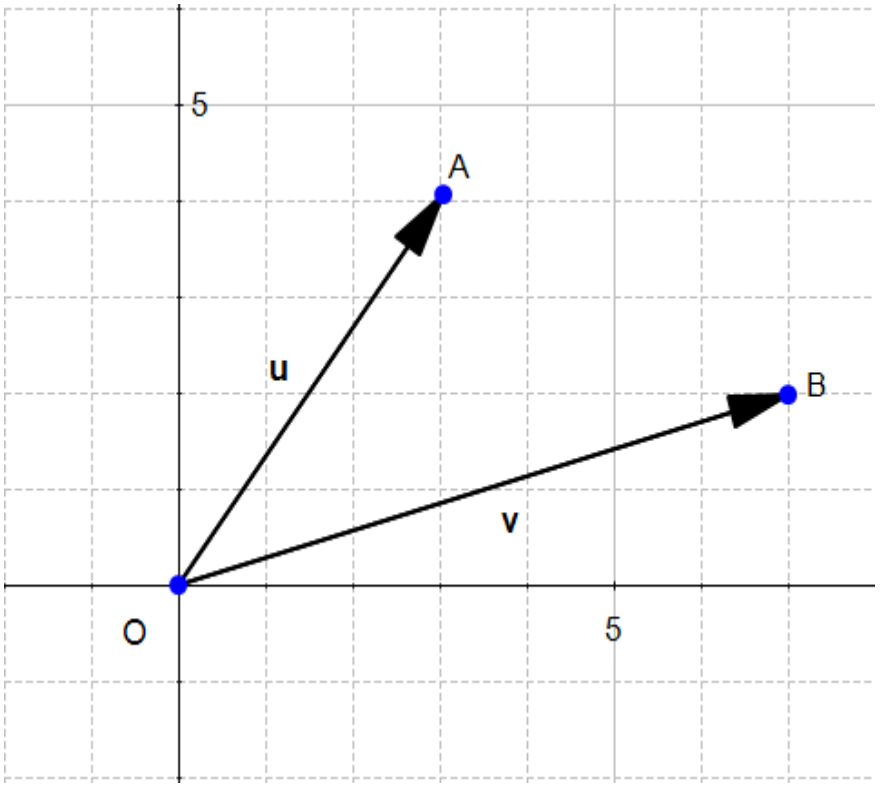
Support Vector Machine:Unit Vector

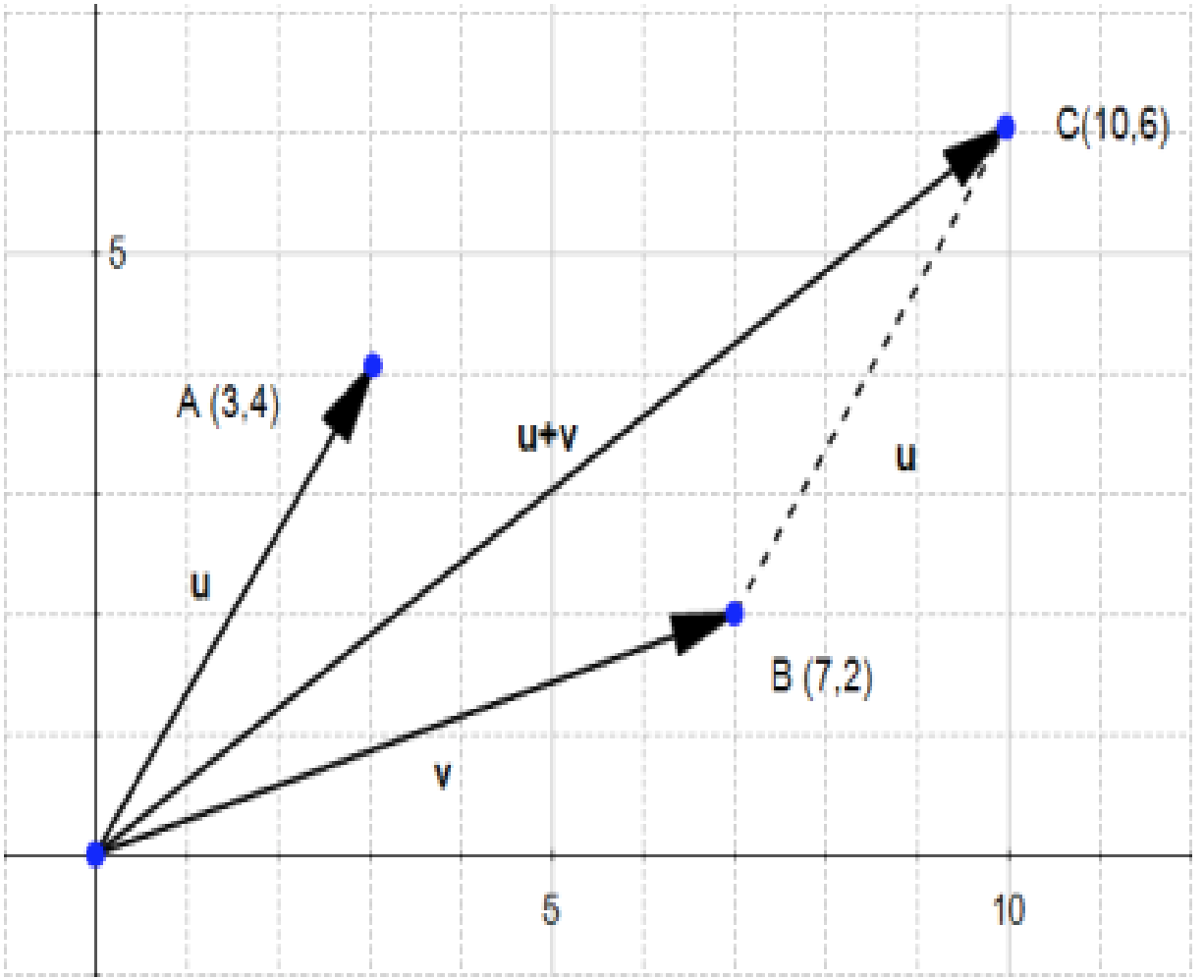



```
np.linalg.norm(np.array([0.6, 0.8])) # 1.0
```

The norm of a unit vector is always 1.

