Perceptron's and SVM

Rithvik Reddy Ananth

24th October 2020

—

Machine Learning

—

Cowan Charles

Perceptron's

As usual, we want to generate a data set to fit a perceptron onto. Recall that in the analysis of perceptrons in the notes, we assumed that all the data was contained within the unit sphere of whatever the underlying dimension was. Consider generating data points *on* the unit sphere in the following way: let $\underline{Z}$ be a k-dimensional vector, where each entry is an i.i.d. standard normal, mean 0 and variance 1. Then define $\underline{X} = \underline{Z}/||\underline{Z}||$ (taking the norm as the 2-norm or Euclidean norm), so that $\underline{X}$ is a random vector of length 1, lying exactly on the k-dimensional unit sphere. Note that because of the spherical symmetry of the $\underline{Z}$, the $\underline{X}$ will be uniformly distributed over the surface of the sphere.

For a given value of $1 > \epsilon > 0$, discard any points where $|X_k| < \epsilon$ - this will create two hemispheres of data points separated by a gap along the equator (in this high dimensional space). Let us classify any remaining point where $X_k \geq \epsilon$ with $Y = +1$, and any remaining point with $X_k \leq -\epsilon$ with $Y = -1$. Note that for a data set defined in this way, we have a very natural perceptron that can be applied:

$$\text{classify}(\underline{x}) = \begin{cases} +1 & \text{if } 0x_1 + 0x_2 + \ldots + 0x_{k-1} + 1x_k > 0 \\ -1 & \text{if } 0x_1 + 0x_2 + \ldots + 0x_{k-1} + 1x_k < 0, \end{cases} \tag{1}$$

with a linear separator given by $x_k = 0$. This is by no means the only feasible perceptron, but it does show that the data is linearly separable.

Additionally, note that the margin of separation between the positive and negative data classes is *at least* ϵ.

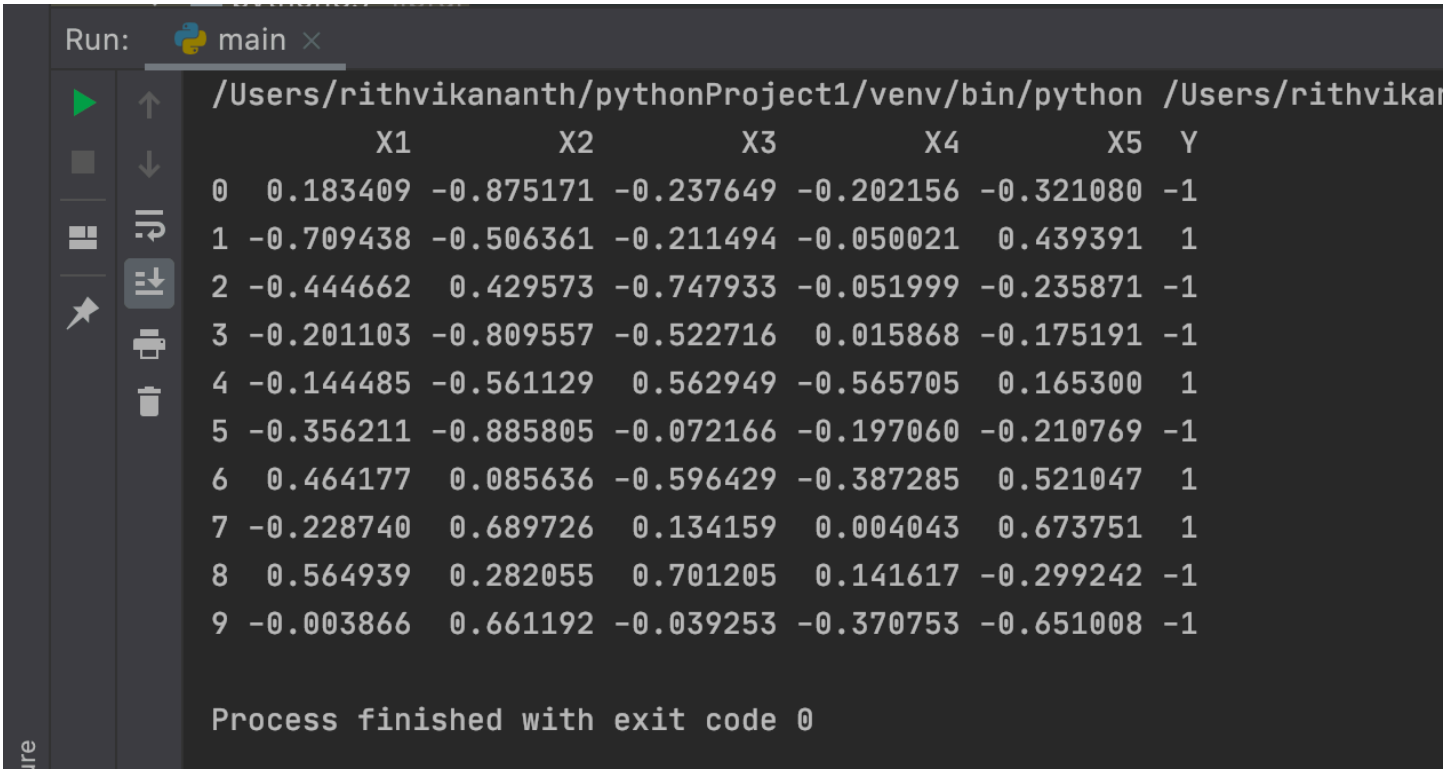
We proved in class that the convergence of the Perceptron Learning Algorithm is bound independently of the dimension of the data and the number of data points, and can be bound entirely in terms of the maximum margin between the two data classes. I'd like you to try to verify this experimentally. For a given value of k and ϵ, and a desired value of m, consider repeatedly generating points $\underline{x}$ but discarding them if they do not satisfy $|x_k| \geq \epsilon$ - repeat this process until m data points have been generated.

