Homework3

October 22, 2020

1 CS 536 : Perceptrons and SVMs

Fatima AlSaadeh (fya7)

2 Perceptrons

[1]:
```
import pandas
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
```

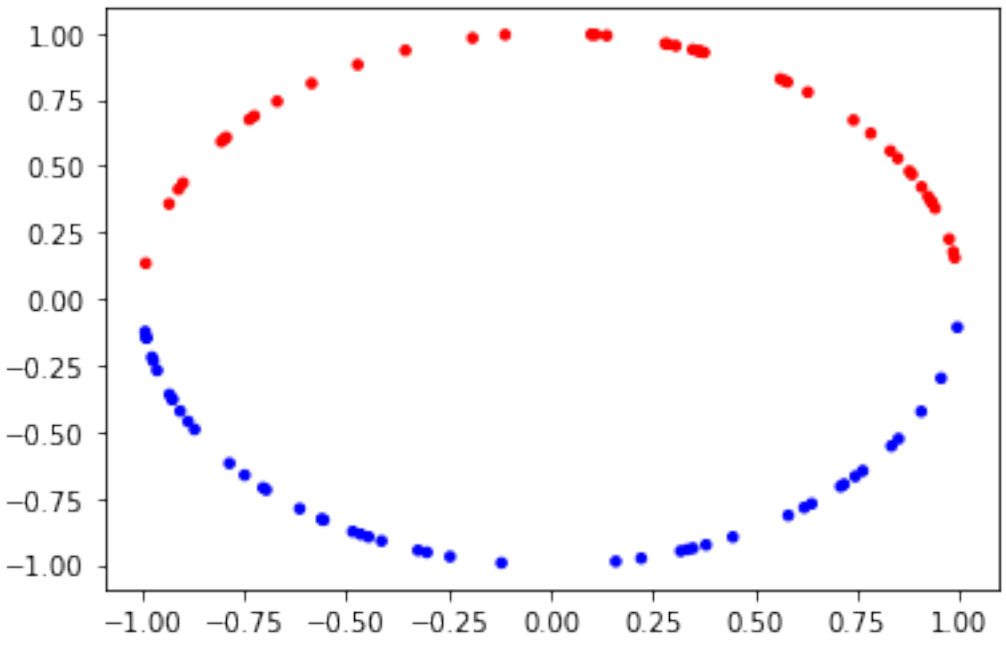
[12]:
```
import pandas as pd
#generate data function et Z be a k-dimensional vector, where each entry
#is an i.i.d. standard normal, mean 0 and variance 1.
#Then define X = Z/||Z|| (taking the norm as the 2-norm or Euclidean norm),
#so that X is a random vector of length 1, lying exactly on the k-dimensional␣
 ↪unit sphere.
def generate_data(k, m, e):
    all_x = []
    all_y = []
    for i in range(m):
        z = np.random.normal(0, 1, k)
        x = z/np.linalg.norm(z)
        temp_x = []
        for j in range(len(x)) :
            if abs(x[j])>=e:
                temp_x.append(x[j])
        all_y.append(1) if x[j]>= e else all_y.append(-1)
        all_x.append(temp_x)
    return all_x, all_y
```

[9]:
```
#test code generator in the two dimensional data
x, y = generate_data(2,100, 0.1)
x_df = pd.DataFrame(x)
y_df = pd.DataFrame(y)
print(x_df)
```

```
print(x_df.shape)
print(y_df)
colors = np.where(y_df[0] == -1, 'b', 'r')
plt.scatter(x_df[0], x_df[1], s=12, c=colors)
```

```
           0         1
0   0.999927       NaN
1  -0.925608 -0.378484
2  -0.555383 -0.831595
3   0.619720 -0.784823
4   0.221027 -0.975268
..       ...       ...
95 -0.933898  0.357538
96 -0.992342 -0.123523
97 -0.990965  0.134124
98  0.333629 -0.942704
99  0.995333       NaN

[100 rows x 2 columns]
(100, 2)
    0
0  -1
1  -1
2  -1
3  -1
4  -1
.. ..
95  1
96 -1
97  1
98 -1
99 -1

[100 rows x 1 columns]
```

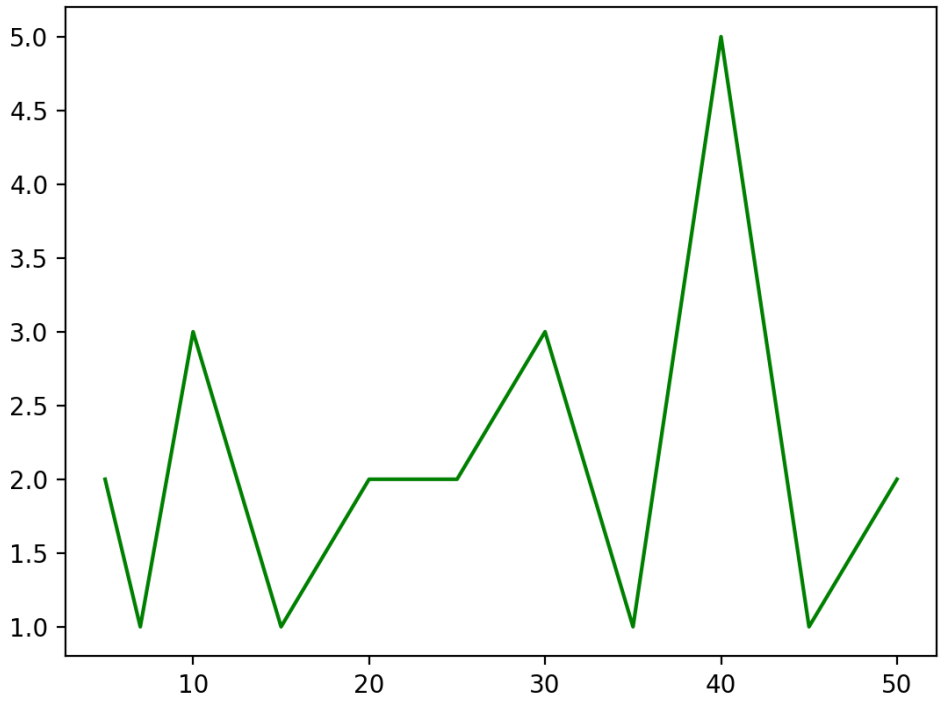
[9]: <matplotlib.collections.PathCollection at 0x11f7cd048>



[3]:
```
#perceptron algorithm
# start with w,b = 0
# Identify a point that is misclassified, i.e.,f(x[i])!=y[i]
# update the wight and bias w,b
# repeat until all points are correctly classified
def perceptron(x, y):
    m, k = x.shape
    w = np.zeros(k)
    b = 0
    steps =10
    miss = np.zeros(10)
    while steps != 0:
        for i in range(m):
            fx = np.sign(np.dot(w, x.loc[i,:].values) + b)
            if y[0][i] != fx:
                miss[steps-1] =1
                w = w + (x.loc[i,:] * y[0][i])
                b = b + y[0][i]
        steps -=1
    return w, b, miss
```