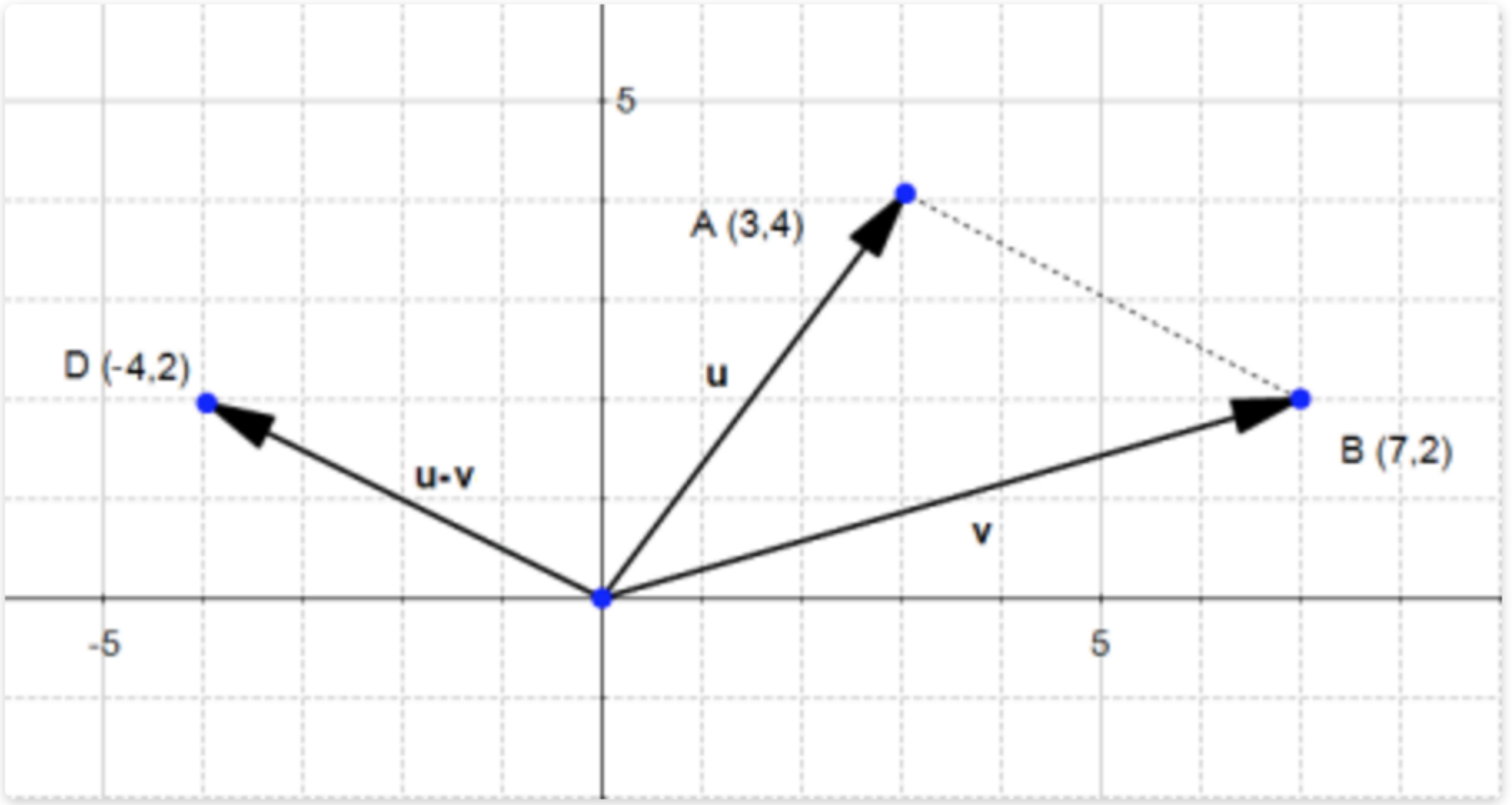
Support Vector Machine:Vector:Sum





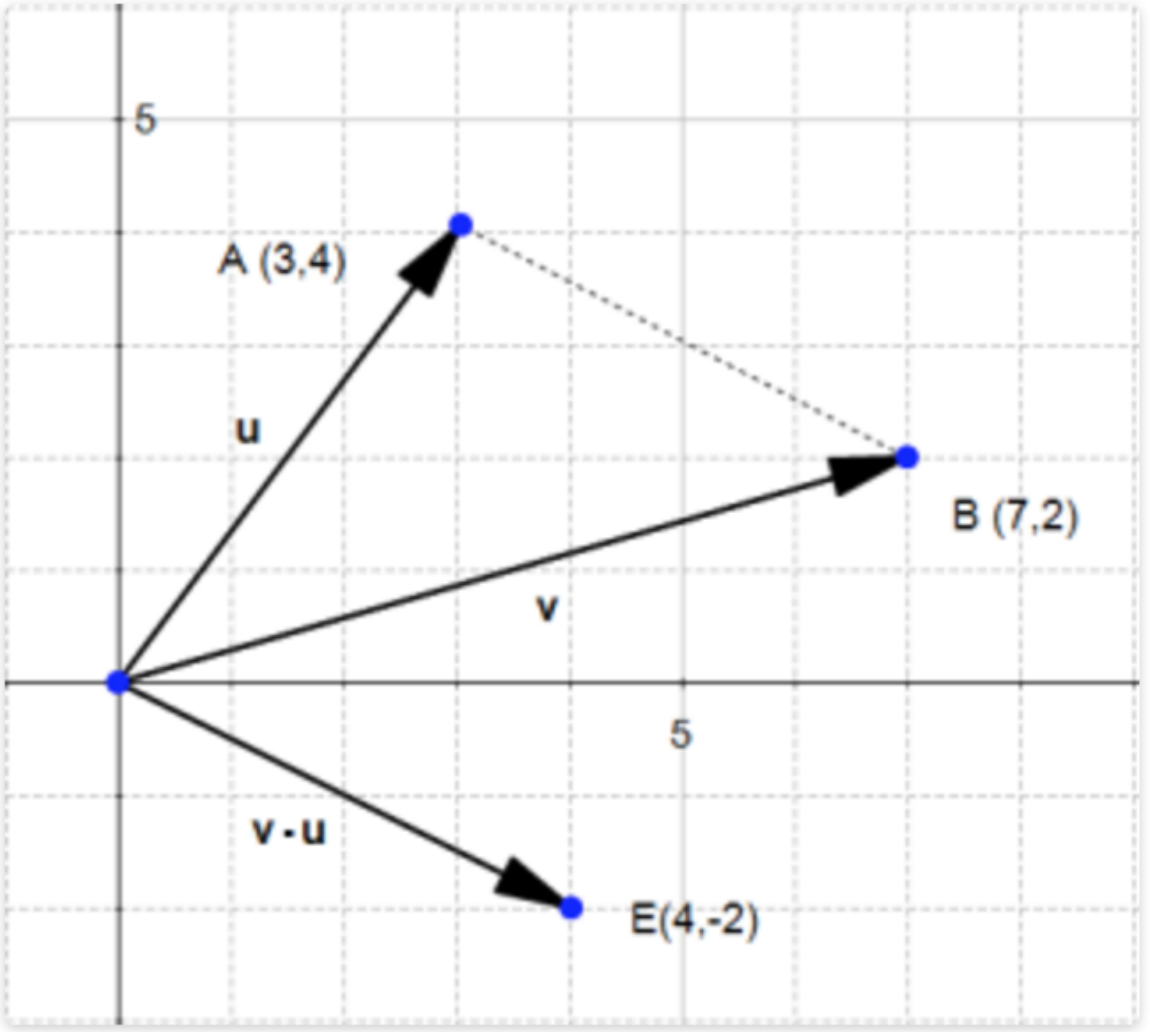
Support Vector Machine:Vector:Difference

$$\mathbf{u} - \mathbf{v} = (u_1 - v_1, u_2 - v_2)$$

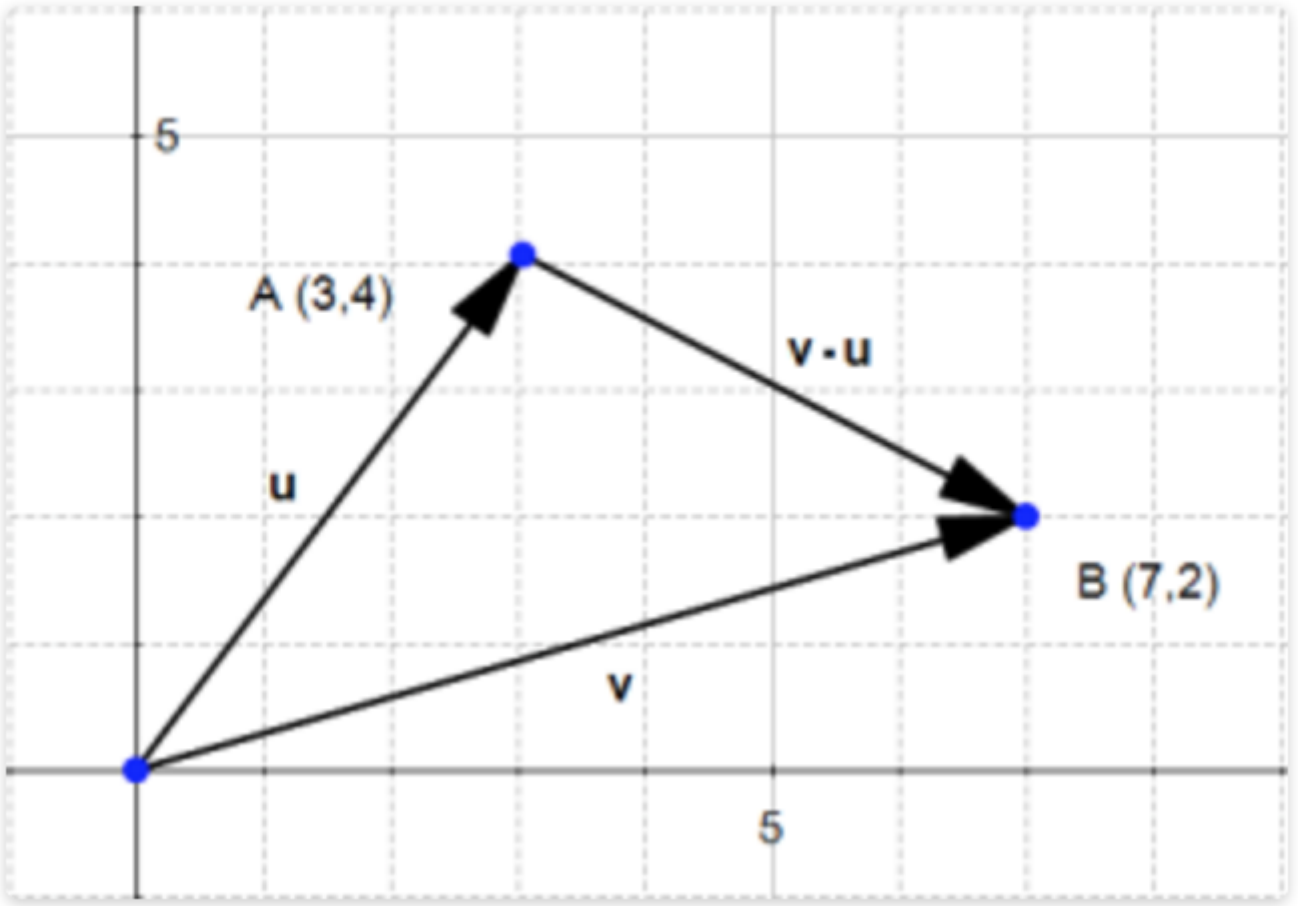


Support Vector Machine:Vector:Difference

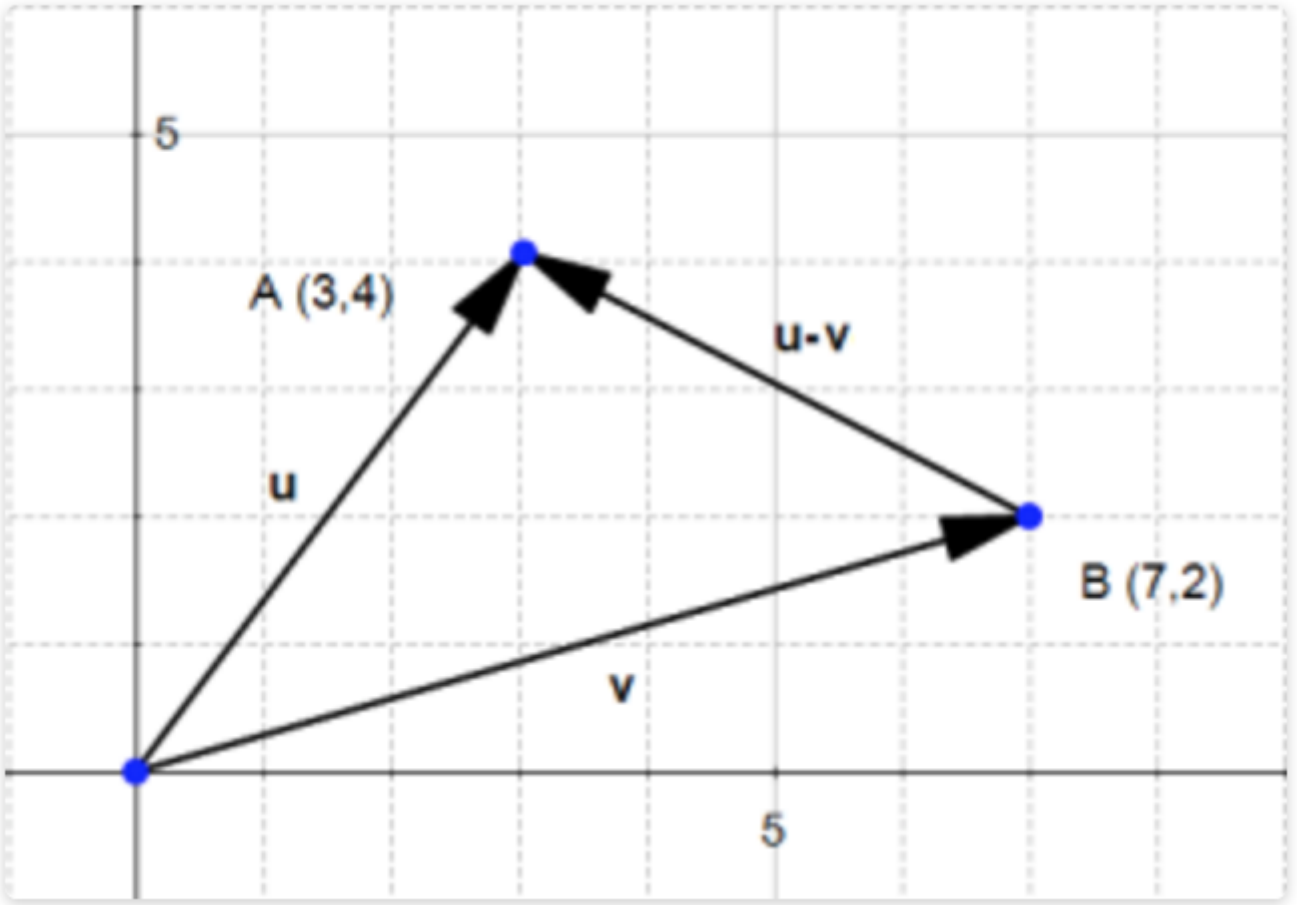
$$\mathbf{v} - \mathbf{u} = (v_1 - u_1, v_2 - u_2)$$



