1. Perceptrons

Answer 1.1

Importing libraries:

```
import numpy as np
from numpy import linalg as LA
import matplotlib.pyplot as plt
import pandas as pd
from timeit import default_timer as timer
import sys
sys.setrecursionlimit(10000)
%matplotlib inline
```

Helper functions:

```
# normalizing each vector i.e. each Z_i in Z
def normalize(Z):
    X=np.empty((np.shape(Z)[0],np.shape(Z)[1]))
    for i in range(np.shape(Z)[0]):
        norm = LA.norm(Z[i])
        if(norm==0):
            X[i]=Z[i]
            print("here at i: ",i)
        else:
            X[i]=Z[i]/norm
        # to check if norm/length == 1
        # print(LA.norm(X[i]))
    return X
```

```
# to normalize a single point i.e. 1D array
def normalize_single(vec):
    norm = LA.norm(vec)
    if norm == 0:
       return vec
    return vec / norm
```

```
# discard the vector/point and replace with new point which satisfies
abs(vec[k-1])>=ep
def X_i_replace(vec, ep):
    # given mean and variance
    mu=0
    sigma=1
    k=len(vec)
    new_vec=np.random.normal(mu,sigma,k)
    new_vec=normalize_single(new_vec)

    if(abs(new_vec[k-1])<ep): # k-1 is th kth index
        # recursively call itself until abs(vec[k-1])>=ep
        return X_i_replace(new_vec, ep)
        # print("new vec")
    else:
        # just checking if norm of new point == 1
        # print("new point created with norm: ",new_vec," ---
",LA.norm(new_vec))
        return new_vec
```

```
# normalizing/scaling weights and bias such that |w|^2 + b*b = 1
def normalize_w_and_b(w,b):
    #w2=np.dot(w,np.transpose(w))
    w2=np.dot(w,w)
    b2=b*b

    sum=w2+b2
    w_=w/pow(sum,0.5)
    b_=b/pow(sum,0.5)

    return w_,b_
```

Function to generate X and Y:

```
# gen X i.e. m datapoints with k features (k dimensions)
def gen_X(m,k,ep):
    pass
    # given mean and variance
    mu=0
    sigma=1
    # i.i.d standard normal data
    Z=np.random.normal(mu,sigma,size=(m,k))

    # normalizing each vector i.e. each Z_i in Z
    X=normalize(Z)

    # check if absolute value of X_i_k < epsilon for all X_i
    for i in range(m):
        if(abs(X[i][k-1])<ep): # k-1 is th kth index
            # discard the vector/point and replace with new point which
satisfies the above condition
            # print(i)
            X[i]=X_i_replace(X[i],ep)
    return X
```

```
# gen Y/target/labels/output
def gen_Y(X,ep):
    pass
    m=np.shape(X)[0]
    k=np.shape(X)[1]
    Y=np.empty((m)).astype(int)
    for i in range(m):
        if(X[i][k-1]>=ep):
            Y[i]=1
        elif(X[i][k-1]<=-ep):
            Y[i]=-1
    return Y
```

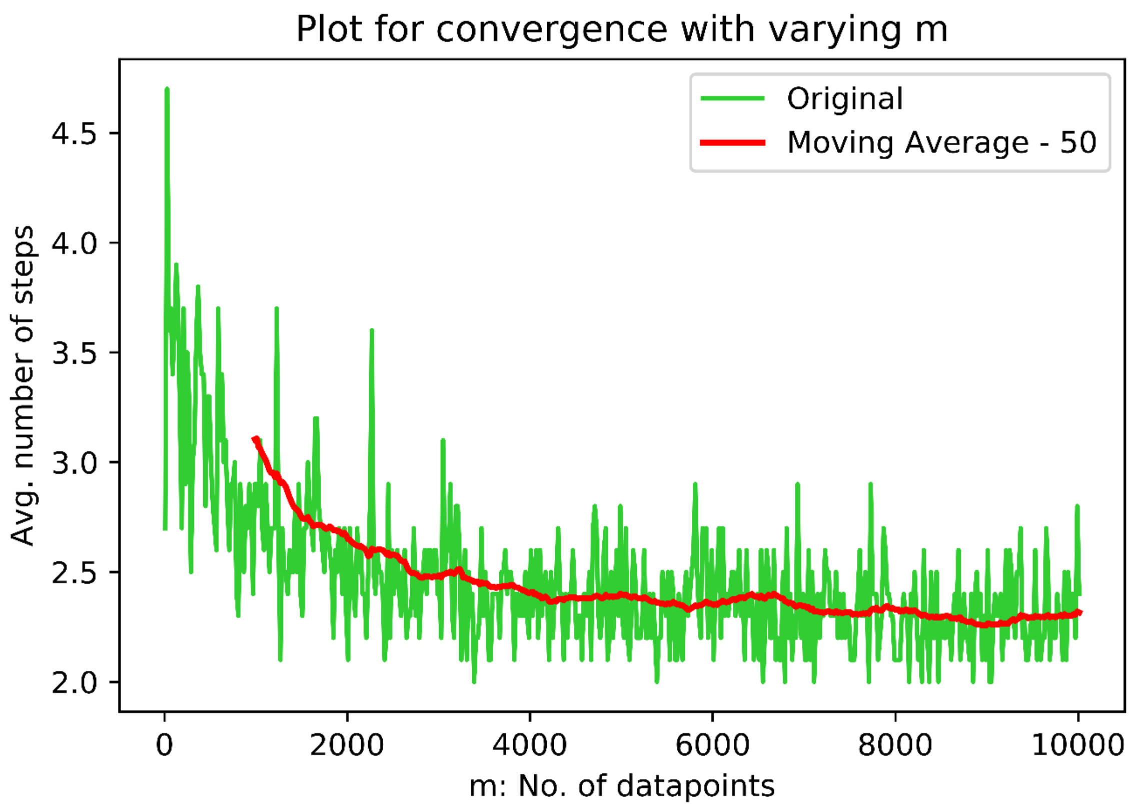
Function - Perceptron Learning Algorithm (PLA):

```
# returns steps to converge, "typical" weights and "typical" bias
def PLA(X,Y,MAX_ITER=10000):
    # init wts and bias
    wt=np.zeros((np.shape(X)[1]))
    b=0
    steps_to_converge=0

    while(True):
        steps_to_converge+=1
        misclassified=0
        for X_i, Y_i in zip(X,Y):
            linear_op=np.dot(X_i,wt)+b
            if(linear_op>0):
                f_x=1.0
            else:
                f_x=-1.0

            # if misclassified
            if(f_x!=Y_i):
                misclassified+=1

                # update step
                wt+=np.dot(X_i,Y_i)
                b+=Y_i

        if(misclassified==0):
            return steps_to_converge, wt, b

        if(steps_to_converge>MAX_ITER):
            #breaking
            return steps_to_converge, wt, b
```