[4]:
```
# For k = 5,  = 0.1, for a range of possible m values,
#repeatedly generate data sets of size m and fit a perceptron to them.
def fit_on_m():
    ms = [5,10,15,20,25,30,35,40,45,50]
    steps = []
```

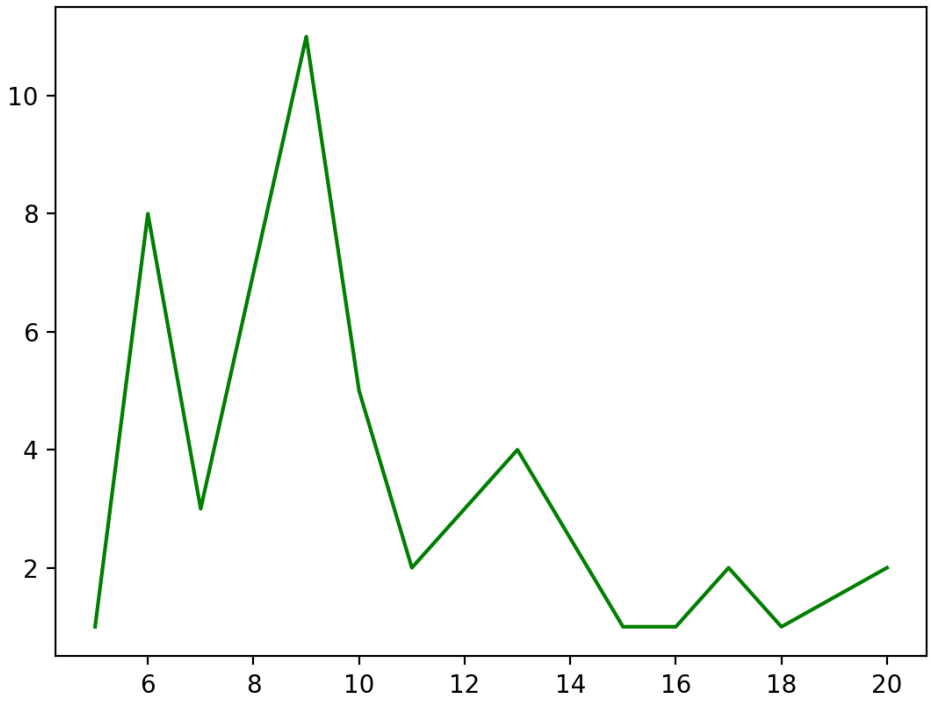
```
    for m in ms :
            new_x, new_y = generate_data(5,m, 0.1)
            new_x_df = pd.DataFrame(new_x)
            new_y_df = pd.DataFrame(new_y)
            w, b, miss= perceptron(new_x_df,new_y_df)
            steps.append(sum(miss))
    plt.plot(ms, steps, 'g')
    plt.show()
fit_on_m()
```



Analysis: In this plots with m in the range [5,50] the number of steps needed for the algorithm varies between 1 and 5 at each m and so I concluded that the number of steps needed doesn't depend on the value of m, this make sense as we defined in the class the number of steps needed depends on the margin of the separator and independent of the underlying dimension. and I don't expect this to change outside the range of the selected m values.

[5]:
```
#For m = 100,  = 0.05, for a range of possible k values,
#repeatedly generate data sets of dimension k, and fit a perceptron to them.
def fit_on_k():
    ks = [5,6,7,9,10,11,12,13,15,16,17,18,20]
    steps = []
```

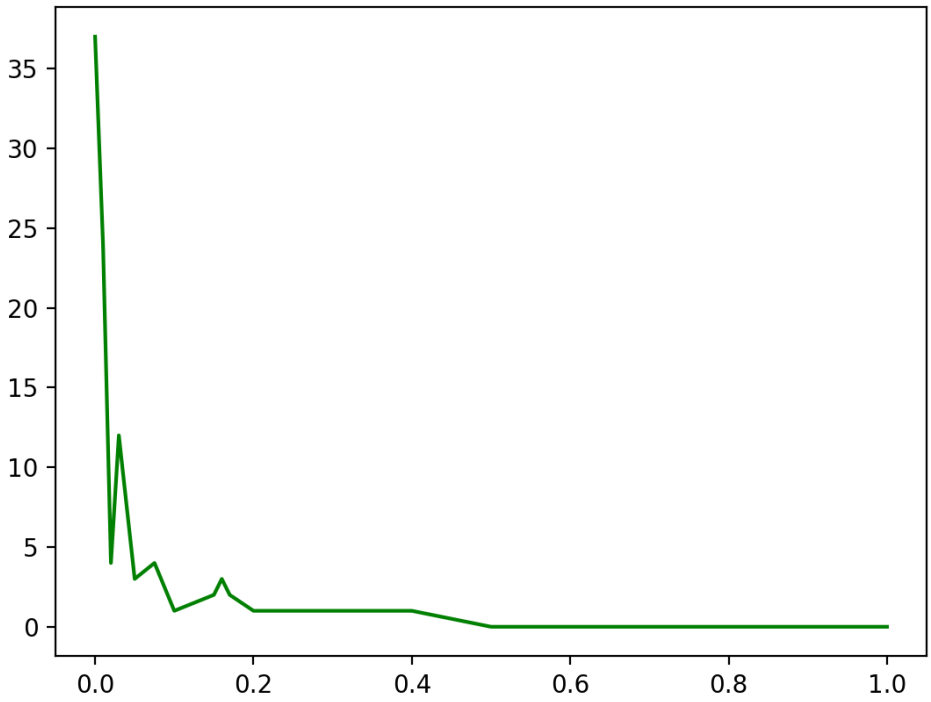
```
    for k in ks :
        x_new, y_new = generate_data(k,100, 0.05)
        x_df_new = pd.DataFrame(x_new)
        y_df_new = pd.DataFrame(y_new)
        w, b, miss= perceptron(x_df_new,y_df_new)
        steps.append(sum(miss))
    plt.plot(ks, steps, 'g')
    plt.show()
fit_on_k()
```