Support Vector Machine:Vector:Difference

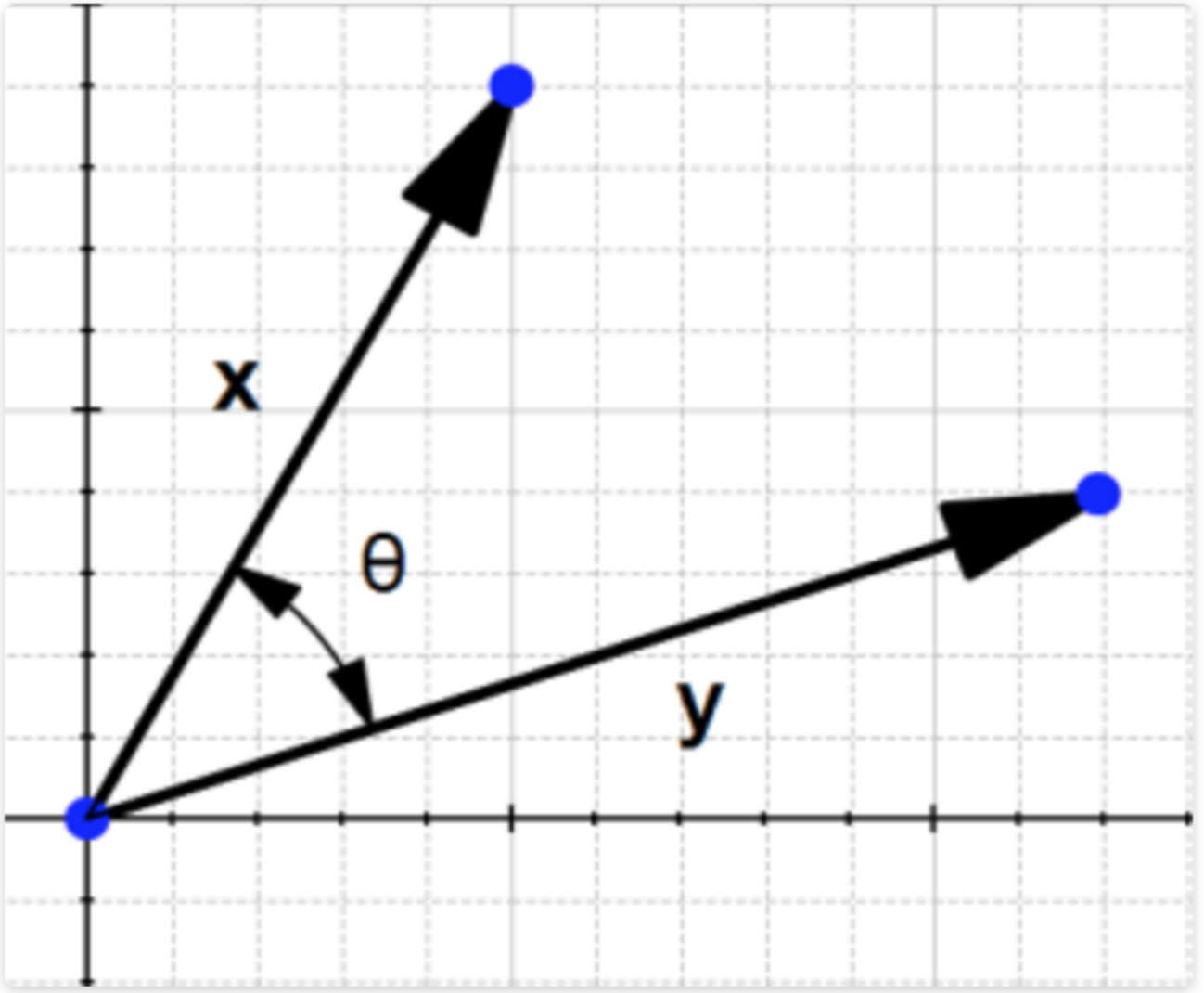


Support Vector Machine:Vector:Difference

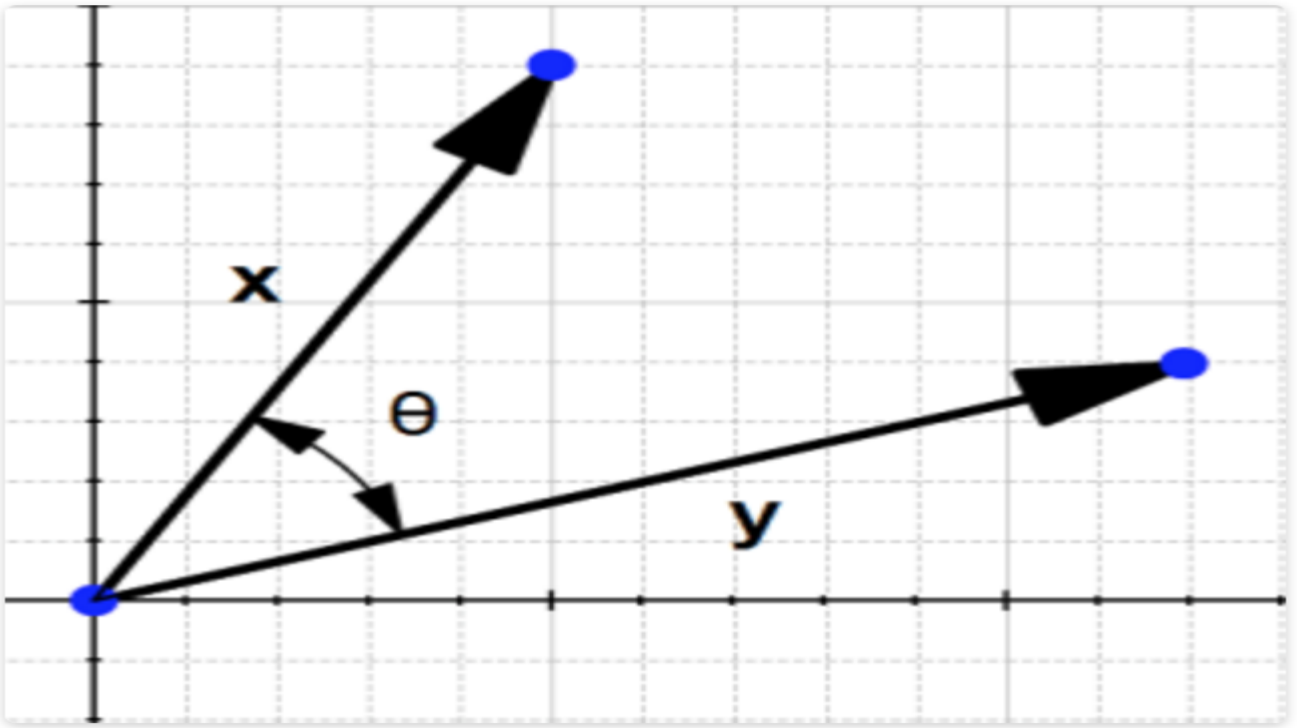


Support Vector Machine:Vector:Dot product

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



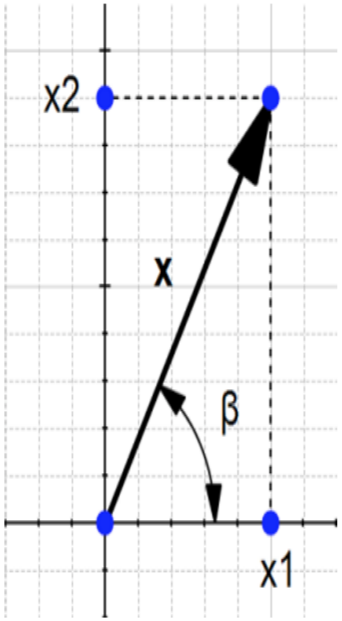
Support Vector Machine:Vector:Dot product

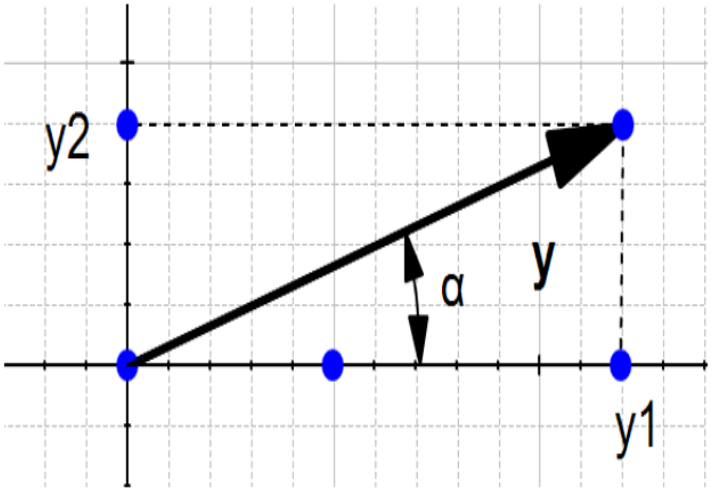


By definition we know that in a right-angled triangle:

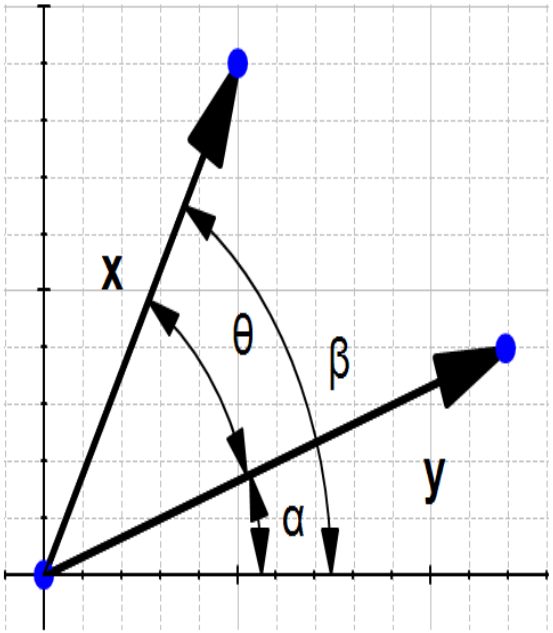
$$cos(\theta) = \frac{adjacent}{hypotenuse}$$