Answer 1:

```
start=timer()

# init parameters
k=10
ep=0.1
m=list(range(10,10011,20))

# number of iterations to calculate the average
avg_iter=10

print("m values (List) =",m)
avg_steps_=[]

# running the sim
for m_i in m:
    # init avg number of steps to converge
    avg_steps=0
    for i in range(avg_iter):
        # generate X and Y
        X=gen_X(m_i,k,ep)
        Y=gen_Y(X,ep)

        # get number of steps to converge from PLA function
        steps_to_converge, _, _ = PLA(X,Y)

        # summing up number of steps to converge
        avg_steps+=steps_to_converge
    # averaging number of steps to converge
    avg_steps/=avg_iter

    # add to a list to plot
    avg_steps_.append(avg_steps)

# converting to pandas dataframe to get the
# rolling mean (moving average) for better analysis
df_avg_steps = pd.DataFrame(avg_steps_)
rolling_mean_steps = df_avg_steps.rolling(window=50).mean()

# plotting
plt.plot(m, avg_steps_, label="Original", color="limegreen")
plt.plot(m, rolling_mean_steps, label="Moving Average - 50", color="red",
linewidth="2")
plt.xlabel("m: No. of datapoints")
plt.ylabel("Avg. number of steps")
plt.legend(loc="upper right")
plt.title("Plot for convergence with varying m")
plt.savefig('Q1_fig.png',dpi=1200)
plt.show()

print("Time taken: ", timer()-start)
```

```
m values (List) = [10, 30, 50, 70, …, 9930, 9950, 9970, 9990, 10010]
```



```
Time taken:  624.5963182
```

For the above plot and following plots, the red line indicates a moving average with the mentioned window size.

From the above plot, we can say that although initially (upto m=4000) it seems that the number of steps to converge is dependent on the number of datapoints, but on a larger scale, the number of steps for convergence does not depend on the number of datapoints (m). Also from the formula $\boldsymbol{T} \leq \frac{4}{\gamma^2(\underline{w},b)}$ (given in the notes), we can confirm the same. There is no dependency on m or k, but only on the margin ($\boldsymbol{\gamma}$).

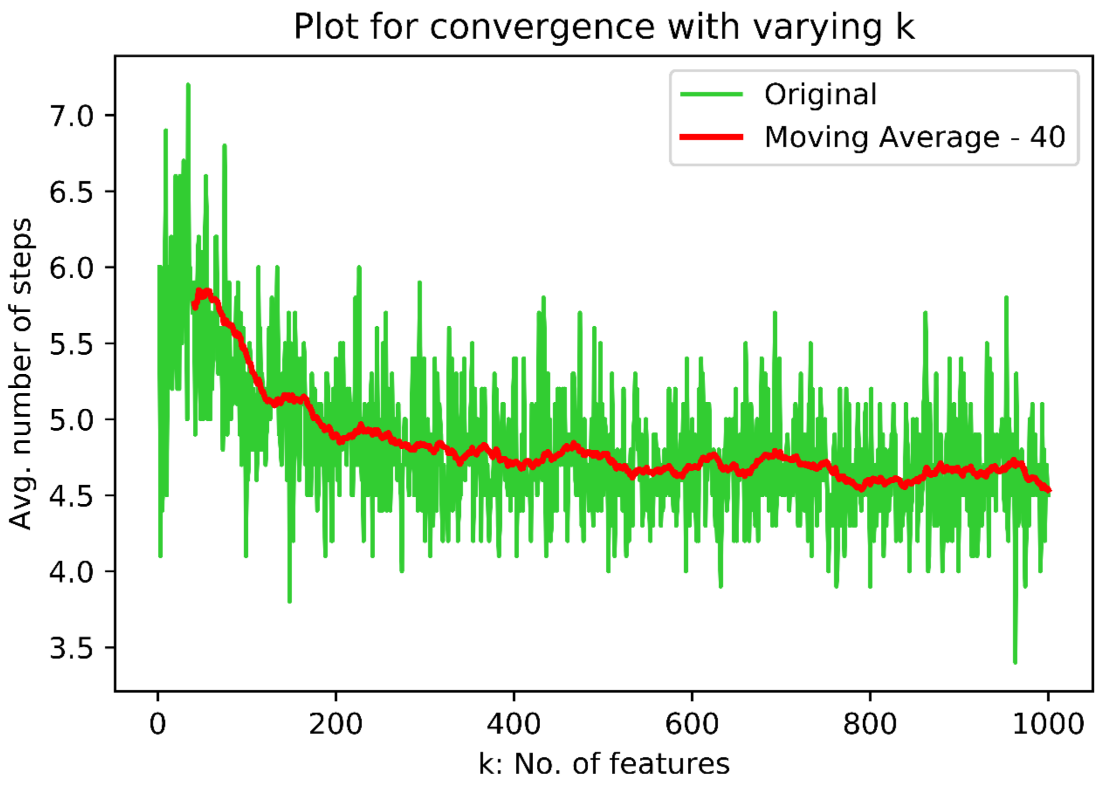
The results make sense since it agrees with the mathematical formulation.

Larger and larger m values will not produce anything different.

Answer 2:

```
start=timer()

# init parameters
k=list(range(2,1001,1))
ep=0.05
m=100

# number of iterations to calculate the average
avg_iter=10


print("k values (List) =",k)
avg_steps_=[]

# running the sim
for k_i in k:
    # init avg number of steps to converge
    avg_steps=0
    for i in range(avg_iter):
        # generate X and Y
        X=gen_X(m,k_i,ep)
        Y=gen_Y(X,ep)

        # get number of steps to converge from PLA function
        steps_to_converge, _, _ = PLA(X,Y)

        # summing up number of steps to converge
        avg_steps+=steps_to_converge

    # averaging number of steps to converge
    avg_steps/=avg_iter

    # add to a list to plot
    avg_steps_.append(avg_steps)

# converting to pandas dataframe to get the
# rolling mean (moving average) for better analysis
df_avg_steps = pd.DataFrame(avg_steps_)
rolling_mean_steps = df_avg_steps.rolling(window=40).mean()

# plotting
plt.plot(k, avg_steps_, label="Original", color="limegreen")
plt.plot(k, rolling_mean_steps, label="Moving Average - 40", color="red",
linewidth="2")
plt.xlabel("k: No. of features")
plt.ylabel("Avg. number of steps")
plt.legend(loc="upper right")
plt.title("Plot for convergence with varying k")
plt.savefig('Q2_fig.png',dpi=1200)
plt.show()

print("Time taken: ", timer()-start)
```

```
k values (List) = [2, 3, 4, 5, …, 995, 996, 997, 998, 999, 1000]
```



```
Time taken:  173.78885200000002
```

From the above plot, we can say that although initially (upto k=200) it seems that the number of steps to converge is dependent on the number of dimensions/features, but on a larger scale, the number of steps for convergence does not depend on the number of dimensions /features (k). Also from the formula $\boldsymbol{T} \leq \frac{\boldsymbol{4}}{\boldsymbol{\gamma}^2(\underline{\boldsymbol{w}},\boldsymbol{b})}$ (given in the notes), we can confirm the same. There is no dependency on m or k, but only on the margin ($\boldsymbol{\gamma}$).

The results make sense since it agrees with the mathematical formulation.