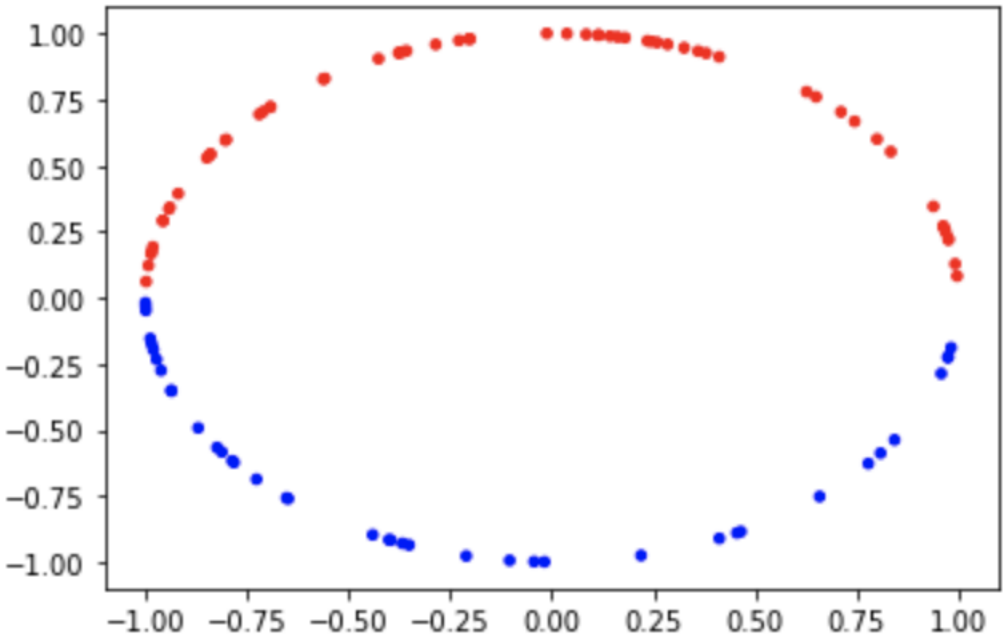
Analysis: In this plots with k in the range [5,20] the number of steps needed for the algorithm varies between 1 and 10 at each k. this being said I concluded that the number of steps needed doesn't depend on the value of k, this make sense as we defined in the class the number of steps needed depends on the margin of the separator and independent of the underlying dimension.and I don't expect this to change outside the range of the selected k values.

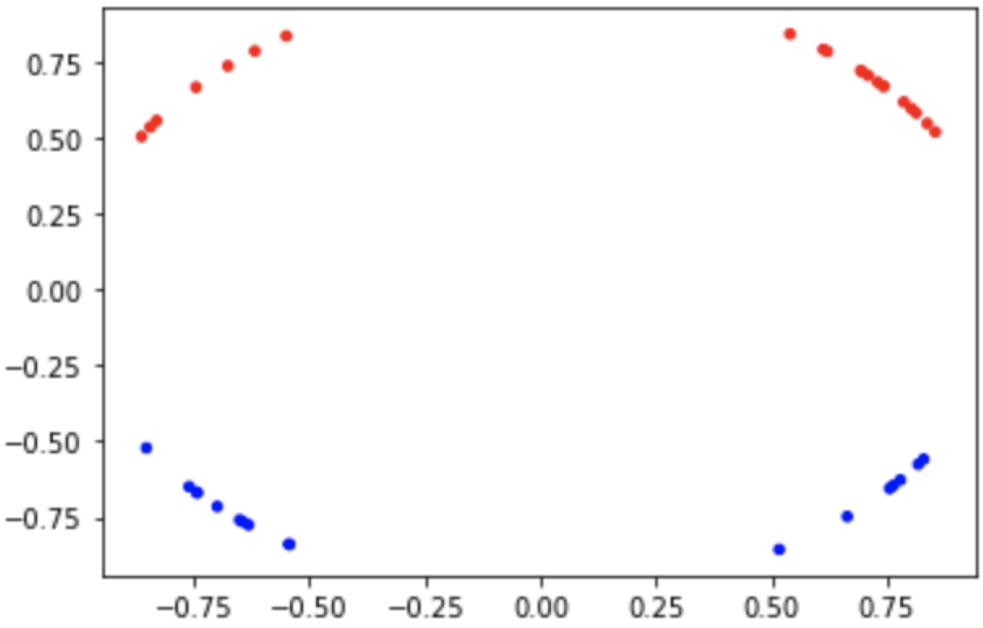
[8]:
```
#For k = 5, m = 100, for a range of possible  values in [0, 1],
#repeatedly generate data sets with an -threshold cutoff and fit a perceptron to␣
 ↪them.
def fit_on_e():
    es = [0, 0.05, 0.075, 0.1,0.2,0.25,0.5, 0.75, 1]
```

```
    steps = []
    for e in es :
        x, y = generate_data(5,100, e)
        x_df = pd.DataFrame(x)
        y_df = pd.DataFrame(y)
        w, b, miss= perceptron(x_df,y_df)
        steps.append(sum(miss))
    plt.plot(es, steps, 'g')
    plt.show()
fit_on_e()
```

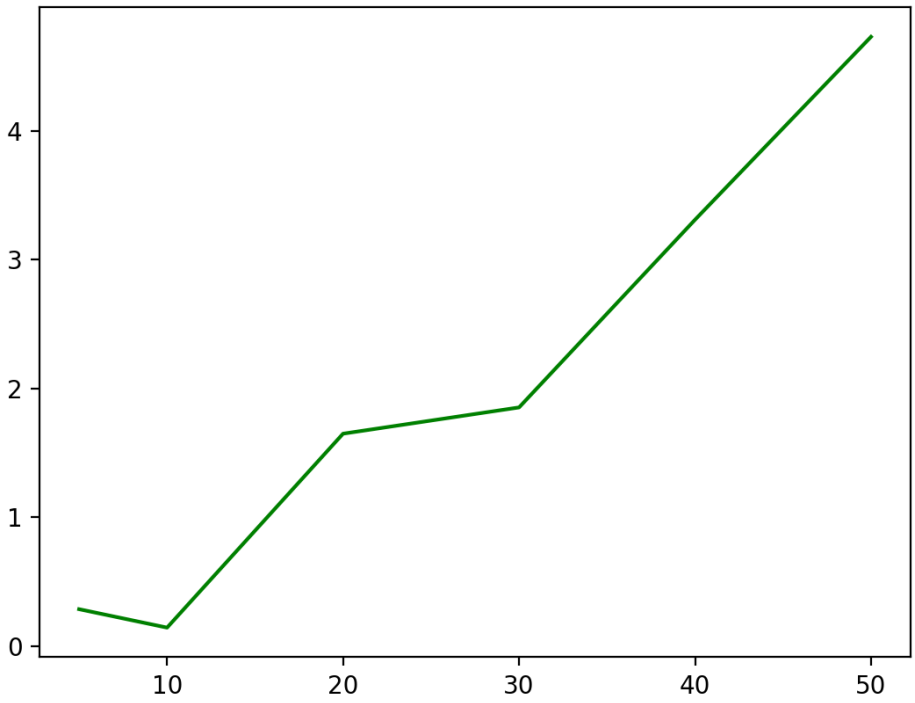


Analysis: with epsilon in the range [0,1] the plot is showing us at epsilon 0 the algorithm needed more than 35 steps to converge, the number if steps varies in the range [0.01,0.2] between 1 and 12 steps after 0.2 the number of steps decreases to become closer to 1, at epsilon 1, it was hard to generate data as we are ignoring any row with value less than epsilon, while the randon normal function I was using generates 80% of the values less than 1. this behavior described is expected to happen regardless of the value of k and m. this make sense as the smaller epsilon is, the harder to find separator between the data for example if we try epsilon .01 and .5 for k,m=2,100 we find the following two plots 0.01, 0.5 respectively it shows how close the data is with epsilon 0.01 and how far apart with epsilon 0.5