Support Vector Machine:Vector:Dot product





- There is no right angled triangle between x & y.
- However, $\theta = \beta - \alpha$ and computing $\cos(\theta) \Leftrightarrow \cos(\beta - \alpha)$



$$cos(\beta - \alpha) = cos(\beta)cos(\alpha) + sin(\beta)sin(\alpha)$$

Support Vector Machine:Vector:Dot product

$$cos(\beta - \alpha) = cos(\beta)cos(\alpha) + sin(\beta)sin(\alpha)$$

- There are actually 2 different ways of calculating a vector dot product.

$$cos(\beta) = \frac{adjacent}{hypotenuse} = \frac{x_1}{\|x\|}$$

$$sin(\beta) = \frac{opposite}{hypotenuse} = \frac{x_2}{\|x\|}$$

$$cos(\alpha) = \frac{adjacent}{hypotenuse} = \frac{y_1}{\|y\|}$$

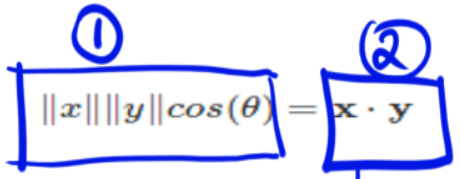
$$sin(\alpha) = \frac{opposite}{hypotenuse} = \frac{y_2}{\|y\|}$$

$$cos(\theta) = cos(\beta - \alpha) = cos(\beta)cos(\alpha) + sin(\beta)sin(\alpha)$$

$$cos(\theta) = \frac{x_1}{\|x\|}\frac{y_1}{\|y\|} + \frac{x_2}{\|x\|}\frac{y_2}{\|y\|}$$

$$cos(\theta) = \frac{x_1 y_1 + x_2 y_2}{\|x\|\|y\|}$$

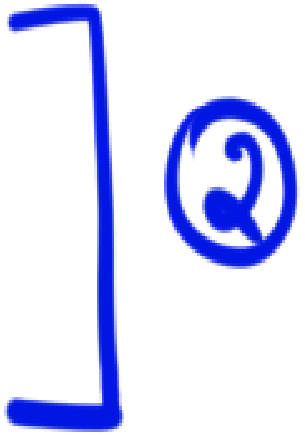
$$\|x\|\|y\|cos(\theta) = x_1 y_1 + x_2 y_2$$



$$\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + x_2 y_2 = \sum_{i=1}^{2}(x_i y_i)$$