We generated the dataset for perceptron as per the given conditions so that we would surely have a linear separator for the generated dataset.

```
import numpy as np
import pandas as pd
from pprint import pprint
import math

def generate_data(k, m, eps):
    count = 0
    data = []
    # mean = 0 standard deviation = 1 of a normal distribution
    while(count != m):
        z = np.random.normal(0, 1, k)
        a = np.sum(np.square(z))
        magnitute = math.sqrt(a)
        X = z / magnitute
        if abs(X[k-1]) < eps:
            continue
        count += 1
        d = X.tolist()
        data.append(d)
    # generate the Y values based on the last feature
```

```
    y = []
    for row in range(m):
        if data[row][-1] > 0:
            y.append(1)
        else:
            y.append(-1)
    # changing the column names
    col_names = []
    for i in range(1, k + 1):
        col_names.append("X" + str(i))
    final_data = pd.DataFrame(data[0:], columns=col_names)
    final_data["Y"] = y
    return final_data

def sign(row):
    if row[-1] >= 1:
        return 1
    else:
        return -1

training_data = generate_data(5, 10, 0.3)
```

Generated Dataset:



Perceptron Algorithm:

```
def perceptron(data):
    x = np.asarray(data.iloc[:, :-1])
    b = 0
    w = []
    d = (np.asarray(data.iloc[:,:-1])).tolist()
    d_for_y = (np.asarray(data.iloc[:,:])).tolist()
    for i in range(0, len(d[0])):
        w.append(0)
    steps = 0

    while 1:
        mismatch = 0
        # finding the misclassified point
        for i in range(0, len(d)):
            s = 0
            y = sign(d_for_y[i])
            for j in range(0, len(d[i])):
                s += d[i][j] * w[j]
            s += b                                  # s = w.x + b
            if s > 0:
                f_y = 1
            else:
                f_y = -1
            if y == f_y:
                continue
                #continue if it is correctly classified
            else:
                # Update w and b and increase step number
                mismatch += 1
                steps += 1
                b += y
                for l in range(0, len(w)):
                    w[l] += d[i][l] * y
# Terminate if there is no mismatch of Y when iterated through all the points in the
dataset
        if mismatch == 0:
            break
    # weights = pd.DataFrame(w[0:])
    # print("Steps: ", steps )
    return steps
```

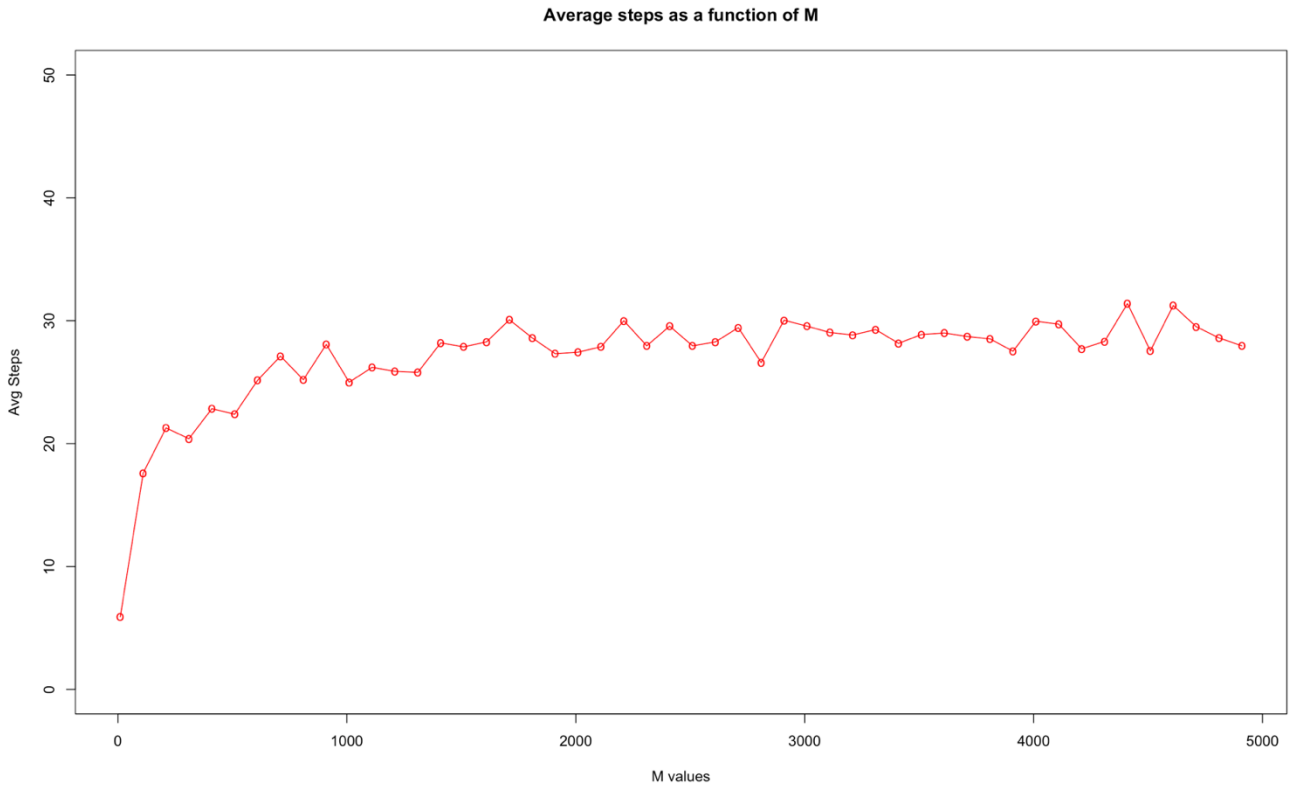
1) For $k = 5, \epsilon = 0.1$, for a range of possible m values, repeatedly generate data sets of size m and fit a perceptron to them. Plot, as a function of m, the average number of steps needed for the Perceptron Learning Algorithm to converge. Do your results make sense? Do you think looking at larger and larger m values outside the range you plot will produce anything different?