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

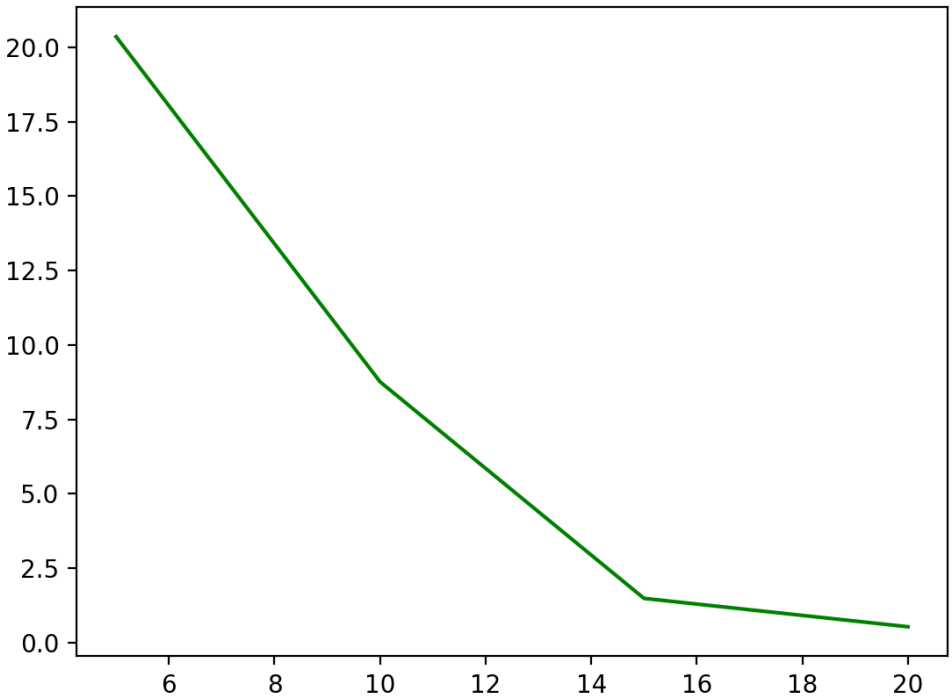

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75 −0.50 −0.25 0.00 0.25 0.50 0.75

[19]:
```
#find how close the 'typical' perceptron generated is to this 'ideal' perceptron␣
 ↪by changing m
#w,b the typical perceptron [0,0,..,1] and 0.0
#on the same m value generate 10 more perceptron average them and assume the␣
 ↪average is the ideal perceptron
#find the distance between the ideal and the typical
def fit_on_m_bonus():
    ms = [5,10,15,20,25,30,35,40,45,50]
    steps = []
    vals = []
    wss = []
    bss = []
    for m in ms :
            wss = []
            bss = []
            new_x, new_y = generate_data(5,m, 0.1)
            new_x_df = pd.DataFrame(new_x)
            new_y_df = pd.DataFrame(new_y)
            firstw, firstb, miss= perceptron(new_x_df,new_y_df)
            for i in range(10):
                new_x, new_y = generate_data(5,m, 0.1)
                new_x_df = pd.DataFrame(new_x)
                new_y_df = pd.DataFrame(new_y)
                w, b, miss= perceptron(new_x_df,new_y_df)
                if len(w)>0:
                    wss.append(list(w))
                    bss.append(b)
            wssavg = np.array(wss).mean(axis=0)
            bssavg = sum(bss)/len(bss)
            val = (np.linalg.norm(wssavg-firstw))**2 + (bssavg-firstb)**2
            vals.append(val)
            steps.append(sum(miss))
    plt.plot(ms, vals, 'g')
    plt.show()
fit_on_m_bonus()
```

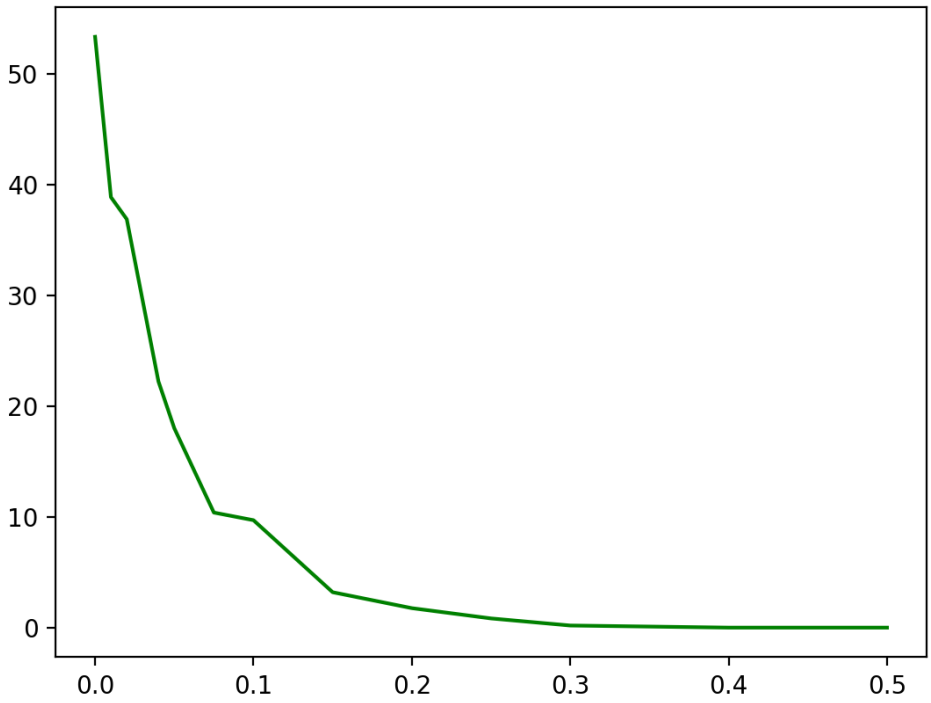


[21]:
```
#find how close the 'typical' perceptron generated is to this 'ideal' perceptron␣
 ↪by changing m
#w,b the typical perceptron [0,0,..,1] and 0.0
#on the same k value generate 10 more perceptron average them and assume the␣
 ↪average is the ideal perceptron
#find the distance between the ideal and the typical
def fit_on_k_bonus():
    ks = [5,6,7,9,10,11,12,13,15,16,17,18,20]
    steps = []
    vals = []
    wss = []
    bss = []
    for k in ks :
        wss = []
        bss = []
        new_x, new_y = generate_data(k, 100, 0.05)
        new_x_df = pd.DataFrame(new_x)
        new_y_df = pd.DataFrame(new_y)
        firstw, firstb, miss = perceptron(new_x_df, new_y_df)
        for i in range(10):
```

```
            new_x, new_y = generate_data(k, 100, 0.05)
            new_x_df = pd.DataFrame(new_x)
            new_y_df = pd.DataFrame(new_y)
            w, b, miss = perceptron(new_x_df, new_y_df)
            if len(w) > 0:
                wss.append(w.values)
                bss.append(b)
        wssavg = np.array(wss).mean(axis=0)
        bssavg = sum(bss) / len(bss)
        val = (np.linalg.norm(wssavg - firstw)) ** 2 + (bssavg - firstb) ** 2
        vals.append(val)
        steps.append(sum(miss))

    plt.plot(ks, vals, 'g')
    plt.show()
fit_on_k_bonus()
```