Support Vector Machine:Vector:Dot product

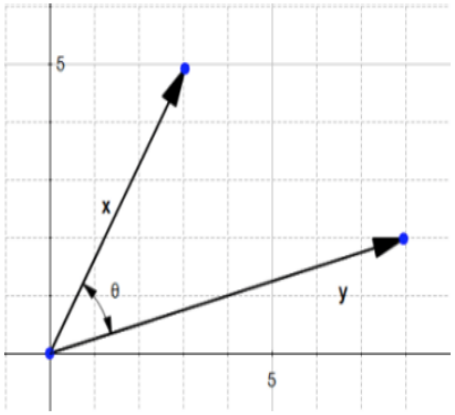
```
def geometric dot product(x,y, theta):
      x norm = np.linalg.norm(x)
      y norm = np.linalg.norm(y)
      return x norm * y norm * math.cos(math.radians(theta))
print("geometric dot product:",geometric dot product(x,y,theta))

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



Support Vector Machine:Vector:Notations

- The dot product is called like that because we write a dot between the two vectors.
  - is the same as talking about the inner product $\langle x,y \rangle$(in linear algebra)
  - scalar product because we take the product of two vectors and it returns a scalar (a real number)

Support Vector Machine:Vector:Orthogonal Projection



$$cos(\theta) = \frac{\|z\|}{\|x\|}$$

$$\|z\| = \|x\| cos(\theta)$$

- By orthogonal projection of x onto y.

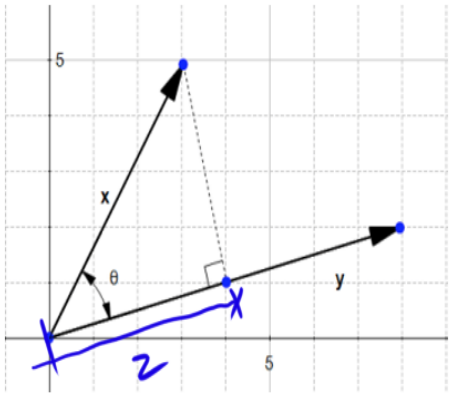
$$cos(\theta) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|x\|\|y\|}$$

- From dot product

$$\|z\| = \|x\| \frac{\mathbf{x} \cdot \mathbf{y}}{\|x\|\|y\|}$$

- replacing cos(θ)

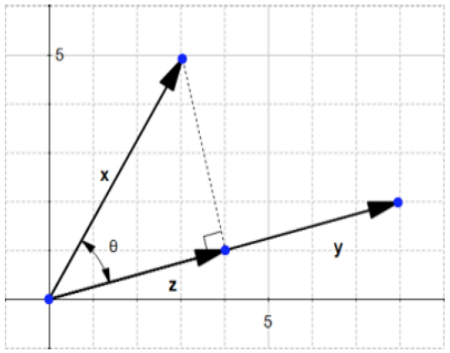
$$\|z\| = \frac{\mathbf{x} \cdot \mathbf{y}}{\|y\|}$$



$$\mathbf{u} = \frac{\mathbf{y}}{\|y\|}$$

u as the direction of y

$$\|z\| = \mathbf{u} \cdot \mathbf{x}$$



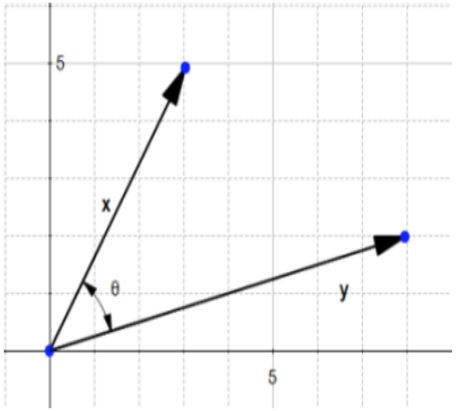
$$\mathbf{u} = \frac{\mathbf{z}}{\|z\|}$$

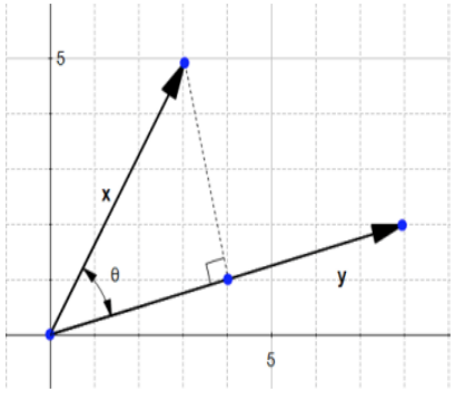
$$\mathbf{z} = \|z\|\mathbf{u}$$

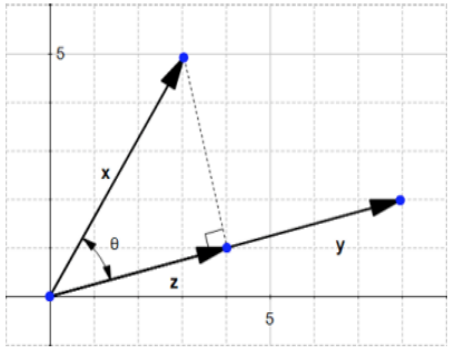
u as the direction vector of y as these 2 directions are the same.

$z = (u \cdot x)\, u$ is the projection of x onto y.

Support Vector Machine:Vector:Orthogonal Projection







$$cos(\theta) = \frac{\|z\|}{\|x\|}$$

$$\|z\| = \|x\| cos(\theta)$$

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

$$\|z\| = \|x\| \frac{\mathbf{x} \cdot \mathbf{y}}{\|x\| \|y\|}$$

$$\|z\| = \frac{\mathbf{x} \cdot \mathbf{y}}{\|y\|}$$

$$\mathbf{u} = \frac{\mathbf{y}}{\|y\|}$$

$$\|z\| = \mathbf{u} \cdot \mathbf{x}$$

$$\mathbf{u} = \frac{\mathbf{z}}{\|z\|}$$

$$\mathbf{z} = \|z\| \mathbf{u}$$

$z = (u \cdot x)\, u$

$= \left(\left[\frac{8}{\sqrt{68}}, \frac{2}{\sqrt{68}} \right] \cdot [3, 5] \right) \times \left[\frac{8}{\sqrt{68}}, \frac{2}{\sqrt{68}} \right]$

$= \left(\frac{8 \times 3}{8.246} + \frac{2 \times 5}{8.246} \right) \times \left[\frac{8}{\sqrt{68}}, \frac{2}{\sqrt{68}} \right]$

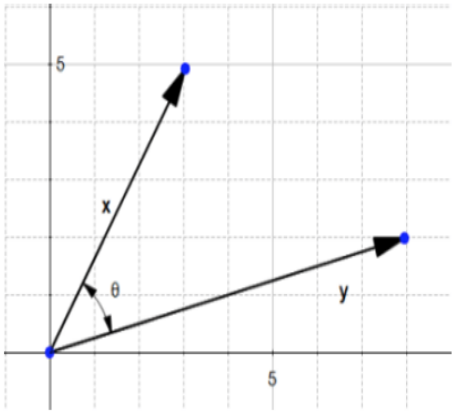
$= (2.910 + 1.212) \times \left[\frac{8}{8.246}, \frac{2}{8.246} \right]$

$= 4.122 \times \left[\frac{8}{8.246}, \frac{2}{8.246} \right]$

$= [4, 1]$