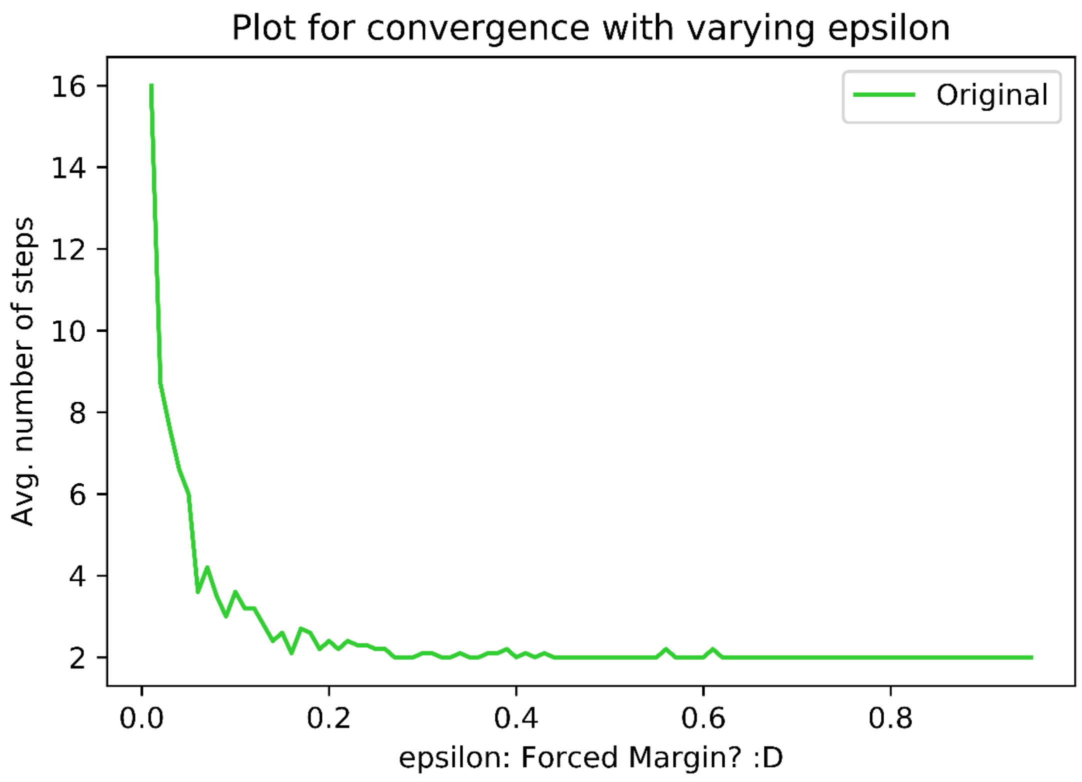
Larger and larger k values will not produce anything different.

Answer 3:

```
start=timer()

# init parameters
k=5
ep=np.arange(0.01,0.96,0.01)
m=100

# number of iterations to calculate the average
avg_iter=10

print("epsilon values (List) =",ep)
avg_steps_=[]
```

```
# running the sim
for ep_i in ep:
    # init avg number of steps to converge
    avg_steps=0
    for i in range(avg_iter):
        # generate X and Y
        X=gen_X(m,k,ep_i)
        Y=gen_Y(X,ep_i)

        # get number of steps to converge from PLA function
        steps_to_converge, _, _ = PLA(X,Y)

        # summing up number of steps to converge
        avg_steps+=steps_to_converge

    # averaging number of steps to converge
    avg_steps/=avg_iter

    # add to a list to plot
    avg_steps_.append(avg_steps)

#df_avg_steps = pd.DataFrame(avg_steps_)
#rolling_mean_steps = df_avg_steps.rolling(window=3).mean()

# plotting
plt.plot(ep, avg_steps_, label="Original", color="limegreen")
#plt.plot(ep, rolling_mean_steps, label="Moving Average - 3", color="red",
linewidth="2", linestyle="dashed")
plt.xlabel("epsilon: Forced Margin? :D")
plt.ylabel("Avg. number of steps")
plt.legend(loc="upper right")
plt.title("Plot for convergence with varying epsilon")
plt.savefig('Q3_fig.png',dpi=1200)
plt.show()

print("Time taken: ", timer()-start)
```

```
epsilon values (List) = [0.01 0.02 0.03 0.04 0.05 … 0.92 0.93 0.94 0.95]
```



```
Time taken:  27.150961800000005
```

From the above plot, we can say that the number of steps for convergence depends on the value of epsilon. Epsilon here acts as a kind of forced margin.
Hence from the formula $\boldsymbol{T} \leq \frac{\mathbf{4}}{\gamma^2(\underline{\boldsymbol{w}},\boldsymbol{b})}$ (given in the notes), we can confirm the same. There is no dependency on m or k, but only on the margin ($\boldsymbol{\gamma}$).

The results make sense since from our intuition we can also imagine that more is the margin (forced), easier it is for the linear separator to be found. For smaller values of epsilon, the linear separator only exists within a small region and hence takes more steps to converge.

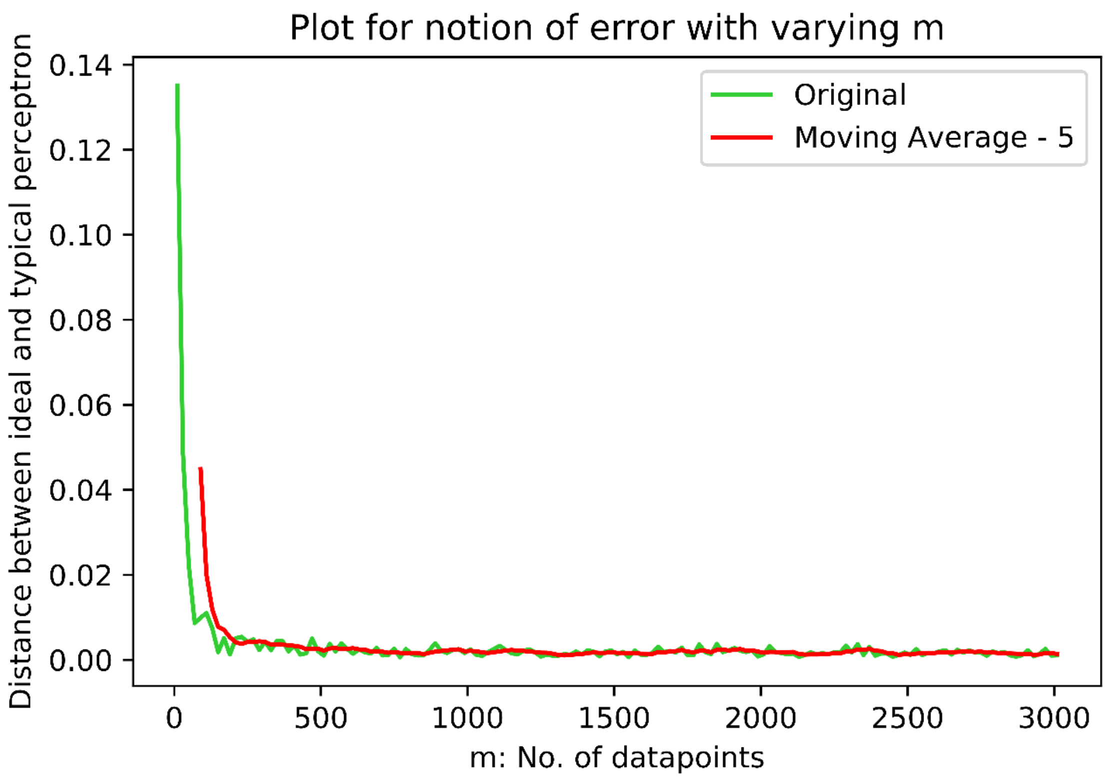
Looking at different k and m (except for small values of k and m) will not produce anything different.