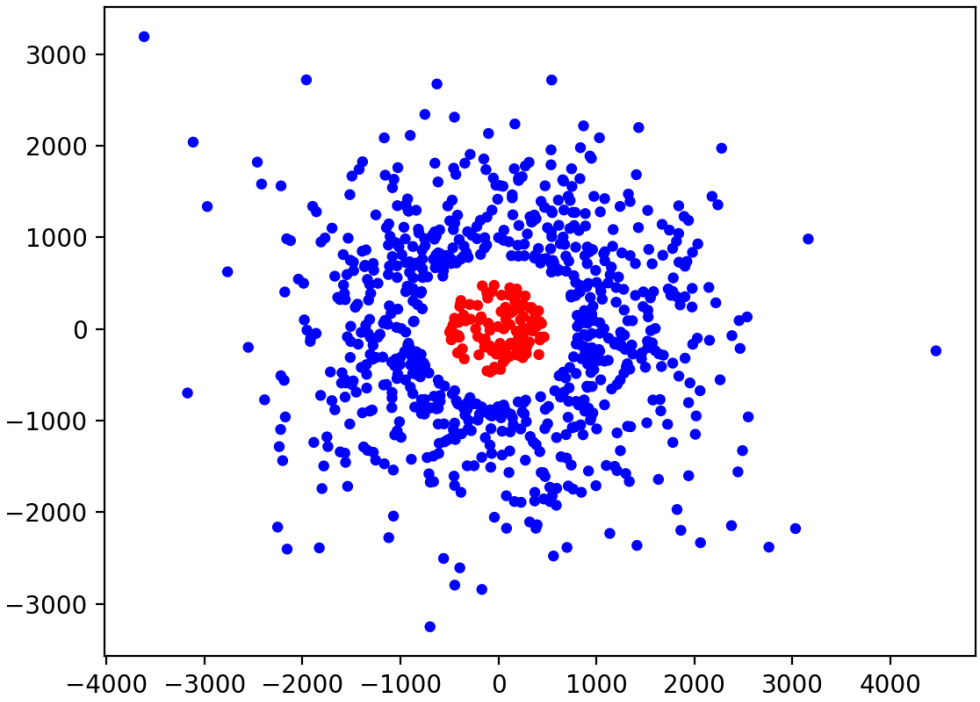
```
[ ]: #find how close the 'typical' perceptron generated is to this 'ideal' perceptron␣
      ↪by changing m
     #w,b the typical perceptron [0,0,..,1] and 0.0
     #on the same e value generate 10 more perceptron average them and assume the␣
      ↪average is the ideal perceptron
     #find the distance between the ideal and the typical
     def fit_on_e_bonus():
         es = [0, .01,.02,.04 ,.05, .075, .1,.15,.2, .25,.3,.4, .5]
         steps = []
         vals = []
         for e in es :
             wss = []
             bss = []
             new_x, new_y = generate_data(5,100, e)
             new_x_df = pd.DataFrame(new_x)
             new_y_df = pd.DataFrame(new_y)
             firstw, firstb, miss = perceptron(new_x_df, new_y_df)
             for i in range(10):
                 new_x, new_y = generate_data(5,100, e)
                 new_x_df = pd.DataFrame(new_x)
                 new_y_df = pd.DataFrame(new_y)
                 w, b, miss = perceptron(new_x_df, new_y_df)
                 if len(w) > 0:
                     wss.append(w.values)
                     bss.append(b)
             if len(bss) >0:
                 wssavg = np.array(wss).mean(axis=0)
                 bssavg = sum(bss) / len(bss)
                 val = (np.linalg.norm(wssavg - firstw)) ** 2 + (bssavg - firstb) ** 2
                 vals.append(val)
             else :
                 vals.append(0)

         plt.plot(es, vals, 'g')
         plt.show()
     fit_on_e_bonus()
```



Analysis: The distance between the ideal and the typical Perceptrons increase as m increase when m in the range [5,50] the distance reaches the maximum when m is 50. This behavior wasn't the same for k and epsilon the distance was decreasing as epsilon and k increase, when k =20 the distance was closer to 0 and epsilon as reaching 0.5 the distance was closer to 0, that makes me conclude that on higher e. for k I tried it multiple times and some results didn't indicate that the increase of k will always decrease the distance.

3 SVMs

1)



an example of the data described in thew question is visualized in the picture above where the positive points are on red and the negative points on blue. to be able to separate this data we will need one more dimension to be added, let's assume the current dimensions are called x1, x2, and the new added dimension is z, the new dimension z will be the square distance of the point from the center a,b:

$$z^2 = (x_1 - a)^2 + (x_2 - b)^2$$

$$x_1^2 + x_2^2 + 2 * x_1 * a + 2 * x_2 * b + a^2 + b^2 - z^2 = 0$$

This means the sign function that we are looking for in order to classify our data will be :

$$F = sign(-1 * (x_1^2 + x_2^2 + 2 * x_1 * a + 2 * x_2 * b + a^2 + b^2 - z^2))$$

if F = +1 the data will be inside the circle and if F = -1 the data will be outside the circle and this means with weights (2a,2b,1,1) and intercept (a^2 +b^2 - z^2), we were able to find the linear separator in the embedding feature space $(1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$ using polynomial kernel regardless of the radius or were the data is centered.

- similar to what we did above but with the equation of the ellipsoid :

$$1 = c * (x_1 - a)^2 + d * (x_2 - b)^2$$

$$c * x_1^2 + d * x_2^2 - c * a * x_1 - b * d * x_2 + c * a^2 + d * b^2 - 1 = 0$$