Finding the average step value for a range of given m values.

```
def function_of_steps():
    #for k = 5 eps = 0.1
    mv = []
    step = []
    for m in range(10, 5000, 100):
        s = []
        mv.append(m)
        for i in range(50):
            data = generate_data(5, m, 0.1)
            steps = perceptron(data)
            s.append(steps)
        step.append((sum(s)/len(s)))
    print(mv)
    print(step)
```

```
103  function_of_steps()

main
/Users/rithvikananth/pythonProject1/venv/bin/python /Users/rithvikananth/pythonProject1/main.py
[10, 110, 210, 310, 410, 510, 610, 710, 810, 910, 1010, 1110, 1210, 1310, 1410, 1510, 1610, 1710, 1810, 1910, 2010, 2110, 2210, 2310, 2410, 2510, 2610, 2710, 2810, 2910,
[6.3, 16.68, 21.32, 22.36, 21.16, 23.76, 24.24, 25.04, 27.0, 23.84, 24.64, 26.92, 25.84, 27.36, 26.32, 29.24, 27.44, 28.48, 27.04, 28.76, 28.88, 28.48, 29.36, 28.08, 28.
```

We observe an increase in the average step size with the increase in M values initially, but after m-value greater than 2000 we observe that average steps are almost the same. The larger and larger m values outside the range of plot would also maintain the same average steps observed according to my understanding.



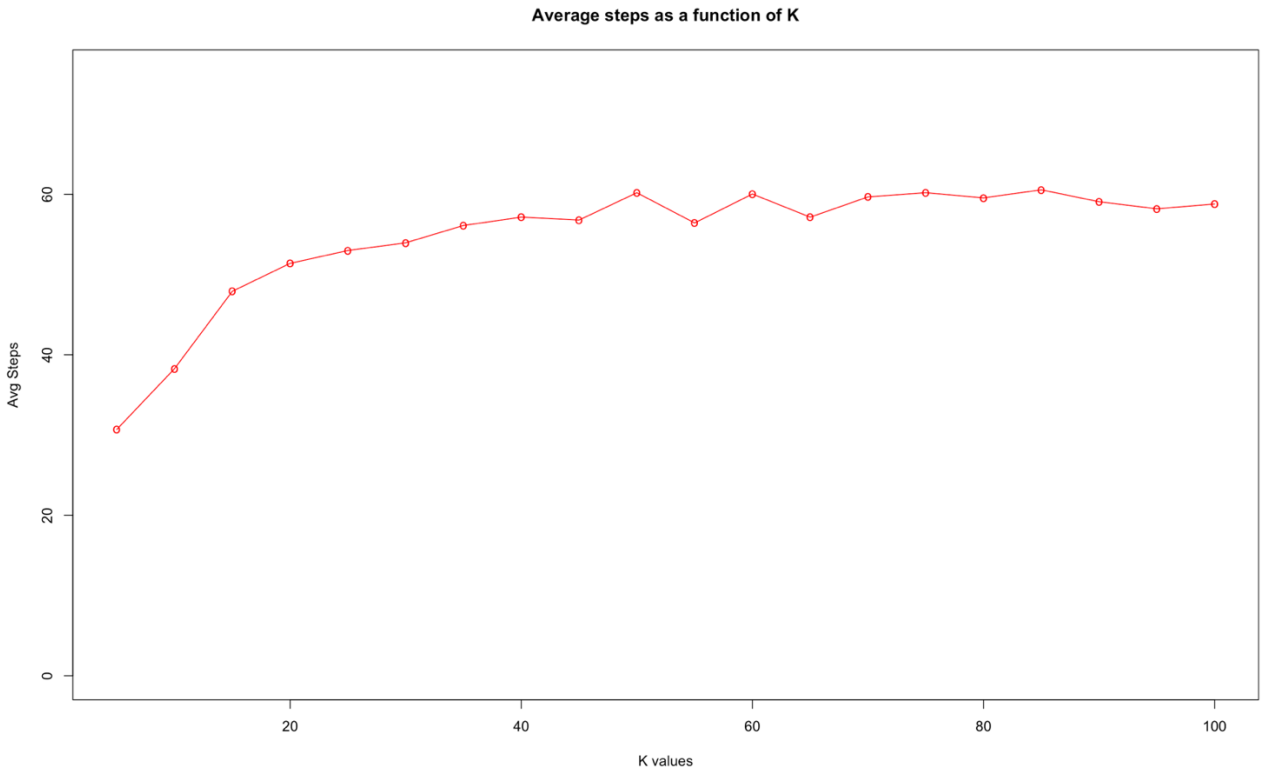
2) For $m = 100, \epsilon = 0.05$, for a range of possible k values, repeatedly generate data sets of dimension k, and fit a perceptron to them. Plot, as a function of k, the average number of steps needed for the Perceptron Learning Algorithm to converge. Do your results make sense? Do you think looking at larger and larger k values outside the range you plot will produce anything different?

```
def function_of_features():
    kv = []
    step = []
    for k in range(5, 101, 5):
        s = []
        kv.append(k)
        for i in range(50):
            data = generate_data(k, 100, 0.05)
            steps = perceptron(data)
            s.append(steps)
        step.append((sum(s) / len(s)))
    print(kv)
    print(step)
```

```
114    function_of_features()
```

```
main
/Users/rithvikananth/pythonProject1/venv/bin/python /Users/rithvikananth/pythonProject1/main.py
[5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100]
[30.7, 38.28, 47.92, 51.4, 53.0, 53.96, 56.12, 57.16, 56.8, 60.2, 56.44, 60.04, 57.16, 59.68, 60.2, 59.56, 60.56, 59.08, 58.2, 58.8]
```