Bonus Answer Part 1:

```
start=timer()

# init parameters
k=10
ep=0.1
m=list(range(10,3011,20))

# number of iterations to calculate the average
avg_iter=10

# init ideal weights and bias
wt_ideal=np.zeros((k))
wt_ideal[-1]=1
b_ideal=0
```

```python
print("m values (List) =",m)
ordinate_val_list=[]

# running the sim
for m_i in m:
    # init typical average wt and bias
    avg_wt_typ=np.zeros((k))
    avg_b_typ=0

    for i in range(avg_iter):
        # generate X and Y
        X=gen_X(m_i,k,ep)
        Y=gen_Y(X,ep)

        # get typical wt and bias from PLA function
        _, wt_typical, b_typical = PLA(X,Y)

        # summing up wts and bias
        avg_wt_typ+=wt_typical
        avg_b_typ+=b_typical

    # averaging wts and bias
    avg_wt_typ/=avg_iter
    avg_b_typ/=avg_iter

    # normalizing/scaling weights and bias such that |w|^2 + b^2 = 1
    avg_wt_typ, avg_b_typ = normalize_w_and_b(avg_wt_typ,avg_b_typ)
    wt_ideal, b_ideal = normalize_w_and_b(wt_ideal,b_ideal)

    # calculating the value of the given equation in steps
    wt_diff=wt_ideal-avg_wt_typ
    b_diff=b_ideal-avg_b_typ
    val=pow(LA.norm(wt_diff),2)+pow(b_diff,2)

    # add to a list to plot
    ordinate_val_list.append(val)

# converting to pandas dataframe to get the
# rolling mean (moving average) for better analysis
df_val = pd.DataFrame(ordinate_val_list)
rolling_mean_steps = df_val.rolling(window=5).mean()

# plotting
plt.plot(m, ordinate_val_list, label="Original", color="limegreen")
plt.plot(m, rolling_mean_steps, label="Moving Average - 5", color="red") #,
linestyle="dashed")
plt.xlabel("m: No. of datapoints")
plt.ylabel("Distance between ideal and typical perceptron")
plt.legend(loc="upper right")
plt.title("Plot for notion of error with varying m")
plt.savefig('Bonus_1_fig.png',dpi=1200)
plt.show()

print("Time taken: ", timer()-start)
```

```
m values (List) = [10, 30, 50, 70, 90, …, 2930, 2950, 2970, 2990, 3010]
```



```
Time taken:  57.121473899999955
```

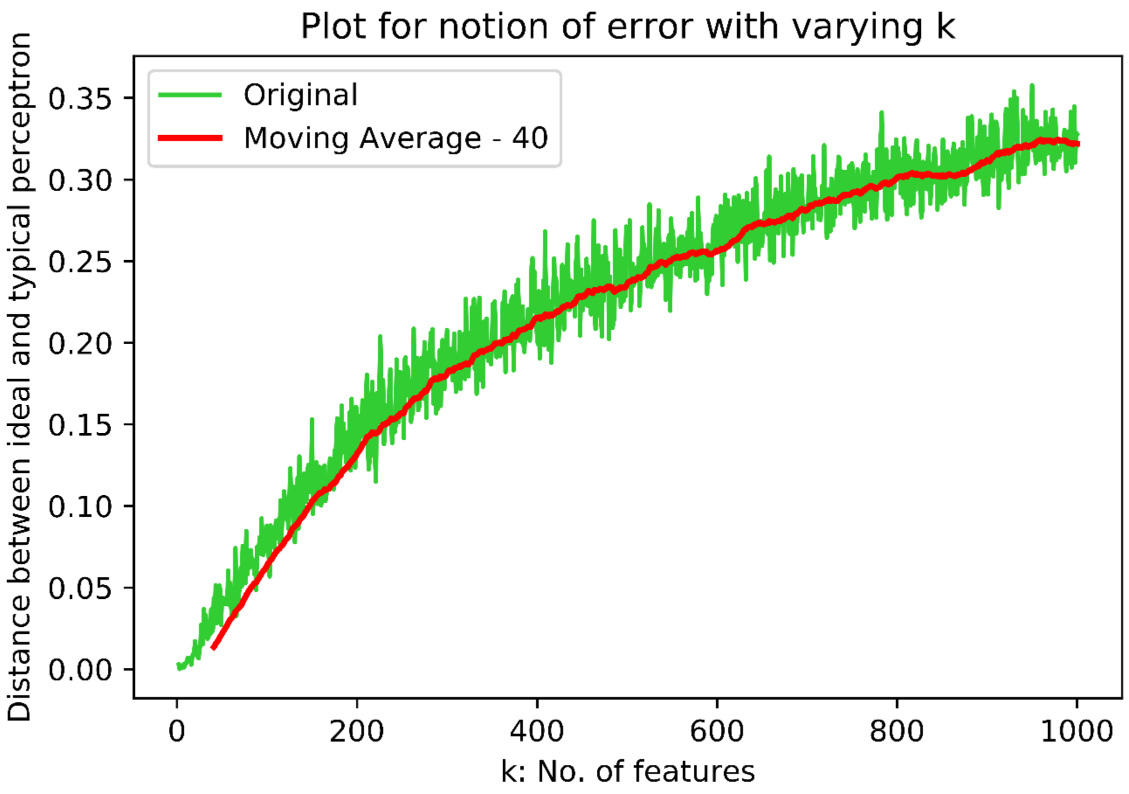
This plot makes sense since if we add more data points, the new data points will tend to occupy all allowed space (on the sphere's surface except the margin/epsilon) and the perceptron will have less space move around (like being crunched by data points) and we prepared the data in such a way such that there exists a linear separator. So the averaged typical perceptron will get very close to the ideal perceptron as we get in more and more data.

NB: Since epsilon is small it quickly reaches an error close to 0, but I tested with epsilon=0.5, and it didn't converge (at least till m~3000)