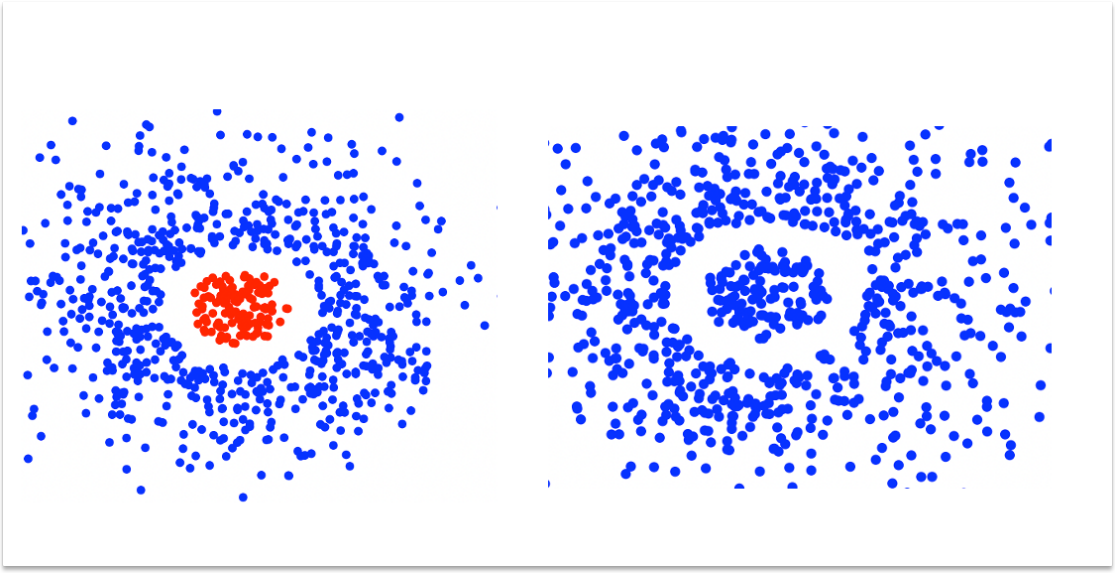
This mean the sign function that we are looking for in order to classify our data will be :

$$F = sign(-1 * (c * x_1^2 + d * x_2^2 - c * a * x_1 - b * d * x_2 + c * a^2 + d * b^2 - 1))$$

if F = +1 the data will be inside the ellipsoid and if F = -1 the data will be outside the ellipsoid and this means with the weights vector (c,d,c*a,b*d) and the intercept ($c * a^2$ $+d * b^2$ - 1) a separator can be found for the embedding feature space $(1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$ using the using polynomial kernel regardless of center, width, and orientation .

---

2)



if we have two circles this means we have the following equations :

$$z_1^2 = (x_1 - a)^2 + (x_2 - b)^2$$

$$x_1^2 + x_2^2 + 2 * x_1 * a + 2 * x_2 * b + a^2 + b^2 - z_1^2 = 0$$

$$z_2^2 = (x_3 - c)^2 + (x_4 - d)^2$$

$$x_3^2 + x_3^2 + 2 * x_3 * c + 2 * x_4 * d + c^2 + d^2 - z_2^2 = 0$$

let's assume for the first circle the inside is positive and the outside is negative, this means for the second one the inside and the outside are negative so we have F1 sign function for the first circle and F2 sign function for the second circle the sign function for the overall data will be :

$$F1 = sign(-1 * ((x_1 - a)^2 + (x_2 - b)^2) - z_1^2)$$

$$F2 = sign(-1 * ((x_3 - c)^2 + (x_4 - d)^2 - z_2^2)$$

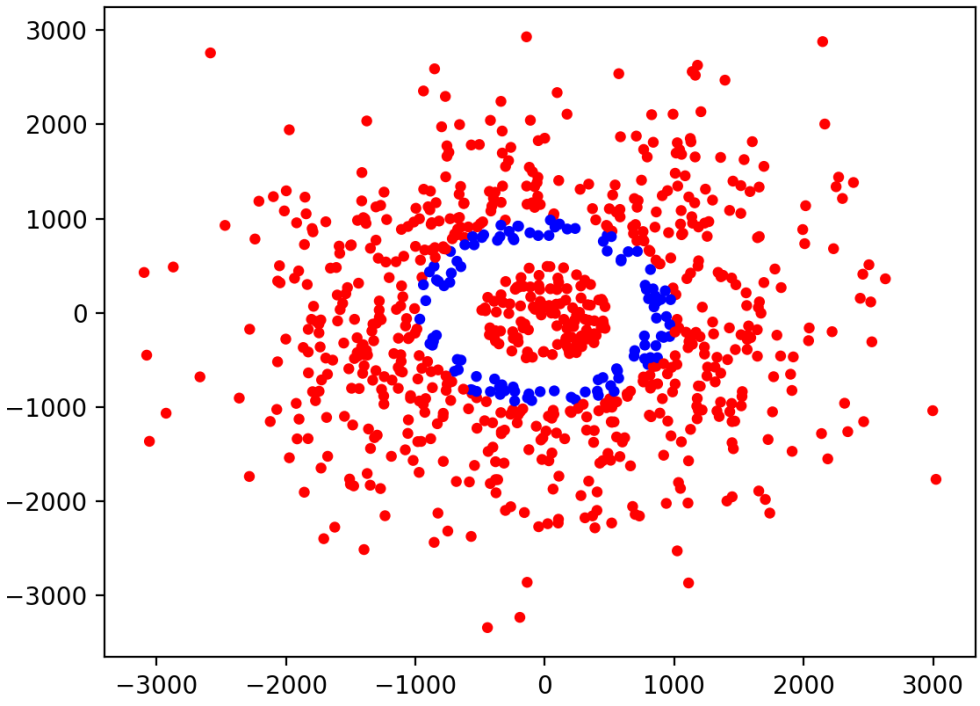
$$F3 = -1 * sign(F1) * sign(F2) = \begin{cases} -1, & \text{F1 = -1 and F2 = -1} \\ +1, & \text{F1 = 1 and F2 = -1} \\ +1, & \text{F1 = -1 and F2 = 1} \\ -1, & \text{F1 = 1 and F2 = 1} \end{cases}$$

the separator is a kernel function resulted of multiplying two polynomial function will give us

$$K(x, y) = (1 + x * y)^4$$

---

3)



Similarly here we have two circles :

$$z_1^2 = (x_1 - a)^2 + (x_2 - b)^2$$

$$x_1^2 + x_2^2 + 2 * x_1 * a + 2 * x_2 * b + a^2 + b^2 - z_1^2 = 0$$

$$z_2^2 = (x_3 - c)^2 + (x_4 - d)^2$$

$$x_3^2 + x_3^2 + 2 * x_3 * c + 2 * x_4 * d + c^2 + d^2 - z_2^2 = 0$$

$$F1 = sign(-1 * ((x_1 - a)^2 + (x_2 - b)^2) - z_1^2)$$
$$F2 = sign(-1 * ((x_3 - c)^2 + (x_4 - d)^2 - z_2^2)$$
$$F3 = sign(F1) * sign(F2) = \begin{cases} +1, & \text{F1 = 1 and F2 = 1} \\ -1, & \text{F1 = 1 and F2 = -1} \\ +1, & \text{F1 = -1 and F2 = -1} \end{cases}$$

the separator is a kernel function resulted of multiplying two polynomial function will give us

$$K(x, y) = (1 + x * y)^4$$

---

4)
The XOR data represented as :
X1: $(-1,+1),+1$
X2: $(-1,-1),-1$
X3: $(+1,-1),+1$
X4: $(+1,+1),-1$.
we construct x as : $\begin{bmatrix} -1 & -1 \\ -1 & +1 \\ +1 & -1 \\ +1 & +1 \end{bmatrix}$ and y as: $\begin{bmatrix} -1 \\ +1 \\ +1 \\ -1 \end{bmatrix}$
having the kernel function

$$K(x, y) = (1 + x * y)^2$$

and the dual svm :

$$\max_{\underline{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i y^i K(\underline{x}^i, \underline{x}^j) y^j \alpha_j,$$
$$\text{(s.t.)} \quad \sum_{i=1}^{m} \alpha_i y^i = 0$$
$$\forall i : \alpha_i \geq 0,$$