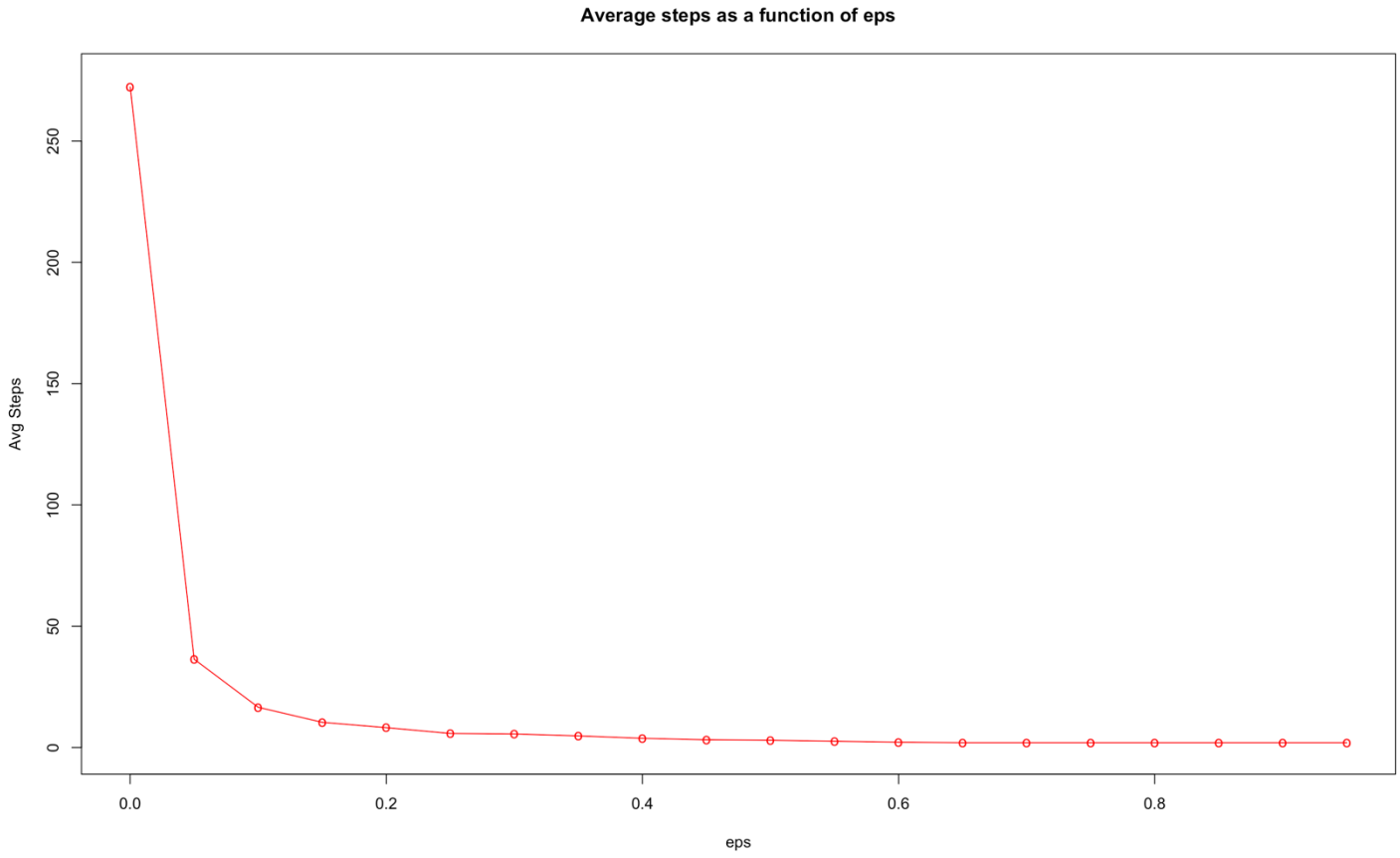
A marginal increase in the average step size is observed with the increase in K values but. For K-value greater than 45 the step size is constant. To my understanding, larger and larger K values outside the range of plot would also maintain the same average steps observed.