$z = (u \cdot x)\, u$ is the projection of x onto y.

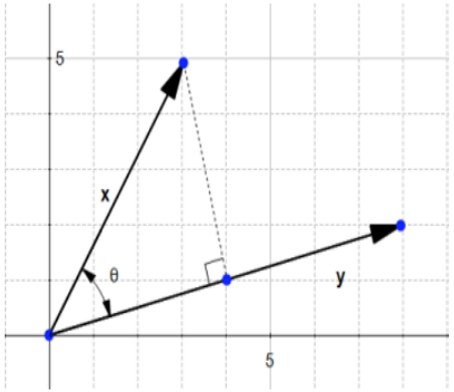
Support Vector Machine:Vector:Orthogonal Projection



$$cos(\theta) = \frac{\|z\|}{\|x\|}$$

$$\|z\| = \|x\| cos(\theta)$$

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

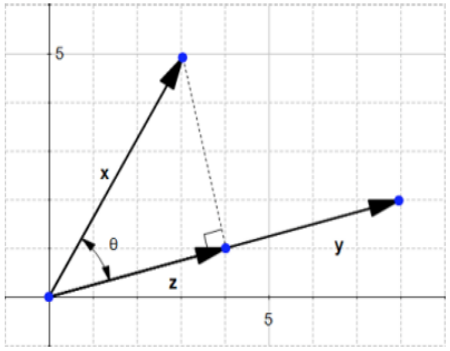


$$\|z\| = \|x\| \frac{\mathbf{x} \cdot \mathbf{y}}{\|x\|\|y\|}$$

$$\|z\| = \frac{\mathbf{x} \cdot \mathbf{y}}{\|y\|}$$

$$\mathbf{u} = \frac{\mathbf{y}}{\|y\|}$$

$$\|z\| = \mathbf{u} \cdot \mathbf{x}$$



$$\mathbf{u} = \frac{\mathbf{z}}{\|z\|}$$

$$\mathbf{z} = \|z\|\mathbf{u}$$

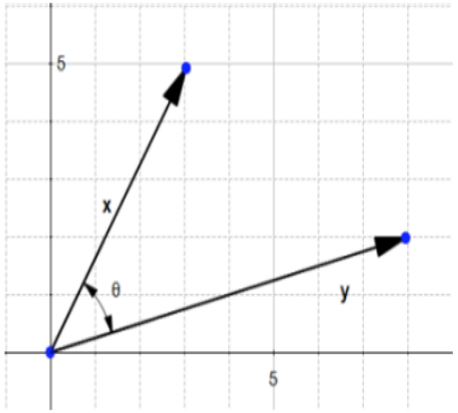
$z = \|z\| u$

$= \sqrt{4^2+1^2} \left(\frac{8}{8.246}, \frac{2}{8.246}\right)$

$= [4, 1]$

$z = (u \cdot x)\, u$ is the projection of x onto y.

Support Vector Machine:Vector:Orthogonal Projection



$$cos(\theta) = \frac{\|z\|}{\|x\|}$$

$$\|z\| = \|x\| cos(\theta)$$

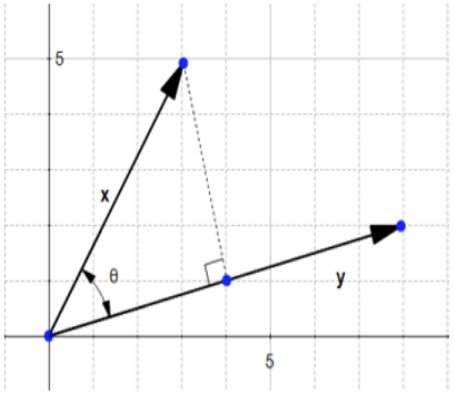
$$cos(\theta) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|x\| \|y\|}$$

$$\|z\| = \|x\| \frac{\mathbf{x} \cdot \mathbf{y}}{\|x\| \|y\|}$$

$$\|z\| = \frac{\mathbf{x} \cdot \mathbf{y}}{\|y\|}$$

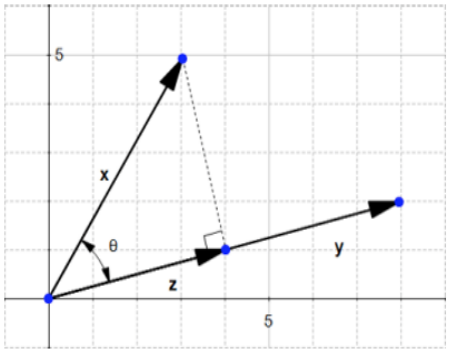
z allows us to compute the distance between x and the line that goes through y.

$$\|x - z\| = \sqrt{(3-4)^2 + (5-1)^2} = \sqrt{17}$$



$$\mathbf{u} = \frac{\mathbf{y}}{\|y\|}$$

$$\|z\| = \mathbf{u} \cdot \mathbf{x}$$



$$\mathbf{u} = \frac{\mathbf{z}}{\|z\|}$$

$$\mathbf{z} = \|z\| \mathbf{u}$$

$z = (u \cdot x)\, u$ is the projection of x onto y.

Support Vector Machine:Vector:Equation of plane

Equation of Line of Plane

$y = mx + b$ $\quad\quad$ $\omega^T x = 0$

$y = mx + b \Leftrightarrow y - mx - b = 0$

given 2 vectors

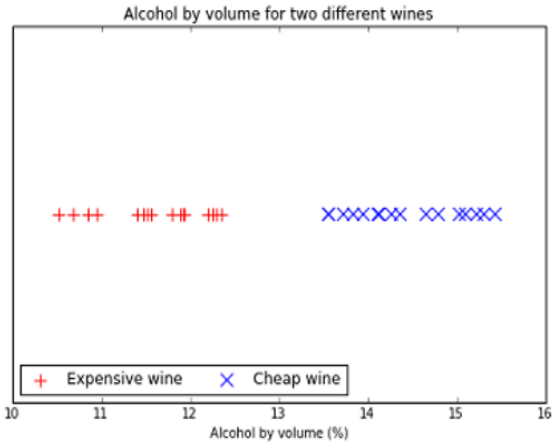
$w \begin{pmatrix} -b \\ -m \\ 1 \end{pmatrix} \quad x \begin{pmatrix} 1 \\ x \\ y \end{pmatrix}$

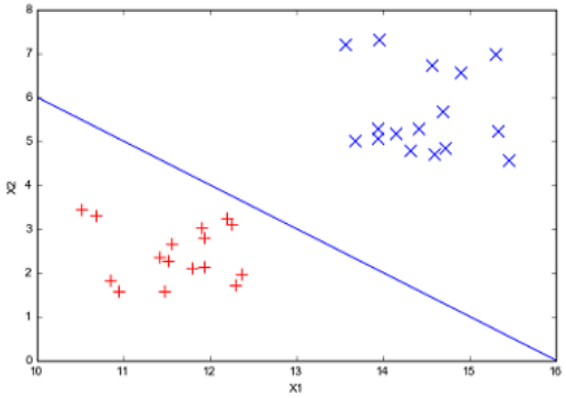
$\omega^T x = -b \times 1 + (-m \times x) + (1 \times y)$

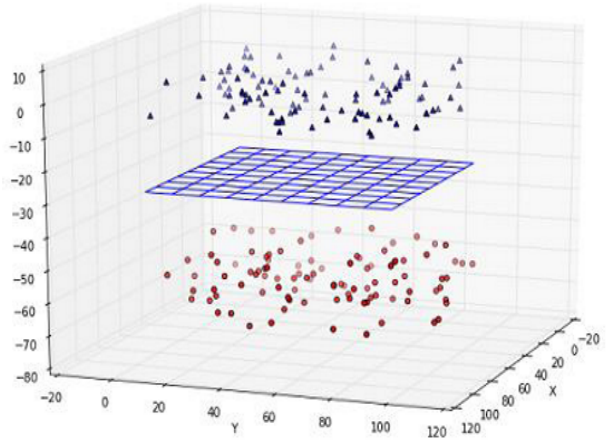
$\omega^T x = y - mx - b$

It is easier to work with more dimensions with $\omega^T x$.

Support Vector Machine:Linear Separability



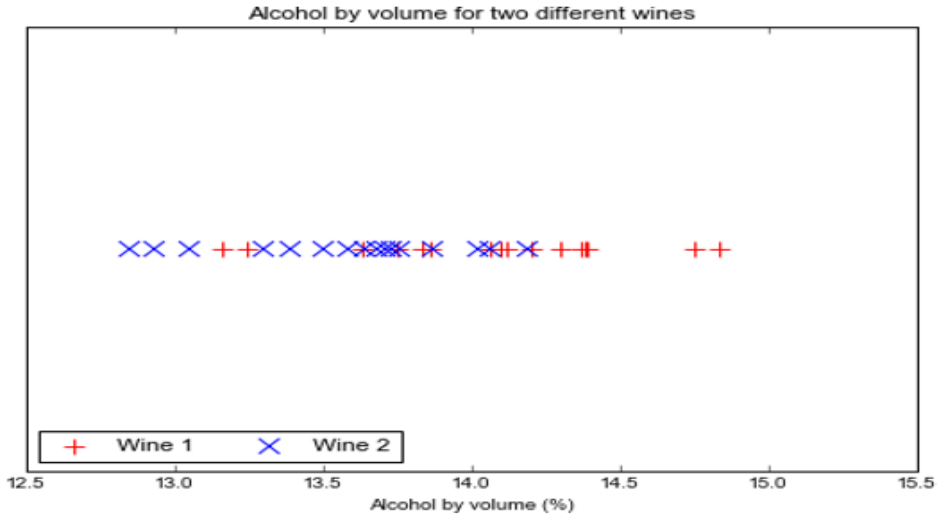


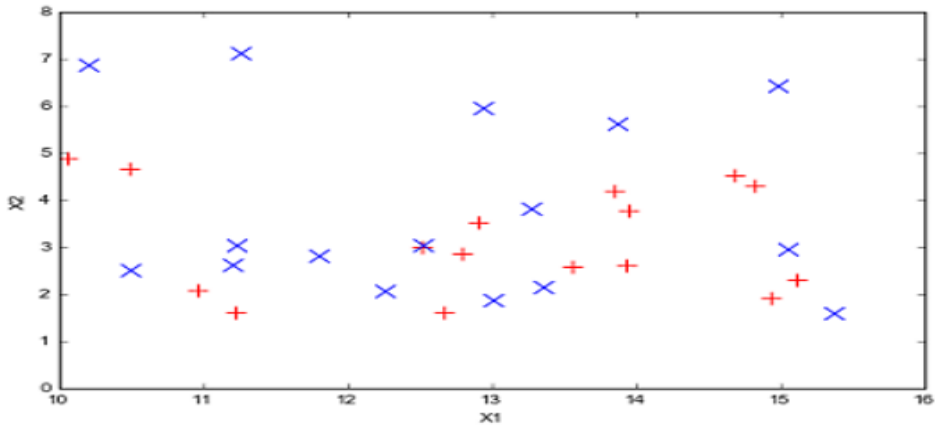


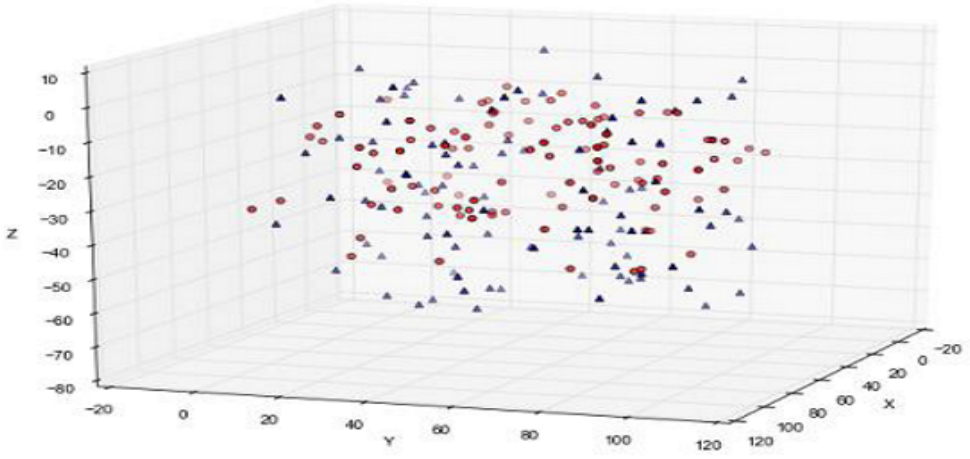
Data is linearly separable when.

1. In one dim, a point seperates the data
2. " 2 ", a line " " "
3. " 3 " " a plane " " "
:

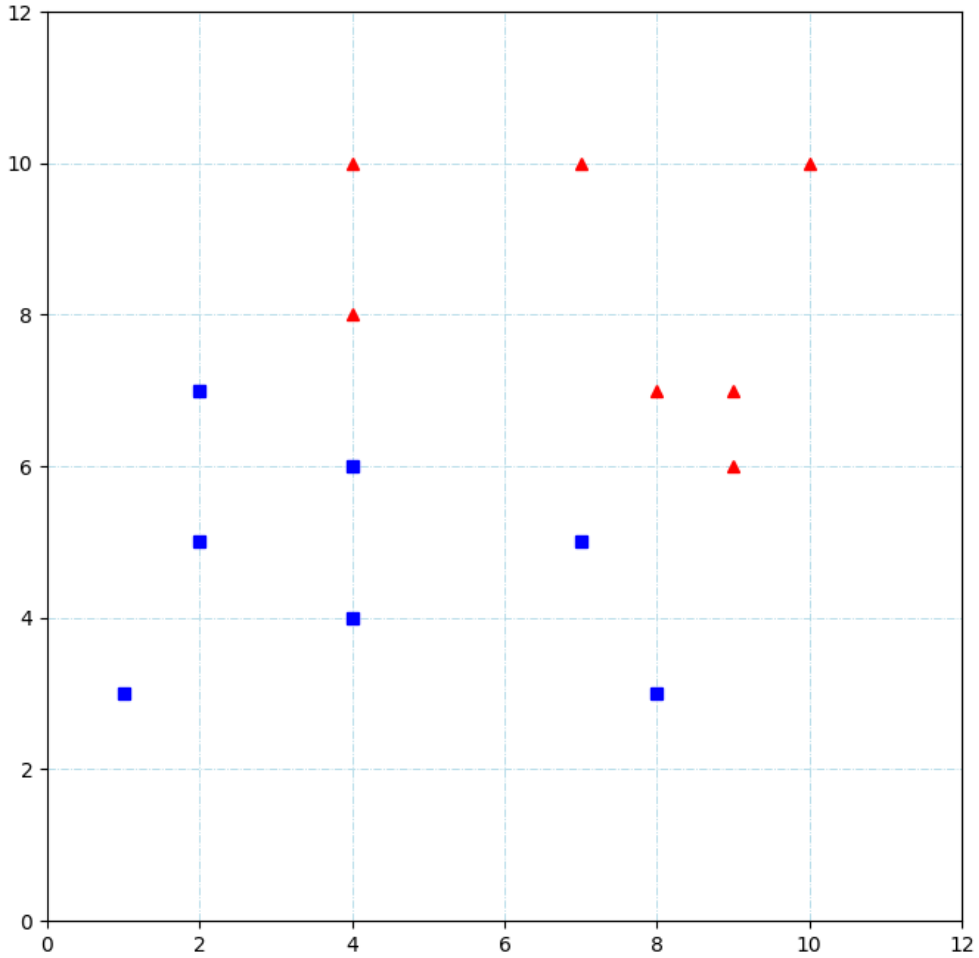
Support Vector Machine:Linear Separability







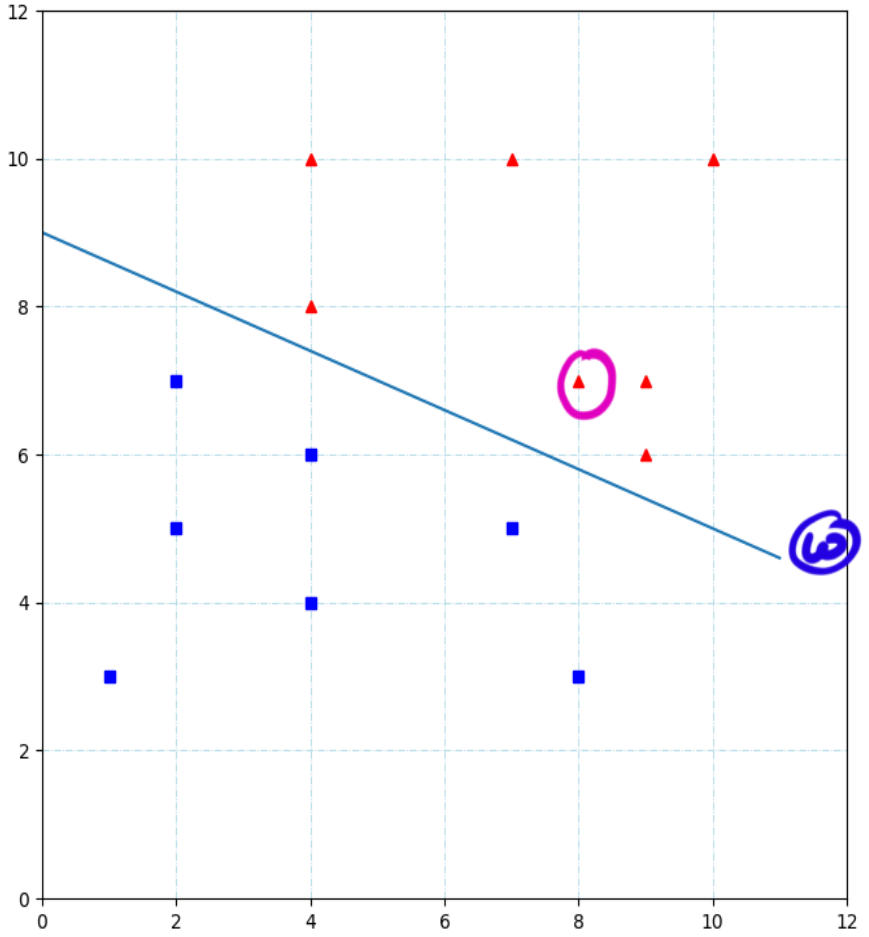
Support Vector Machine:Classifying with Hyperplane