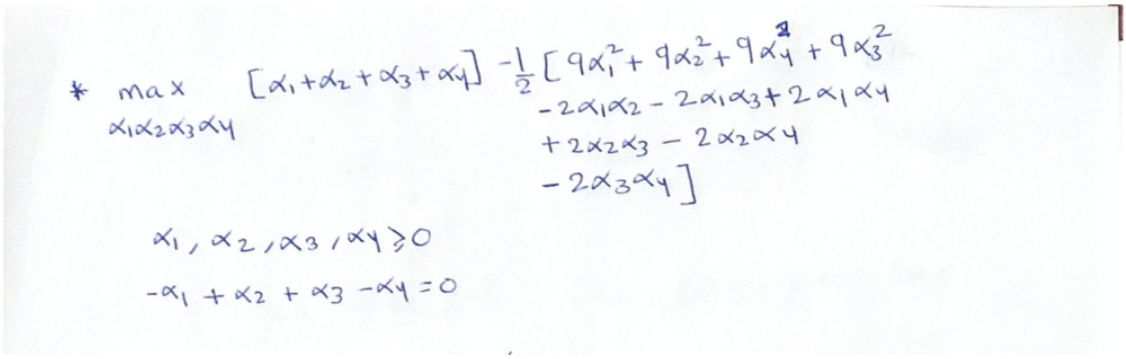
$$\text{classifier}(\underline{x}) = \text{sign}\left(\sum_i \alpha_i y^i K(\underline{x}^i, \underline{x}) + b \right).$$

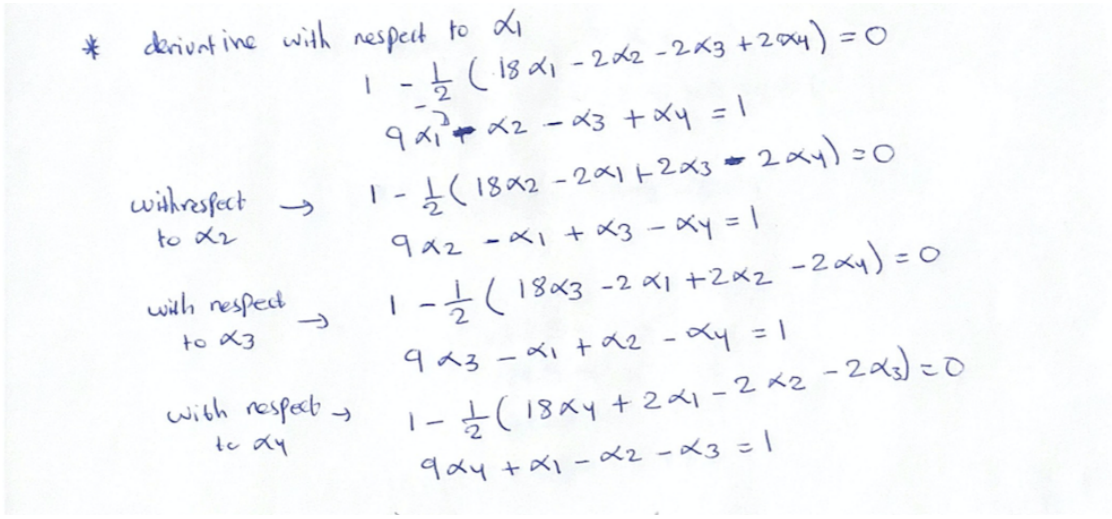
$$K1,1 = (1 + \begin{bmatrix} -1 & -1 \end{bmatrix} * \begin{bmatrix} -1 \\ -1 \end{bmatrix})^2 = 9$$
$$K1,2 = (1 + \begin{bmatrix} -1 & -1 \end{bmatrix} * \begin{bmatrix} -1 \\ +1 \end{bmatrix})^2 = 1$$
$$K1,3 = (1 + \begin{bmatrix} -1 & -1 \end{bmatrix} * \begin{bmatrix} +1 \\ -1 \end{bmatrix})^2 = 1$$

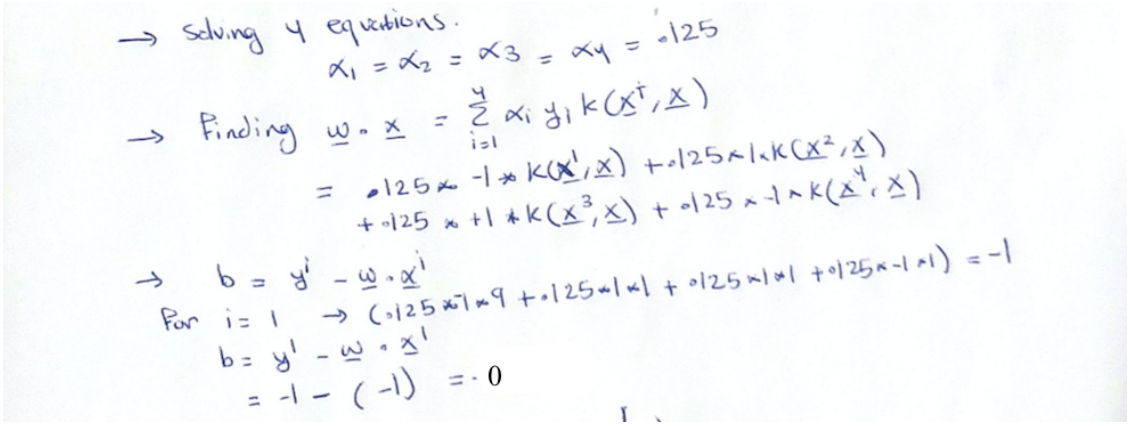
$$K1,4 = (1 + \begin{bmatrix} -1 & -1 \end{bmatrix} * \begin{bmatrix} +1 \\ +1 \end{bmatrix})^2 = 1$$

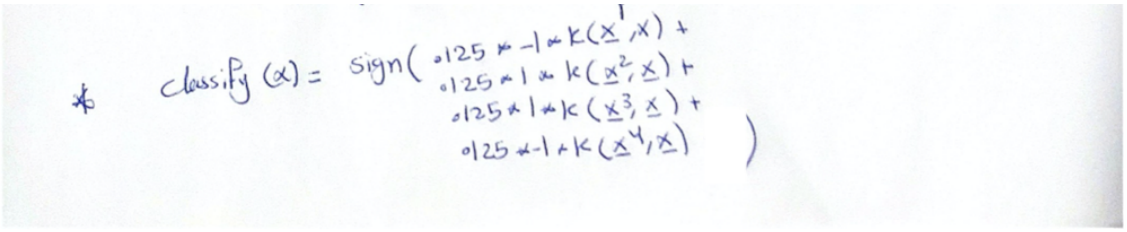
$$K1,1 * y1 * y1 = 9 * +1 * +1 = +9$$

$$K = \begin{bmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{bmatrix}$$









to find the region for (x1,x2):