3) For $k = 5, m = 100$, for a range of possible ϵ values in $[0, 1]$, repeatedly generate data sets with an ϵ-threshold cutoff and fit a perceptron to them. Plot, as a function of ϵ, the average number of steps needed for the Perceptron Learning Algorithm to converge. Do your results make sense? Do you think looking at different k, m values will produce anything different?

```python
def function_of_eps():
    eps_val = []
    step = []
    eps = 0
    for i in range(21):
        s = []
        eps_val.append(eps)
        for j in range(50): # Taking the average
            data = generate_data(5, 100, eps)
            steps = perceptron(data)
            s.append(steps)
        step.append((sum(s) / len(s)))
        eps += 0.05
    print(eps_val)
    print(step)
```

```
137    function_of_eps()

main ×
/Users/rithvikananth/pythonProject1/venv/bin/python /Users/rithvikananth/pythonProject1/main.py
[0, 0.05, 0.1, 0.15000000000000002, 0.2, 0.25, 0.3, 0.35, 0.39999999999999997, 0.44999999999999996, 0.49999999999999994, 0.5499999999999999, 0.6, 0.65, 0.70
[272.3, 36.4, 16.6, 10.4, 8.2, 5.8, 5.6, 4.8, 3.8, 3.2, 3.0, 2.6, 2.2, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0]
```

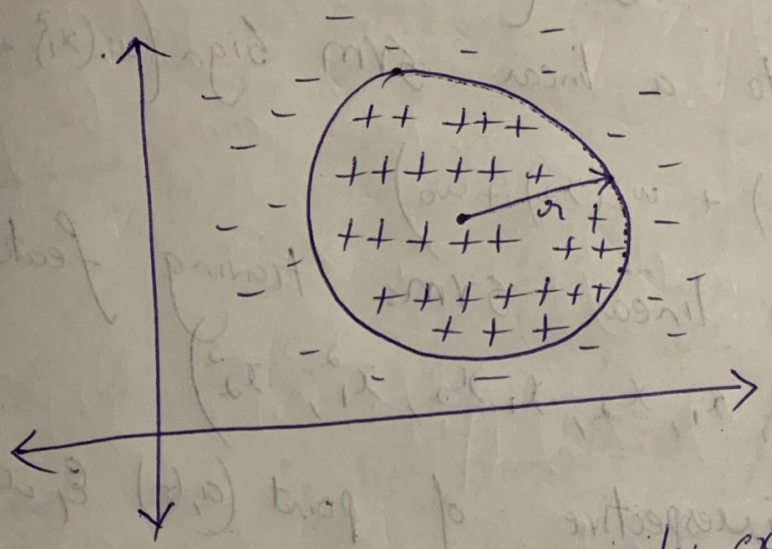


As the margin increases, we can clearly observe that the average number of steps taken to converge reduces. At margin 0.3 or greater we observe the average step size is 2. Irrespective of what the K and m values are, the graph for epsilon vs average steps would remain the same.

R- code:

```
plot(m, steps, type="o", col="red", pch="o", lty=1, ylim=c(0,50), ylab="Avg Steps", xlab = "M values")
+ title("Average steps as a function of M")


plot(K, steps, type="o", col="red", pch="o", lty=1, ylim=c(0,75), ylab="Avg Steps", xlab = "K values") +
  title("Average steps as a function of K")


plot(eps, steps, type="o", col="red", pch="o", lty=1, ylim=c(0,22), ylab="Avg Steps", xlab = "eps") +
  title("Average steps as a function of eps")
```

1) Given that positive class lay within a certain radius of a point, negative ones lie outside that radius.