Bonus Answer Part 2:

```
start=timer()

# init parameters
k=list(range(2,1001,1))
ep=0.05
m=100

# number of iterations to calculate the average
avg_iter=10

# init ideal bias
b_ideal=0

print("k values (List) =",k)
ordinate_val_list=[]

# running the sim
for k_i in k:
    # init typical average wt and bias
    avg_wt_typ=np.zeros((k_i))
    avg_b_typ=0

    # init ideal weights
    wt_ideal=np.zeros((k_i))
    wt_ideal[-1]=1

    for i in range(avg_iter):
        # generate X and Y
        X=gen_X(m,k_i,ep)
        Y=gen_Y(X,ep)

        # get typical wt and bias from PLA function
        _, wt_typical, b_typical = PLA(X,Y)

        # summing up wts and bias
        avg_wt_typ+=wt_typical
        avg_b_typ+=b_typical

    # averaging wts and bias
    avg_wt_typ/=avg_iter
    avg_b_typ/=avg_iter

    # normalizing/scaling weights and bias such that |w|^2 + b^2 = 1
    avg_wt_typ, avg_b_typ = normalize_w_and_b(avg_wt_typ,avg_b_typ)
    wt_ideal, b_ideal = normalize_w_and_b(wt_ideal,b_ideal)

    # calculating the value of the given equation in steps
    wt_diff=wt_ideal-avg_wt_typ
    b_diff=b_ideal-avg_b_typ
    val=pow(LA.norm(wt_diff),2)+pow(b_diff,2)

    # add to a list to plot
    ordinate_val_list.append(val)

# converting to pandas dataframe to get the
# rolling mean (moving average) for better analysis
df_val = pd.DataFrame(ordinate_val_list)
rolling_mean_steps = df_val.rolling(window=40).mean()


# plotting
plt.plot(k, ordinate_val_list, label="Original", color="limegreen")
plt.plot(k, rolling_mean_steps, label="Moving Average - 40", color="red",
linewidth="2")#, linestyle="dashed")
plt.xlabel("k: No. of features")
plt.ylabel("Distance between ideal and typical perceptron")
plt.legend(loc="upper left")
plt.title("Plot for notion of error with varying k")
plt.savefig('Bonus_2_fig.png',dpi=1200)
plt.show()

print("Time taken: ", timer()-start)
```

```
k values (List) = [2, 3, 4, 5, 6, …, 994, 995, 996, 997, 998, 999, 1000]
```



```
Time taken:  176.35292900000013
```

This plot makes sense since if we add more dimensions to the data, there are more possible perceptrons that can exist and hence more space for error.

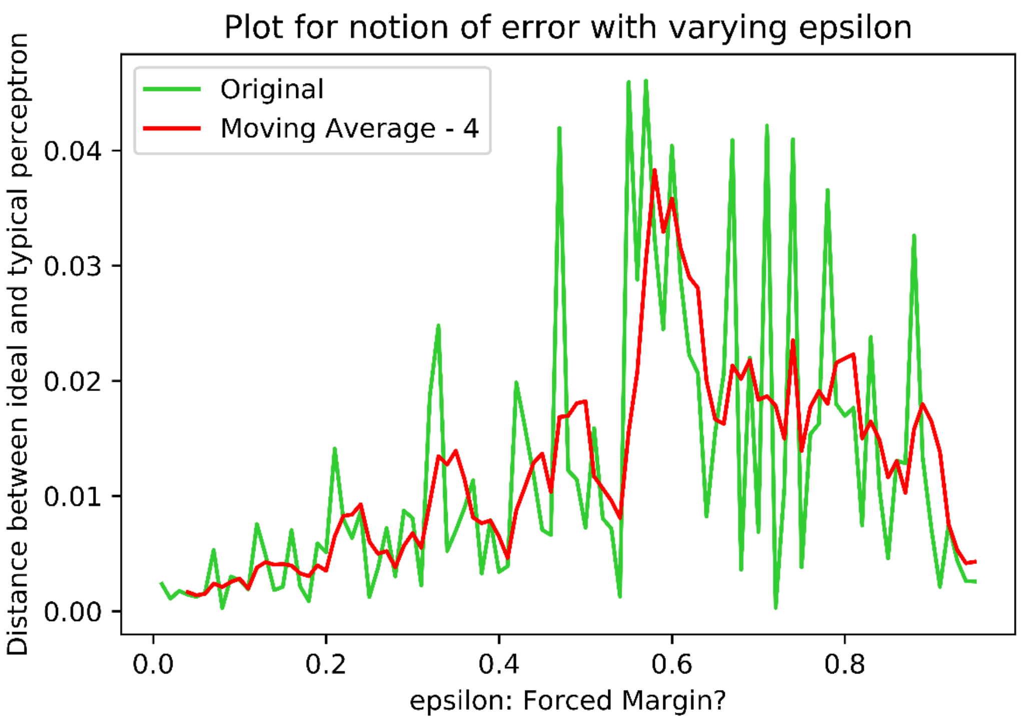
We can visualize this somewhat: In 2D, the line can rotate in only one dimension, but in 3D, the plane can rotate in 2 dimensions. So as we go up in dimentionality (features), the hyperplane (the linear separator) will have more space to move around.

Bonus Answer Part 3:

```
start=timer()

# init parameters
k=5
ep=np.arange(0.01,0.96,0.01)
m=100

# number of iterations to calculate the average
avg_iter=10

# init ideal weights and bias
wt_ideal=np.zeros((k))
wt_ideal[-1]=1
b_ideal=0

print("epsilon values (List) =",ep)
ordinate_val_list=[]
```

```
# running the sim
for ep_i in ep:
    # init typical average wt and bias
    avg_wt_typ=np.zeros((k))
    avg_b_typ=0

    for i in range(avg_iter):
        # generate X and Y
        X=gen_X(m,k,ep_i)
        Y=gen_Y(X,ep_i)

        # get typical wt and bias from PLA function
        _, wt_typical, b_typical = PLA(X,Y)

        # summing up wts and bias
        avg_wt_typ+=wt_typical
        avg_b_typ+=b_typical

    # averaging wts and bias
    avg_wt_typ/=avg_iter
    avg_b_typ/=avg_iter

    # normalizing/scaling weights and bias such that |w|^2 + b^2 = 1
    avg_wt_typ, avg_b_typ = normalize_w_and_b(avg_wt_typ,avg_b_typ)
    wt_ideal, b_ideal = normalize_w_and_b(wt_ideal,b_ideal)

    # calculating the value of the given equation in steps
    wt_diff=wt_ideal-avg_wt_typ
    b_diff=b_ideal-avg_b_typ
    val=pow(LA.norm(wt_diff),2)+pow(b_diff,2)

    # add to a list to plot
    ordinate_val_list.append(val)

# converting to pandas dataframe to get the
# rolling mean (moving average) for better analysis
df_val = pd.DataFrame(ordinate_val_list)
rolling_mean_steps = df_val.rolling(window=4).mean()


# plotting
plt.plot(ep, ordinate_val_list, label="Original", color="limegreen")
plt.plot(ep, rolling_mean_steps, label="Moving Average - 4", color="red")
#, linestyle="dashed")
plt.xlabel("epsilon: Forced Margin?")
plt.ylabel("Distance between ideal and typical perceptron")
plt.legend(loc="upper left")
plt.title("Plot for notion of error with varying epsilon")
plt.savefig('Bonus_3_fig.png',dpi=1200)
plt.show()

print("Time taken: ", timer()-start)
```

```
epsilon values (List) = [0.01 0.02 0.03 0.04 0.05 0.06 …  0.91 0.92 0.93
0.94 0.95]
```



```
Time taken:  23.876258000000007
```

This plot doesn't make sense, as my intuition says error should increase with increasing epsilon. The typical perceptron gets more available space as we increase epsilon and more chance to stray away from the ideal perceptron. Although the error increases till epsilon ~ 0.6, but on further increasing epsilon, it tends to decrease which is weird.

One guess is that it has something to do with probability. Kind of like the gaussian/normal distribution where most of the data is near the mean. The perceptron has space to move in the upper as well as the lower "hemisphere" (hyper hemisphere?) but maybe stays near the "equator" with higher probability.