* to find w: for the $\phi(x) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)^T$

$$w = \sum_{i=1}^{4} \alpha_i y_i \phi(x)$$

$$= 0.125 \cdot -1 \cdot \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + 0.125 \cdot 1 \cdot \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + 0.125 \cdot 1 \cdot \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ 1 \end{bmatrix} + 0.125 \cdot -1 \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

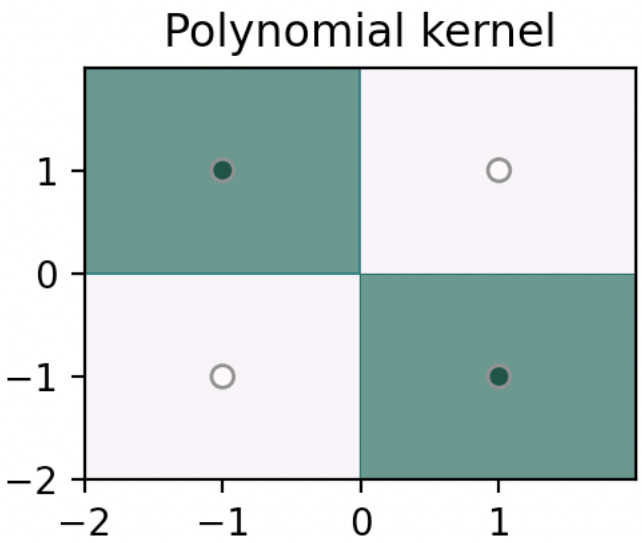
$$w = [0 \quad 0 \quad 0 \quad -0.5 \quad 0 \quad 0]^T$$

$$y(x) = w^T \phi(x)$$

$$= [0 \quad 0 \quad 0 \quad -0.5 \quad 0 \quad 0] \cdot \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

$y(x) = \text{sign}(-0.5\, x_1 x_2)$

↳ and this is the classifier identifying x_1 x_2 region



---

we construct x as : $\begin{bmatrix} -1 & -1 \\ -1 & +1 \\ +1 & -1 \\ +1 & +1 \end{bmatrix}$ and y as: $\begin{bmatrix} -1 \\ +1 \\ +1 \\ -1 \end{bmatrix}$

having the kernel function

$$K(x, y) = exp(-||x - y||^2)$$

and the dual svm :

$$\max_{\underline{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i y^i K(\underline{x}^i, \underline{x}^j) y^j \alpha_j,$$

$$\text{(s.t.)} \quad \sum_{i=1}^{m} \alpha_i y^i = 0$$

$$\forall i : \alpha_i \geq 0,$$

$$\text{classifier}(\underline{x}) = \text{sign}\left(\sum_i \alpha_i y^i K(\underline{x}^i, \underline{x}) + b \right).$$

$$K1,1 = (e^{-||\begin{bmatrix} -1 \\ -1 \end{bmatrix} - \begin{bmatrix} -1 \\ -1 \end{bmatrix}||^2}) = 1$$

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

* max $[\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4] - \frac{1}{2}[\alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2 - 2\alpha_1\alpha_2 e^{-4} - 2\alpha_1\alpha_3 e^{-4} + 2\alpha_1\alpha_4 e^{-8} + 2\alpha_2\alpha_3 e^{-8} - 2\alpha_2\alpha_4 e^{-4} - 2\alpha_3\alpha_4 e^{-4}]$

$\alpha_1, \alpha_2, \alpha_3, \alpha_4$

$-\alpha_1 + \alpha_2 + \alpha_3 - \alpha_4 = 0$

$\alpha_1, \alpha_2, \alpha_3, \alpha_4 \geq 0$

* derivative with respect to α_1

$$1 - \frac{1}{2}[2\alpha_1 - 2e^{-4}\alpha_2 - 2e^{-4}\alpha_3 + 2e^{-8}\alpha_4]$$

$$= 1 - [\alpha_1 - \alpha_2 e^{-4} - \alpha_3 e^{-4} + \alpha_4 e^{-8}]$$

$$\Rightarrow -\alpha_1 + \alpha_2 e^{-4} + \alpha_3 e^{-4} - \alpha_4 e^{-8} = -1$$

$$\alpha_1 - \alpha_2 e^{-4} - \alpha_3 e^{-4} + \alpha_4 e^{-8} = 1 \rightarrow (1)$$

$$-\alpha_1 e^{-4} + \alpha_2 + \alpha_3 e^{-8} - \alpha_4 e^{-4} = 1 \rightarrow (2)$$

$$-\alpha_1 e^{-4} + \alpha_2 e^{-8} + \alpha_3 - \alpha_4 e^{-4} = 1 \rightarrow (3)$$

$$\alpha_1 e^{-8} - \alpha_2 e^{-4} - \alpha_3 e^{-4} + \alpha_4 = 1 \rightarrow (4)$$

$\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 1.0376$

* Finding $\underline{w} \cdot \underline{x} = \sum_{i=1}^{4} \alpha_i y_i k(x, y)$

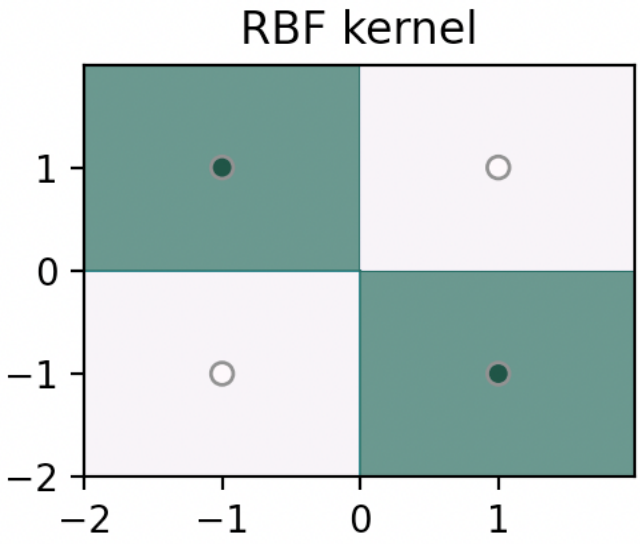
$= 1.0376\left(-k(\underline{x}^1, \underline{x}) + k(\underline{x}^2, \underline{x}) + k(\underline{x}^3, \underline{x}) + k(\underline{x}^4, \underline{x})\right)$

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

for $i=1 \rightarrow 1.0376(-1 + e^{-4} + e^{-4} + e^{-8}) = -.999 \approx -1$

$b = -1 - (-1) = 0$

* classify(x) = sign $\left(1.0376[-1 * k(\underline{x}^1, x) + k(\underline{x}^2, \underline{x}) + k(\underline{x}^3, \underline{x}) - k(\underline{x}^4, x)\right)$



for both kernels we found classifier for the xor problem with similar margins based on my calculations as b was zero but we prefer polynomial in this case as its simpler