```
def get training examples():
    X1 = np.array([[8, 7], [4, 10], [9, 7], [7, 10],
                   [9, 6], [4, 8], [10, 10]])
    y1 = np.ones(len(X1)) * -1
    X2 = np.array([[2, 7], [8, 3], [7, 5], [4, 4],
                   [4, 6], [1, 3], [2, 5]])
    y2 = np.ones(len(X2))
    return X1, y1, X2, y2
```

Linearly seperable data

Support Vector Machine:Classifying with Hyperplane



$mx + by = y$ is equivalent to

$y - mx - b = 0$

$$w \begin{pmatrix} -(9) \\ -(-.4) \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} -9 \\ .4 \\ 1 \end{pmatrix}$$

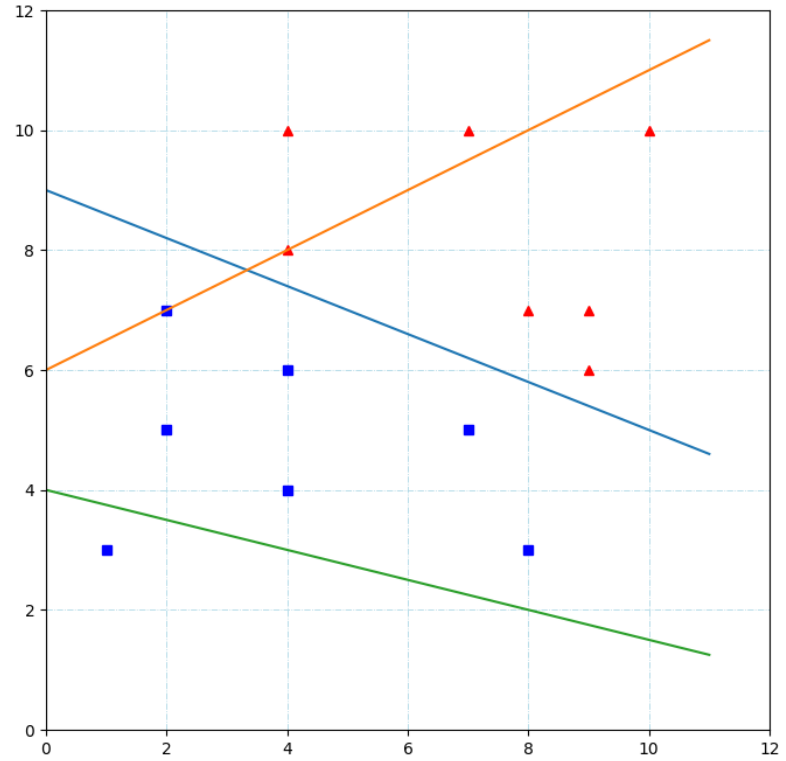
$$x \begin{pmatrix} 1 \\ 8 \\ 7 \end{pmatrix}$$

$$w \cdot x = \begin{pmatrix} -9 \\ .4 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 8 \\ 7 \end{pmatrix} = \begin{pmatrix} -9 \\ 3.2 \\ 7 \end{pmatrix} = 1.2$$

Hypothesis:-

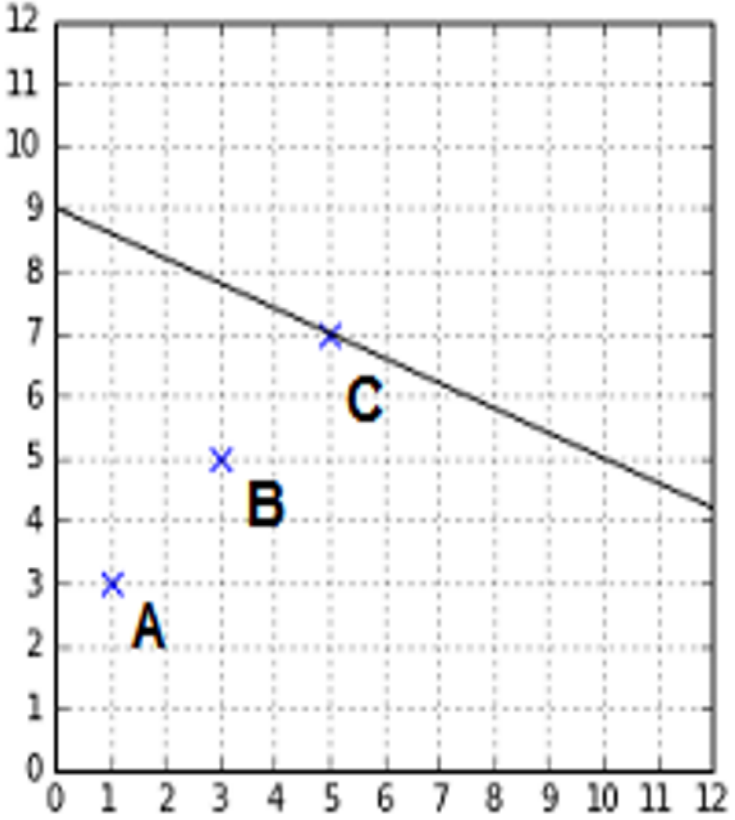
$$h(x_i) = \begin{cases} +1 & \text{if } w \cdot x_i + b \geq 0 \\ -1 & \text{if } w \cdot x_i + b < 0 \end{cases}$$

Support Vector Machine:Classifying with Hyperplane



Different values of w vector will
us different hyperplanes.
SVMs objective is to find the best.

Support Vector Machine:Optimal Hyperplane



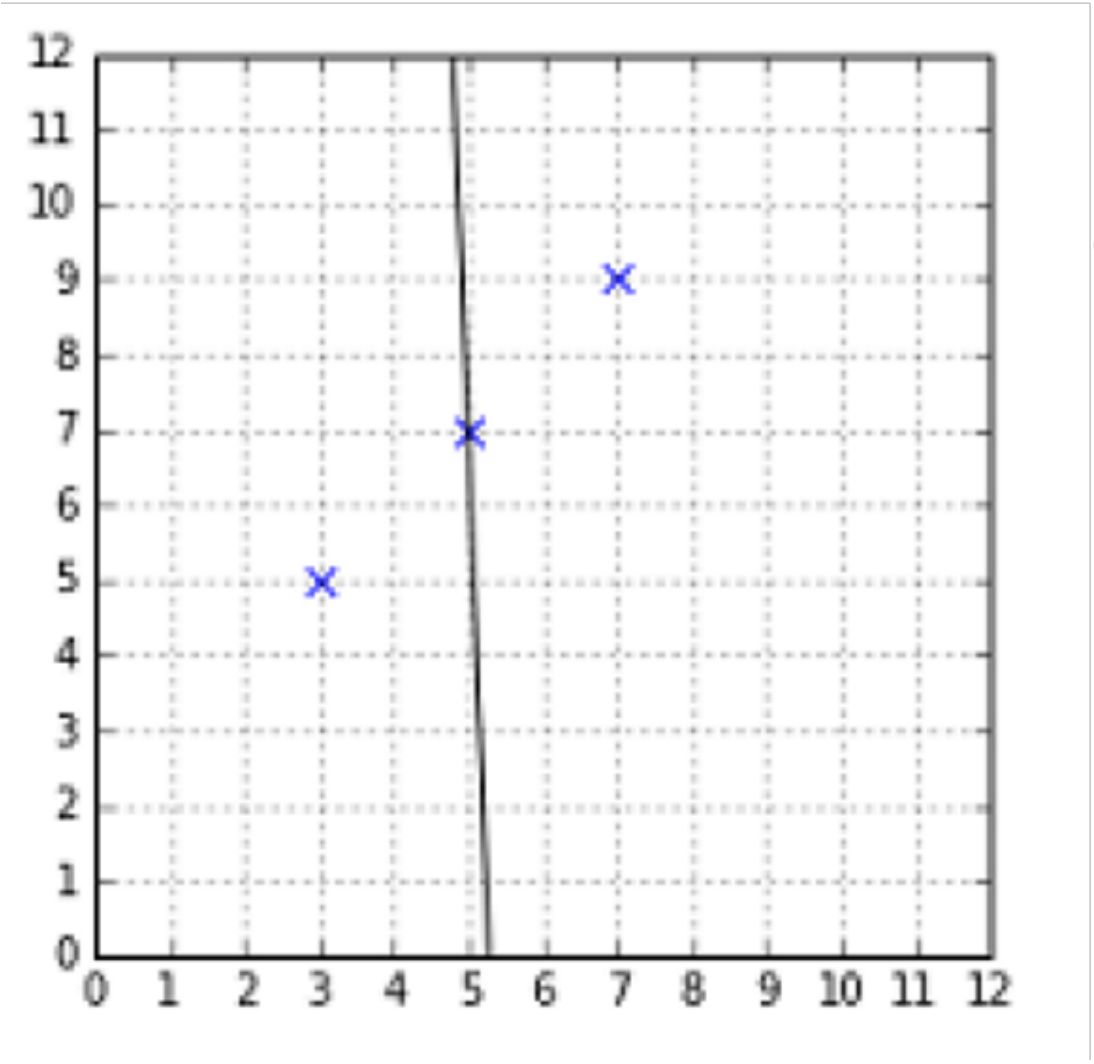
• For away points give us bigger number than the closer points.

for point $A(1,3)$, using vector $\mathbf{a} = (1,3)$ we get $\mathbf{w} \cdot \mathbf{a} + b = 5.6$

for point $B(3,5)$, using vector $\mathbf{b} = (3,5)$ we get $\mathbf{w} \cdot \mathbf{b} + b = 2.8$

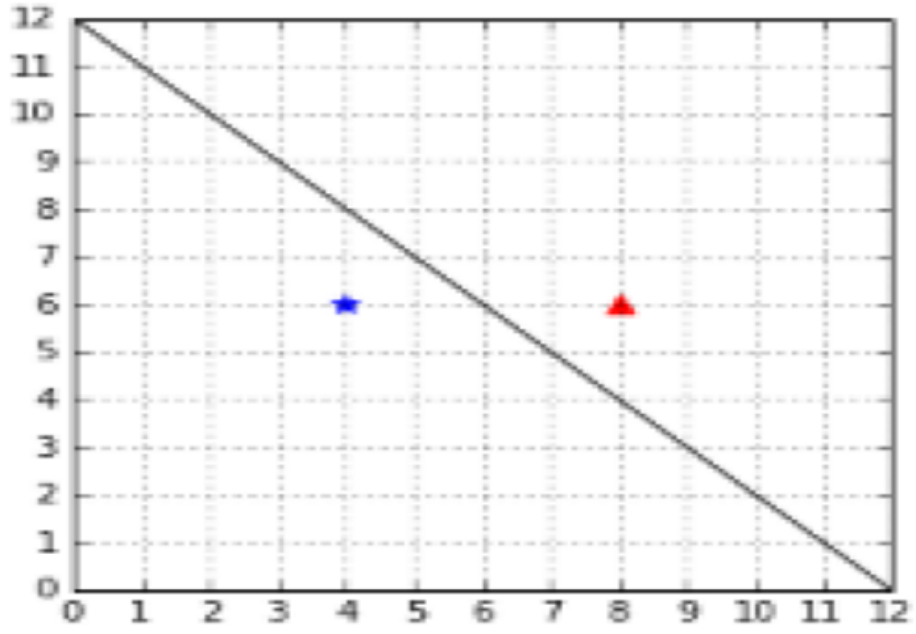
for point $C(5,7)$, using vector $\mathbf{c} = (5,7)$ we get $\mathbf{w} \cdot \mathbf{c} + b = 0$

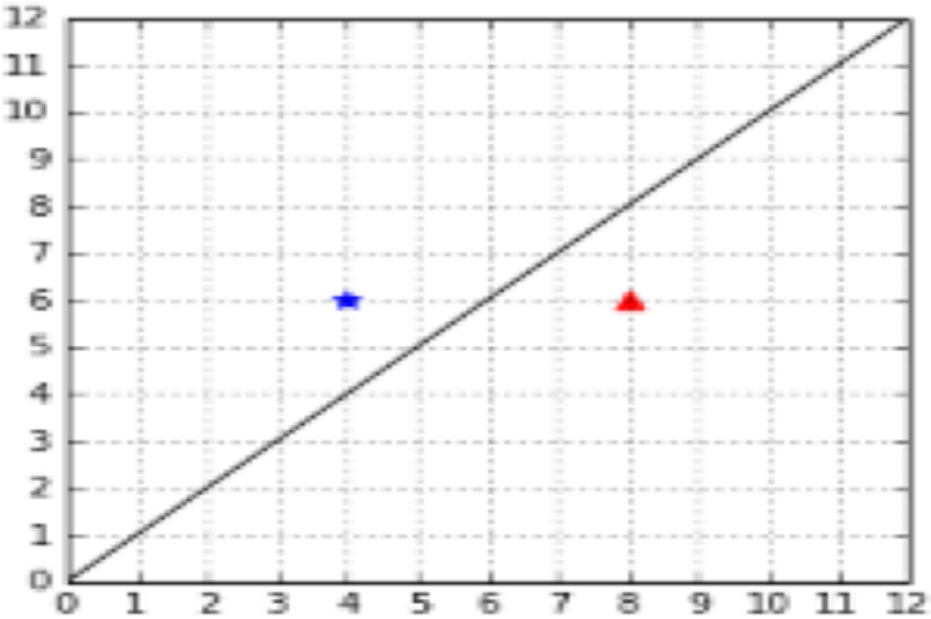
Support Vector Machine:Optimal Hyperplane



- Points ought not visually above or below.

Support Vector Machine:Optimal Hyperplane:functional margin





- both these hyperplanes return the same value but only one is correct.
- the formula can be adjusted by multiplying the label with β

$\beta = \omega \cdot x + b$

$\delta = y \times \beta$

where y is the label.

Support Vector Machine:Optimal Hyperplane:functional margin

$$f = y \times \beta$$

$$f = y(\mathbf{w} \cdot \mathbf{x} + b)$$

- **Positive if the point is correctly classified**
- **Negative if the point is incorrectly classified**
- **Beginning of a solution that can compare hyperplanes**

Support Vector Machine:Optimal Hyperplane:functional margin

$$\bar{F} = \min_{i=1...m} f_i$$

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

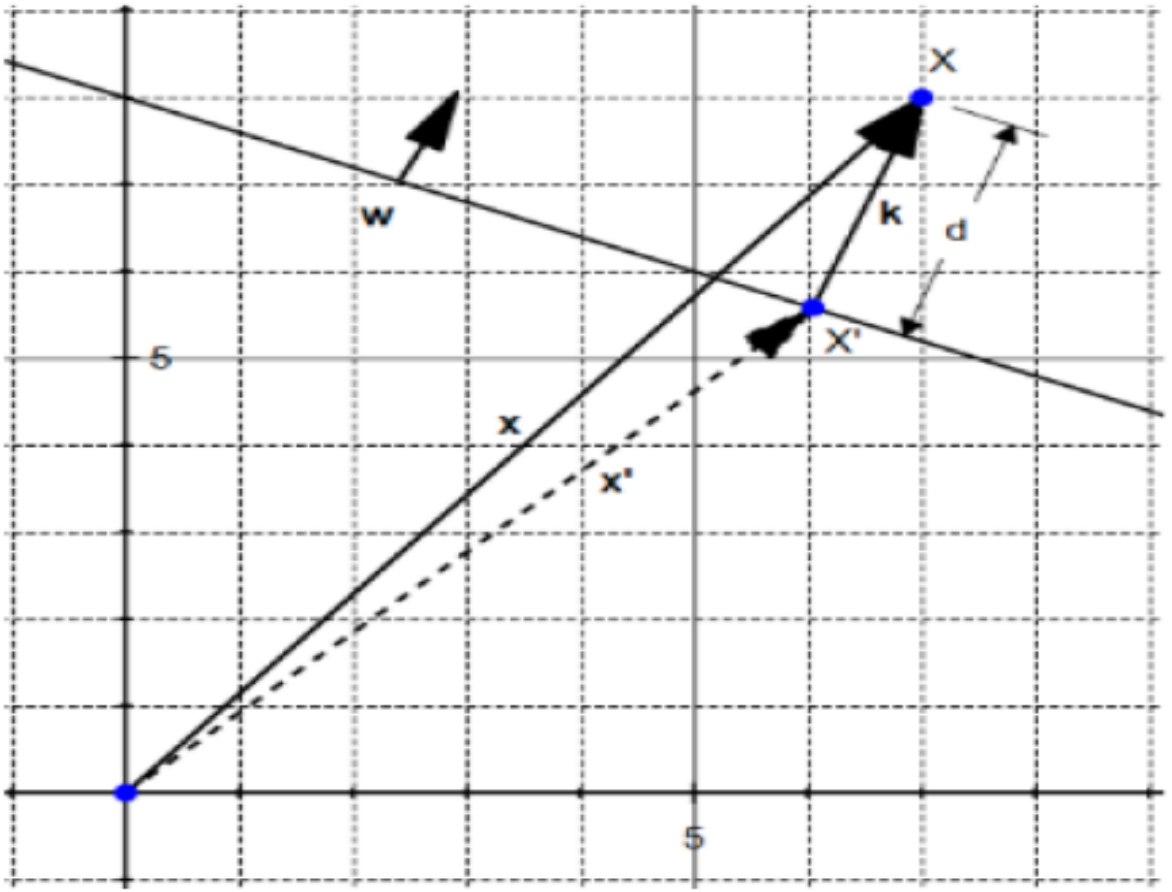
- **With this formula, when comparing two hyperplanes, we will still select the one for which is the largest.**
- **The added bonus is that in special cases like the ones in slide-36, we will always pick the hyperplane that classifies correctly**

Support Vector Machine:Optimal Hyperplane: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):
    print("x:",x)
    result = y * (np.dot(w, x) + b)
    print("result:",result)
    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)])


x = np.array([3,5])
y = 1
b 1 = 9
w 1 = np.array([-.4, -1])
w 2 = w 1*10
b 2 = b 1*10
print(example functional margin(w 1, b 1, x, y)) # 2.8
print(example functional margin(w 2, b 2, x, y)) # 28



x1,y1,x2,y2=get training examples()
```