As told in the class of a similar example (unit circle)

The features of the data are $1, x_1, x_2, x_1^2, x_2^2$

$= \phi(x_1, x_2)$

Equivalently with kernel

$$K(\underline{x}, \underline{y}) = (1 + \underline{x} \cdot \underline{y})^2$$

The positive points are defined as

$x_1^2 + x_2^2 \leq 1$ (or) $0 \leq 1 - x_1^2 - x_2^2$

In class (for unit circle) classifier was shown as $\Rightarrow$ $\text{Sign}(1 - x_1^2 - x_2^2)$

for the given data at point (a, b) with radius r

$\Rightarrow \text{Sign}(r^2 - (x_1 - a)^2 - (x_2 - b)^2)$

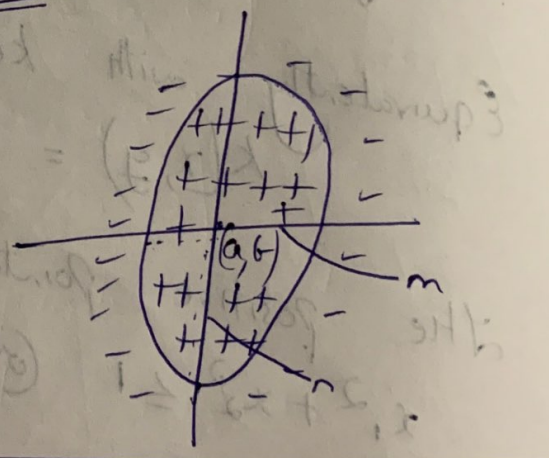
$$\Rightarrow \text{Sign}\left(r^2 - (x_1^2 + a^2 - 2ax_1) - (x_2^2 + b^2 - 2bx_2)\right)$$

$$\Rightarrow \text{Sign}\left(r^2 - a^2 - b^2 + 2x_1 a + 2x_2 b - x_1^2 - x_2^2\right)$$

Hence to a linear SVM $\text{Sign}\left(w_0 + w_1(x_1) + w_{11}(x_1)^2 + w_2(x_2) + w_{22}(x_2)^2 + w_{12}(x_1 x_2)\right)$ with linear SVMs having features $(1, x_1, x_2, x_1 x_2, x_1^2, x_2^2)$

∴ So irrespective of point (a, b) & radius r the data will always have a linear seperator.

For ellipsodal Seperator



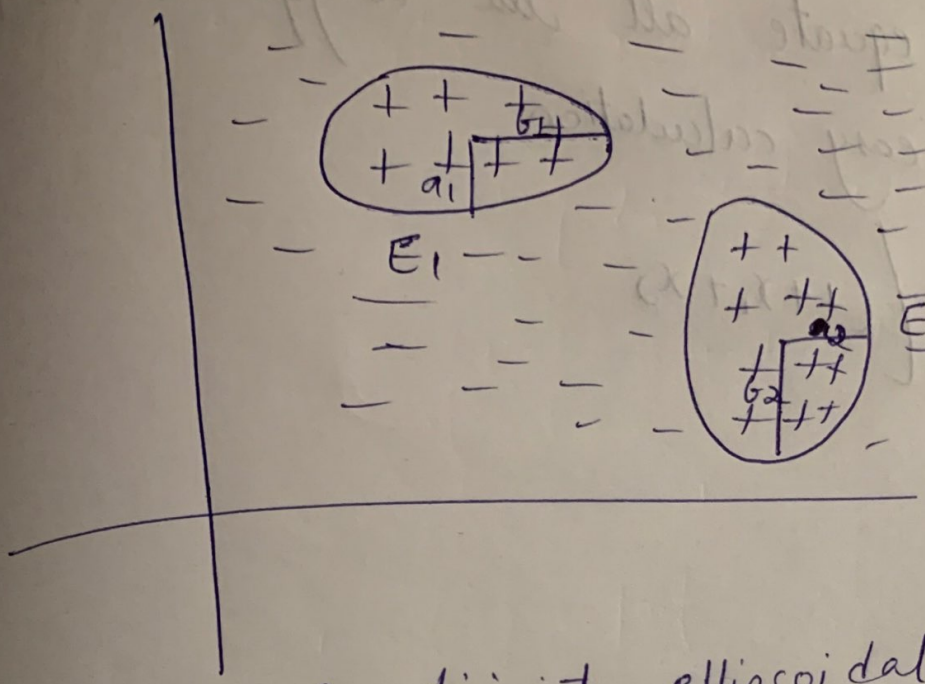
We classify $(\underline{x})$ as

$$\text{Sign}\left(m^2(x_1 - a)^2 + n^2(x_2 - b)^2 - m^2 n^2\right) = S$$

$$\Rightarrow \begin{cases} +1 & S \leq 0 \\ -1 & S > 0 \end{cases}$$

We see that the parameters for this equation can also be achieved by the same mapping function.

2)



Consider 2 disjoint ellipsoidal regions with centers (m_1, n_1) & (m_2, n_2)

major axis ~~$b_1 b_2$~~ $a_1 b_1$ minor axis ~~$a_1 a_2$~~ $a_2 b_2$.

Given that the positive points lie inside E_1 & E_2, for any positive class data points (x_1, x_2)

$$\left(1 - \frac{(x_1 - m_1)^2}{b_1^2} - \frac{(x_2 - n_1)^2}{a_1^2} \geq 0\right) \text{ (or)}$$

$$\left(1 - \frac{(x_1 - m_2)^2}{b_2^2} - \frac{(x_2 - n_2)^2}{a_2^2} \geq 0\right)$$

The classifier for above would be

$$\text{sign}\left(\left[a_1^2 b_1^2 - a_1^2 (x_1 - m_1)^2 - b_1^2 (x_2 - n_1)^2\right] \times \left[a_2^2 b_2^2 - a_2^2 (x_1 - m_2)^2 - b_2^2 (x_2 - n_2)^2\right]\right)$$