Another guess is that the perceptron has more freedom to rotate, so in a way although it will misclassify a lot compared to the ideal perceptron, but perform better than a perceptron which is horizontal (kind of) and far away from the ideal perceptron.

2. SVMs

Answer 2.1a

Let the radius be ‘r’
Let the center of the be (a, b) [Given its 2D, so 2 coordinates]

The linear separator would be a circle of radius ‘r’ with center at (a, b).

Equation of the circle would be

$$(x_1 - a)^2 + (x_2 - b)^2 = r^2 \tag{1}$$

Anything outside the circle would be classified as negative and anything inside the circle would be classified as positive.

So, we can classify any datapoint (vector) $\underline{x}$ as follows:

$$classify\left(\underline{x}\right) = sign\left(r^2 - (x_1 - a)^2 - (x_2 - b)^2\right)$$

On expanding (1), we get:

$$x_1{}^2 - 2ax_1 + a^2 + x_2{}^2 - 2bx_2 + b^2 = r^2$$

or,

$$r^2 - x_1{}^2 + 2ax_1 - a^2 - x_2{}^2 + 2bx_2 - b^2 = 0$$

or,

$$r^2 - a^2 - b^2 + 2ax_1 + 2bx_2 - x_1{}^2 - x_2{}^2 = 0$$

or,

$$(r^2 - a^2 - b^2) + (2a)x_1 + (2b)x_2 + (0)x_1x_2 + (-1)x_1{}^2 + (-1)x_2{}^2 = 0$$

The above is of the form which can be represented by the given feature map: $\varphi(x_1, x_2) = (1, x_1, x_2, x_1x_2, x_1{}^2, x_2{}^2)$

Thus we see that a linear separator can always be found in the given embedded space.

Answer 2.1b

Let the number of dimensions be ‘k’
Let the center of the be (c_1, c_2, …, c_k)
Let the width along each axis be (w_1, w_2, …, w_k)

Equation of the ellipsoid (k dimensional) would be

$$\frac{(x_1-c_1)^2}{w_1{}^2} + \frac{(x_2-c_2)^2}{w_2{}^2} + \cdots + \frac{(x_k-c_k)^2}{w_k{}^2} = 1 \tag{2}$$

Anything outside the ellipsoid would be classified as negative and anything inside the ellipsoid would be classified as positive.

So, we can classify any datapoint (vector) $\underline{x}$ as follows:

$$classify\left(\underline{x}\right) = sign\left(1 - \frac{(x_1 - c_1)^2}{w_1{}^2} - \frac{(x_2 - c_2)^2}{w_2{}^2} - \cdots - \frac{(x_k - c_k)^2}{w_k{}^2}\right)$$

On expanding (2), we get:

$$\frac{\prod_{i=1}^{k} w_i^{\,2}}{w_1^{\,2}}(x_1^{\,2} - 2c_1x_1 - c_1^{\,2}) + \frac{\prod_{i=1}^{k} w_i^{\,2}}{w_2^{\,2}}(x_2^{\,2} - 2c_2x_2 - c_2^{\,2}) + \cdots + \frac{\prod_{i=1}^{k} w_i^{\,2}}{w_k^{\,2}}(x_k^{\,2} - 2c_kx_k - c_k^{\,2}) = 1$$

The math is long, but we can follow the same procedure as in 2.1a and show that the separator lies in the quadratic feature space.

Also, I ignored orientation as those would bring thetas (θ) but $\sin(\theta_1)$ *and* $\cos(\theta_2)$ are real-valued coefficients and can be absorbed.

This is possible because there is no term $x_1x_2 \ldots x_k$, so it always remains in the quadratic feature space.

Answer 2.2

Based on the wording in the question, I thought the positive class would be in ***either one*** *of the two disjoint ellipses, but based on the clarification on the discussion board, I'm considering both ellipses to contain the positive class (and the negative class everywhere else).*

Let the first ellipse be A and the second ellipse be B.
Let the center of A be (h_A, k_A) and that of B be (h_B, k_B).
Let the width along each axis of A be (a_A, b_A) and that of B be (a_B, b_B).

Equation of ellipse A:

$$\frac{(x_1 - h_A)^2}{a_A^{\,2}} + \frac{(x_2 - k_A)^2}{b_A^{\,2}} = 1$$

or,

$$b_A^{\,2}(x_1 - h_A)^2 + a_A^{\,2}(x_2 - k_A)^2 = a_A^{\,2}b_A^{\,2}$$

or,

$$a_A^{\,2}b_A^{\,2} - b_A^{\,2}(x_1 - h_A)^2 - a_A^{\,2}(x_2 - k_A)^2 = 0$$

Let $\boldsymbol{E_1 = a_A^{\,2}b_A^{\,2} - b_A^{\,2}(x_1 - h_A)^2 - a_A^{\,2}(x_2 - k_A)^2}$

Similarly, the equation of ellipse B:

$$a_B^{\,2}b_B^{\,2} - b_B^{\,2}(x_1 - h_B)^2 - a_B^{\,2}(x_2 - k_B)^2 = 0$$

Let $\boldsymbol{E_2 = a_B^{\,2}b_B^{\,2} - b_B^{\,2}(x_1 - h_B)^2 - a_B^{\,2}(x_2 - k_B)^2}$

So, we can classify any datapoint (vector) $\underline{x}$ as follows:

$$classify\ (\underline{x}) = sign(-\boldsymbol{E_1} * \boldsymbol{E_2})$$

From Answer 1, we confirmed that be it a circle or an ellipse, the separator follows the same idea/logic (just more coefficient terms in the case of ellipses). So I tested it out on paper by trying different test cases on two disjoint (non-concentric) circles to find the resultant sign.

$$\boldsymbol{E_1 * E_2 = [a_A^{\,2}b_A^{\,2} - b_A^{\,2}(x_1 - h_A)^2 - a_A^{\,2}(x_2 - k_A)^2] * [a_B^{\,2}b_B^{\,2} - b_B^{\,2}(x_1 - h_B)^2 - a_B^{\,2}(x_2 - k_B)^2]}$$

or (absorbing/replacing variables which are not $\boldsymbol{x_1}$ ***or*** $\boldsymbol{x_2}$ as C's,

$$\boldsymbol{E_1 * E_2 = \ C_1 - C_2(x_1 - C_3)^2 - C_4(x_2 - C_5)^2] * [C_6 - C_7(x_1 - C_8)^2 - C_9(x_2 - C_{10})^2]}$$

or (absorbing/replacing variables which are not $\boldsymbol{x_1}$ $\boldsymbol{or}$ $\boldsymbol{x_2}$ as D's,

$$\boldsymbol{E_1 * E_2 = [C_1 - C_2{x_1}^2 + D_1x_1 - D_2 - C_4{x_2}^2 + D_3x_2 - D_4] * [C_6 - C_7{x_1}^2 + D_5x_1 - D_6 - C_9{x_2}^2 + D_7x_2 - D_8]}$$

or (absorbing/replacing variables which are not $\boldsymbol{x_1}$ $\boldsymbol{or}$ $\boldsymbol{x_2}$ as F's,

$$\boldsymbol{E_1 * E_2 = [F_1 - F_2{x_1}^2 + F_3x_1 - F_4{x_2}^2 + F_5x_2] * [F_6 - F_7{x_1}^2 + F_8x_1 - F_9{x_2}^2 + F_{10}x_2]}$$