- for slide 36 it will give +2 for the first one and -12 for the second one.
- We pick the first one because it has a bigger margin.
- The problem is that it is scale invariant.

Support Vector Machine:Optimal Hyperplane:geometric margin



$$\mathbf{k} = d\frac{\mathbf{w}}{\|w\|}.$$

$$\mathbf{x}' = \mathbf{x} - \mathbf{k}$$

$$\mathbf{x}' = \mathbf{x} - d\frac{\mathbf{w}}{\|w\|}$$

$$\mathbf{w} \cdot \mathbf{x}' + b = 0$$

$$\mathbf{w} \cdot \left(\mathbf{x} - d\frac{\mathbf{w}}{\|w\|}\right) + b = 0$$

$$\mathbf{w} \cdot \mathbf{x} - d\frac{\mathbf{w} \cdot \mathbf{w}}{\|w\|} + b = 0$$

$$\mathbf{w} \cdot \mathbf{x} - d\frac{\|w\|^2}{\|w\|} + b = 0$$

$$\mathbf{w} \cdot \mathbf{x} - d\|w\| + b = 0$$

$$d = \frac{\mathbf{w} \cdot \mathbf{x} + b}{\|w\|}$$

$$d = \frac{\mathbf{w}}{\|w\|} \cdot \mathbf{x} + \frac{b}{\|w\|}$$

geometric margin

Support Vector Machine:Optimal Hyperplane:geometric margin

```python
# Compute the functional 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)
    print("x:",x)
    result = y * (np.dot(w/norm, x) + b/norm)
    print("result:",result)
    return result

# Compute the functional 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)])


x = np.array([3,5])
y = 1
b 1 = 9
w 1 = np.array([-.4, -1])
w 2 = w 1*10
b 2 = b 1*10
print(example geometric margin(w 1, b 1, x, y)) # 2.8
print(example geometric margin(w 2, b 2, x, y)) # 2.8
```

$$f = y(\mathbf{w} \cdot \mathbf{x}) + b$$

- we devide w by its norm

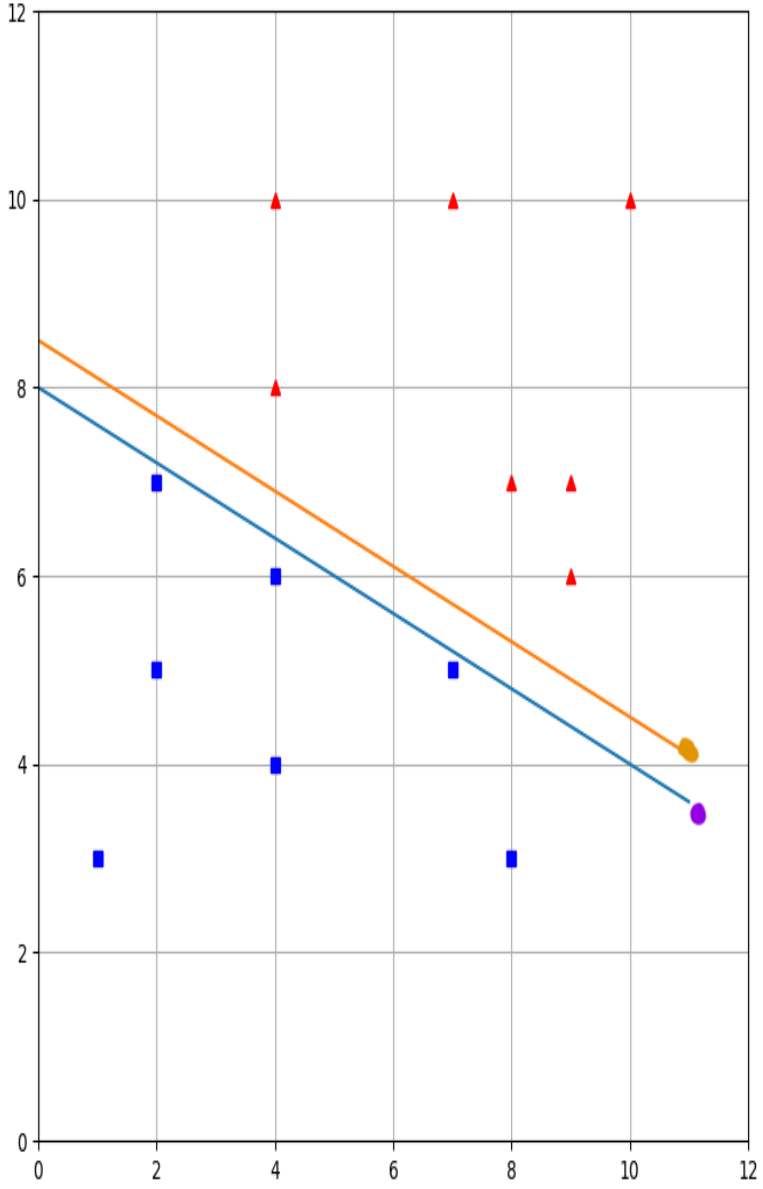
$$\gamma = y\left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot \mathbf{x} + \frac{b}{\|\mathbf{w}\|}\right)$$

- also devide b by the norm

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

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

Support Vector Machine:Optimal Hyperplane:geometric margin




```python
# Compute the functional 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 functional 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)])


# Compare two hyperplanes using the geometrical margin.

x1,y1,x2,y2=get training examples()
x=np.vstack((x1,x2))
y=np.hstack((y1,y2))
b 1 = 8
b 2 = 8.5
w 1 = np.array([-.4, -1])

print(geometric margin(w 1, b 1, x, y)) # 0.185695338177
print(geometric margin(w 1, b 2, x, y)) # 0.64993368362
```

- Now it becomes an optimization problem.

- It is apparent that the blue vector has a better geometric margin