Equating all coefficients to 1 for easy output

$$\Rightarrow \text{Sign}\Big((1 + x_1 + x_2 + x_1^2 + x_2^2)\big(1 + x_1 + x_2 + x_1^2 + \dots$$

$$\text{Sign}\Big(1 + x_1 + x_2 + x_1^2 + x_2^2 + x_1^2 + x_1x_2 + x_1^3 + x_2^2x_1$$
$$+ x_2 + x_1x_2 + x_2^2 + x_1^2x_2 + x_2^3 + x_1^2 + x_1^3 + x_1^2x_2 + x_1^4$$
$$+ x_1^2x_2^2 + x_2^2 + x_1x_2^2 + x_2^3 + x_1^2x_2^2 + x_2^4\Big)$$

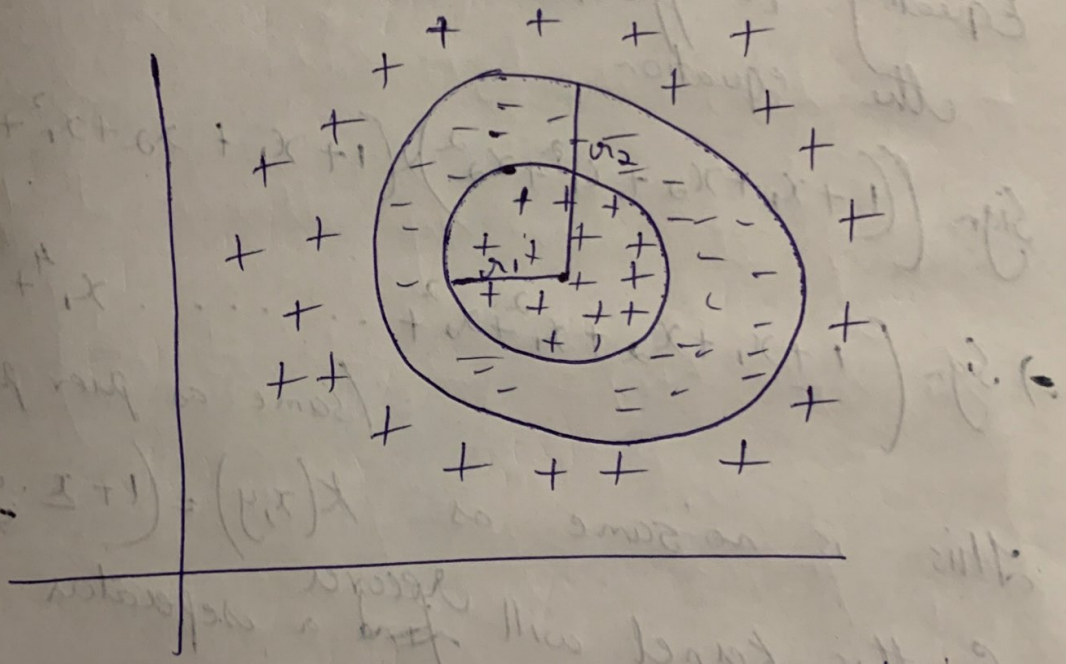
we get the parameters as

$(1, x_1, x_2, x_1x_2, x_1^2, x_2^2, x_1^2x_2^2, x_1x_2^2 \ldots \ldots x_1^4, x_2^4)$

We can Hence use the kernel function to represent this mapping function through

$$K(\underline{x}, \underline{y}) = (1 + \underline{x} \cdot \underline{y})^4$$

3)



Let r_1 & r_2 be the radius of inner & outer circles. respectively

Center is (a, b)

Positive points :- $(x_1 - a)^2 + (x_2 - b)^2 \leq r_1^2$

(a) $(x_1 - a)^2 + (x_2 - b)^2 > r_2^2$

The classifier for above would be

$$\text{Sign}\left[\left[r_1^2-(x_1-a)^2-(x_2-b)^2\right]\left[r_2^2-(x_1-a)^2-(x_2-b)^2\right]\right]$$

$$\text{Sign}\left[\left[r_1^2-x_1^2-a^2+2x_1a-x_2^2-b^2+2x_2b\right]\left[r_2^2-x_1^2-a^2+2x_1a-x_2^2-b^2+2x_2b\right]\right]$$

Equating co-efficients to 1 & calculating the equation

$$\text{Sign}\left[\left(1+x_1+x_2+x_1^2+x_2^2\right)\left(1+x_1+x_2+x_1^2+x_2^2\right)\right]$$

$$\Rightarrow \text{Sign}\left[1+x_1+x_2+x_1^2+x_2^2+\ldots\ldots\ldots x_1^4+x_2^4\right]$$

(Same as prev problem)

This is same as $K(x,y)=(1+x\cdot y)^4$

& this kernel will recover a seperator

4) Given XOR data $(\pm 1, \pm 1)$

x		
x_1	$+1$	-1
x_2	-1	$+1$
x_3	$+1$	-1
x_4	$+1$	$+1$

y	→ Result
-1	
$+1$	
$+1$	
-1	

$K(\underline{x}, \underline{y}) = (1 + \underline{x} \cdot \underline{y})^2$

we can find the values of as the following

$K_{11} = \left[1 + \begin{bmatrix} -1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & -1 \end{bmatrix}\right]^2$