or (absorbing/replacing variables which are not $\boldsymbol{x_1}$ $\boldsymbol{or}$ $\boldsymbol{x_2}$ as G's,

$$\boldsymbol{\begin{aligned} E_1 * E_2 = [&(G_1 - G_2{x_1}^2 + G_3x_1 - G_4{x_2}^2 + G_5x_2) + (-G_6{x_1}^2 + G_7{x_1}^4 - G_8{x_1}^3 + G_9{x_1}^2{x_2}^2 - G_{10}{x_1}^2x_2) \\ &+ (G_{11}x_1 - G_{12}{x_1}^3 + G_{13}{x_1}^2 - G_{14}x_1{x_2}^2 + G_{15}x_1x_2) \\ &+ (-G_{16}{x_2}^2 + G_{17}{x_1}^2{x_2}^2 - G_{18}x_1{x_2}^2 + G_{19}{x_2}^4 - G_{20}{x_2}^3) \\ &+ (G_{21}x_2 - G_{22}{x_1}^2x_2 + G_{23}x_1x_2 - G_{24}{x_2}^3 + G_{25}{x_2}^2)] \end{aligned}}$$

The above can be condensed further by taking common terms out of x_1 and x_2. Then it is of the form which can be represented by the feature map:

$$\varphi(x_1, x_2) = (1, x_1, x_2, x_1x_2, {x_1}^2, {x_2}^2, {x_1}^2x_2, x_1{x_2}^2, {x_1}^3, {x_2}^3, {x_1}^3x_2, x_1{x_2}^2, {x_1}^2{x_2}^2)$$

or equivalently with the kernel:

$$\boldsymbol{K(x_1, x_2) = (1 + x_1.x_2)^4}$$

Thus, the kernel $\boldsymbol{K(x, y) = (1 + x.y)^4}$ will recover a separator.

Answer 2.3

There are 2 concentric circles.
Let the smaller circle be A and the larger circle be B.
Let both of them be centered at (a,b)
Let the radius of the smaller circle be 'r_A'.
Let the radius of the larger circle be 'r_B'.

Equation of circle A:

$$(x_1 - a)^2 + (x_2 - b)^2 = {r_A}^2$$

or,

$${r_A}^2 - (x_1 - a)^2 - (x_2 - b)^2 = 0$$

Let $\boldsymbol{E_1 = {r_A}^2 - (x_1 - a)^2 - (x_2 - b)^2}$

Similarly, the equation of circle B:

$${r_B}^2 - (x_1 - a)^2 - (x_2 - b)^2 = 0$$

Let $\boldsymbol{E_2 = {r_B}^2 - (x_1 - a)^2 - (x_2 - b)^2}$

So, we can classify any datapoint (vector) $\underline{x}$ as follows:

$$classify\,(\underline{x}) = sign(\boldsymbol{E_1 * E_2})$$

I tested it out on paper by trying different test cases on two concentric circles to find the resultant sign.

$$\boldsymbol{E_1 * E_2 = \left[{r_A}^2 - (x_1 - a)^2 - (x_2 - b)^2\right] * \left[{r_B}^2 - (x_1 - a)^2 - (x_2 - b)^2\right]}$$

or (absorbing/replacing variables which are not $\boldsymbol{x_1}$ $\boldsymbol{or}$ $\boldsymbol{x_2}$ as C's,

$$\boldsymbol{E_1 * E_2 = C_1 - (x_1 - C_2)^2 - (x_2 - C_3)^2] * [C_4 - (x_1 - C_5)^2 - (x_2 - C_6)^2]}$$

or (absorbing/replacing variables which are not $\boldsymbol{x_1}$ $\boldsymbol{or}$ $\boldsymbol{x_2}$ as D's,

$$\boldsymbol{E_1 * E_2 = [C_1 - {x_1}^2 + D_1x_1 - D_2 - {x_2}^2 + D_3x_2 - D_4\,] * [C_4 - {x_1}^2 + D_5x_1 - D_6 - {x_2}^2 + D_7x_2 - D_8]}$$

or (absorbing/replacing variables which are not $\boldsymbol{x_1}$ $\boldsymbol{or}$ $\boldsymbol{x_2}$ as F's,

$$\boldsymbol{E_1 * E_2 = [F_1 - {x_1}^2 + F_2x_1 - {x_2}^2 + F_3x_2\,] * [F_4 - {x_1}^2 + F_5x_1 - {x_2}^2 + F_6x_2]}$$

or (absorbing/replacing variables which are not $\boldsymbol{x_1}$ $\boldsymbol{or}$ $\boldsymbol{x_2}$ as G's,

$$\boldsymbol{\begin{aligned} E_1 * E_2 = [&\left(G_1 - {x_1}^2 + G_2x_1 - G_3{x_2}^2 + G_4x_2\right) + \left(-G_5{x_1}^2 + {x_1}^4 - G_6{x_1}^3 + G_7{x_1}^2{x_2}^2 - G_8{x_1}^2x_2\right) \\ &+ \left(G_9x_1 - G_{10}{x_1}^3 + G_{11}{x_1}^2 - G_{12}x_1{x_2}^2 + G_{13}x_1x_2\right) \\ &+ \left(- G_{14}{x_2}^2 + {x_1}^2{x_2}^2 - G_{15}x_1{x_2}^2 + {x_2}^4 - G_{16}{x_2}^3\right) \\ &+ \left(G_{17}x_2 - G_{18}{x_1}^2x_2 + G_{19}x_1x_2 - G_{20}{x_2}^3 + G_{21}{x_2}^2\right)] \end{aligned}}$$

The above can be condensed further by taking common terms out of x_1 and x_2. Then it is of the form which can be represented by the feature map:

$$\varphi(x_1, x_2) = (1, x_1, x_2, x_1x_2, {x_1}^2, {x_2}^2, {x_1}^2x_2, x_1{x_2}^2, {x_1}^3, {x_2}^3, {x_1}^3x_2, x_1{x_2}^2, {x_1}^2{x_2}^2)$$

or equivalently with the kernel:

$$\boldsymbol{K(x_1, x_2) = (1 + x_1.x_2)^4}$$

Thus, the kernel $\boldsymbol{K(x, y) = (1 + x.y)^4}$ will recover a separator.

Answer 2.4

The labels for the datapoints is not given, so I'm assuming them to be the same as was discussed in the class, i.e.

A = ((1, 1), -1)
B = ((-1, 1), 1)
C = ((-1, -1), -1)
D = ((1, -1), 1)

This uses the representation: ((x_1,x_2), y), where $\underline{x} = x_1, x_2$ is the datapoint and y is the label/output. We have four datapoints A, B, C and D.