$= (1+2)^2 = 9$

$K_{12} = \left[1 + \begin{bmatrix} -1 \\ +1 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix}\right]^2$

$\Rightarrow [1+0]^2 \Rightarrow 01$

K matrix $\Rightarrow$ $\begin{bmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{bmatrix}$

[K x result (y)]

$\begin{bmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{bmatrix}$ $\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$ $\Rightarrow$ $\begin{bmatrix} 9 & -1 & -1 & 1 \\ -1 & 9 & 1 & -1 \\ -1 & 1 & 9 & -1 \\ 1 & -1 & -1 & 9 \end{bmatrix}$

$[-1 \quad 1 \quad +1 \quad -1]$

Kernel function for SVM is

$\max \sum_{i=1}^{n} \alpha_i = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i y_0^i k_{ij} y^j \alpha_j$

such that $\sum_{i=1}^{n} \alpha_i y^i = 0 \quad \forall \alpha_i \geq 0$

So Now

$$\max_{\alpha_1 \cdots \alpha_4} \left(\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4\right) - \frac{1}{2}\Big[9\alpha_1^2 + 9\alpha_2^2 + 9\alpha_3^2 + 9\alpha_4^2\Big] + 2\Big(-\alpha_1\alpha_2 + \alpha_2\alpha_3 + \alpha_4\alpha_1 - \alpha_3\alpha_4 - \alpha_2\alpha_4 - \alpha_1\alpha_3\Big)\Big]$$

We have max fn with $\alpha_1, \alpha_2, \alpha_3, \alpha_4$

$$\max_{\alpha_1 \cdots \alpha_4} \left(\sum_{i=1}^{4} \alpha_i\right) - \frac{9}{2}\left[\alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2\right]$$

$$+ 2\Big(-\alpha_1\alpha_2 + \alpha_2\alpha_3 + \alpha_4\alpha_1 - \alpha_3\alpha_4 - \alpha_2\alpha_4 - \alpha_1\alpha_3\Big)$$

Solving the above eqn max is at $1/4$

$$\left(\alpha_1, \alpha_2, \alpha_3, \alpha_4\right) = \left(\frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}\right)$$

We find the sign function

$\underline{w} \cdot \underline{x}$ is given by

$$\sum_{i=1}^{n} \alpha_i y^i k(x^i, \underline{x})$$

$$\Rightarrow \alpha_1 y_1 k_{1x} + \alpha_2 y_2 k_{2x} + \alpha_3 y_3 k_{3x} + \alpha_4 y_4 k_{4x}$$

$$\Rightarrow \frac{1}{8}\left[-k_{1x} + k_{2x} + k_{3x} - k_{4x}\right]$$

bias $b = y^i - \underline{w} x$

$i = 1$

$\underline{w} x_i = \frac{1}{8}\left[-k_{11} + k_{21} + k_{31} - k_{41}\right]$

$= \frac{1}{8}\left[-9 + 1 + 1 - 1\right]$

$= -1$

$b = -1 - (-1) = \boxed{0}$

∴ The seperator

$\frac{1}{8}\left[-k_1 x + k_2 x + k_3 x - k_4 x\right] + 0$

$\boxed{K = (1 + x_i \cdot x)^2}$

$\Rightarrow K(x, y) = e^{-(|x| - |y|)^2}$

Calculating the kernal values

$|x - y| = (x_1 - y_1)^2 + (x_2 + y_2)^2$

$k_{11} = e^{-((1+1)^2 + (1+1)^2)}$

$= e^0 = 1$

$k_{12} = e^{-((1+1)^2 + (1-1)^2)}$

$= e^{-4}$

Doing so for XOR we get the K matrix as

$$\begin{bmatrix} 1 & e^{-4} & e^{-4} & e^{-8} \\ e^{-4} & 1 & e^{-8} & e^{-4} \\ e^{-4} & e^{-8} & 1 & e^{-4} \\ e^{-8} & e^{-4} & e^{-4} & 1 \end{bmatrix}$$

finding $\max_{\alpha_i} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i y^i k_{ij} y^j \alpha_j$

we get $\alpha = \begin{bmatrix} e^{0.03} & e^{0.03} & e^{0.03} & e^{0.03} \end{bmatrix}$

by solving the eqn

$$\alpha \cong \begin{bmatrix} 1.03, & 1.03, & 1.03, & 1.03 \end{bmatrix}$$

$$b = y^i - \underline{w}\, x$$

$$\underline{w}\, x_i = \sum_{i=1}^{m} \alpha_i y^i k(x^i, x)$$

$$= 1.03\left[-k_{1x} + k_{2x} + k_{3x} - k_{4x}\right]$$

Put $i=1$

$$1.03\left[-1 + e^{-4} + e^{-4} - e^{-8}\right]$$

$$\Rightarrow -0.999 \cong -1$$

$$b = y_i - w_i x_i = -1 - (-1) = 0.$$

Separator is

$$1.03\left[-k_{1x} + k_{2x} + k_{3x} - k_{4x}\right] + 0$$

$$k_{1x} = e^{\left(-|x_1 - x|^2\right)}$$