And a correct classifier would be:

$$classify\ (\underline{x}) = sign(\boldsymbol{-x_1 * x_2})$$

We will now solve and prove the above using the dual SVM

The dual SVM problem would be:

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

Expanding the above:

$$\max_{\alpha_A,\alpha_B,\alpha_C,\alpha_D} [\alpha_A + \alpha_B + \alpha_C + \alpha_D] - \frac{1}{2}[\alpha_A K(A,A)\alpha_A - \alpha_A K(A,B)\alpha_B + \alpha_A K(A,C)\alpha_C - \alpha_A K(A,D)\alpha_D]$$
$$- \frac{1}{2}[-\alpha_B K(B,A)\alpha_A + \alpha_B K(B,B)\alpha_B - \alpha_B K(B,C)\alpha_C + \alpha_B K(B,D)\alpha_D]$$
$$- \frac{1}{2}[\alpha_C K(C,A)\alpha_A - \alpha_C K(C,B)\alpha_B + \alpha_C K(C,C)\alpha_C - \alpha_C K(C,D)\alpha_D]$$
$$- \frac{1}{2}[-\alpha_D K(D,A)\alpha_A + \alpha_D K(D,B)\alpha_B - \alpha_D K(D,C)\alpha_C + \alpha_D K(D,D)\alpha_D]$$

$$(s.t.)\alpha_A - \alpha_B + \alpha_C - \alpha_D = 0$$

$$\alpha_A, \alpha_B, \alpha_C, \alpha_D \geq 0$$

Part 1:
For the Polynomial Kernel: $K(\underline{x}, \underline{y}) = (1 + \underline{x}.\underline{y})^2$

$$\varphi(\underline{x}) = [1, x_1^2, \sqrt{2}x_1x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2]^T$$

and

$$\varphi(\underline{y}) = [1, y_1^2, \sqrt{2}y_1y_2, y_2^2, \sqrt{2}y_1, \sqrt{2}y_2]^T$$

Referencing

A = ((1, 1), -1)
B = ((-1, 1), 1)
C = ((-1, -1), -1)
D = ((1, -1), 1)

$$K(A,A) = (1 + \underline{A}.\underline{A})^2 = (1 + (\mathbf{1},\mathbf{1})\boldsymbol{dot}(\mathbf{1},\mathbf{1}))^2 = (1+1+1)^2 = 9$$
$$K(A,B) = (1 + \underline{A}.\underline{B})^2 = (1 + (\mathbf{1},\mathbf{1})\boldsymbol{dot}(-\mathbf{1},\mathbf{1}))^2 = (1-1+1)^2 = 1$$
$$K(A,C) = (1 + \underline{A}.\underline{C})^2 = (1 + (\mathbf{1},\mathbf{1})\boldsymbol{dot}(-\mathbf{1},-\mathbf{1}))^2 = (1-1-1)^2 = 1$$
$$K(A,D) = (1 + \underline{A}.\underline{D})^2 = (1 + (\mathbf{1},\mathbf{1})\boldsymbol{dot}(\mathbf{1},-\mathbf{1}))^2 = (1+1-1)^2 = 1$$

$$K(B,A) = (1 + \underline{B}.\underline{A})^2 = (1 + (-\mathbf{1},\mathbf{1})\boldsymbol{dot}(\mathbf{1},\mathbf{1}))^2 = (1-1+1)^2 = 1$$
$$K(B,B) = (1 + \underline{B}.\underline{B})^2 = (1 + (-\mathbf{1},\mathbf{1})\boldsymbol{dot}(-\mathbf{1},\mathbf{1}))^2 = (1+1+1)^2 = 9$$
$$K(B,C) = (1 + \underline{B}.\underline{C})^2 = (1 + (-\mathbf{1},\mathbf{1})\boldsymbol{dot}(-\mathbf{1},-\mathbf{1}))^2 = (1+1-1)^2 = 1$$
$$K(B,D) = (1 + \underline{B}.\underline{D})^2 = (1 + (-\mathbf{1},\mathbf{1})\boldsymbol{dot}(\mathbf{1},-\mathbf{1}))^2 = (1-1-1)^2 = 1$$

$$K(C,A) = (1 + \underline{C}.\underline{A})^2 = (1 + (-\mathbf{1},-\mathbf{1})\boldsymbol{dot}(\mathbf{1},\mathbf{1}))^2 = (1-1-1)^2 = 1$$
$$K(C,B) = (1 + \underline{C}.\underline{B})^2 = (1 + (-\mathbf{1},-\mathbf{1})\boldsymbol{dot}(-\mathbf{1},\mathbf{1}))^2 = (1+1-1)^2 = 1$$
$$K(C,C) = (1 + \underline{C}.\underline{C})^2 = (1 + (-\mathbf{1},-\mathbf{1})\boldsymbol{dot}(-\mathbf{1},-\mathbf{1}))^2 = (1+1+1)^2 = 9$$
$$K(C,D) = (1 + \underline{C}.\underline{D})^2 = (1 + (-\mathbf{1},-\mathbf{1})\boldsymbol{dot}(\mathbf{1},-\mathbf{1}))^2 = (1-1+1)^2 = 1$$

$$K(D,A) = (1 + \underline{B}.\underline{A})^2 = (1 + (\mathbf{1},-\mathbf{1})\boldsymbol{dot}(\mathbf{1},\mathbf{1}))^2 = (1+1-1)^2 = 1$$

$$K(D,B) = (1 + \underline{B}.\underline{B})^2 = (1 + (\mathbf{1},-\mathbf{1})\boldsymbol{dot}(-\mathbf{1},\mathbf{1}))^2 = (1-1-1)^2 = 1$$
$$K(D,C) = (1 + \underline{B}.\underline{C})^2 = (1 + (\mathbf{1},-\mathbf{1})\boldsymbol{dot}(-\mathbf{1},-\mathbf{1}))^2 = (1-1+1)^2 = 1$$
$$K(D,D) = (1 + \underline{B}.\underline{D})^2 = (1 + (\mathbf{1},-\mathbf{1})\boldsymbol{dot}(\mathbf{1},-\mathbf{1}))^2 = (1+1+1)^2 = 9$$

Now our objective function becomes:

$$\max_{\alpha_A,\alpha_B,\alpha_C,\alpha_D} [\alpha_A + \alpha_B + \alpha_C + \alpha_D] - \frac{1}{2}[9\alpha_A{}^2 - \alpha_A\alpha_B + \alpha_A\alpha_C - \alpha_A\alpha_D] - \frac{1}{2}[-\alpha_B\alpha_A + 9\alpha_B{}^2 - \alpha_B\alpha_C + \alpha_B\alpha_D]$$
$$- \frac{1}{2}[\alpha_C\alpha_A - \alpha_C\alpha_B + 9\alpha_C{}^2 - \alpha_C\alpha_D] - \frac{1}{2}[-\alpha_D\alpha_A + \alpha_D\alpha_B - \alpha_D\alpha_C + 9\alpha_D{}^2]$$

or,

$$\max_{\alpha_A,\alpha_B,\alpha_C,\alpha_D} \alpha_A + \alpha_B + \alpha_C + \alpha_D$$
$$- \frac{1}{2}[9\alpha_A{}^2 - 2\alpha_A\alpha_B + 2\alpha_A\alpha_C - 2\alpha_A\alpha_D + 9\alpha_B{}^2 - 2\alpha_B\alpha_C + 2\alpha_B\alpha_D + 9\alpha_C{}^2 - 2\alpha_C\alpha_D + 9\alpha_D{}^2]$$

Applying $\frac{\boldsymbol{\partial(Obj\ Func(\underline{\alpha}))}}{\boldsymbol{\partial\alpha_i}} = \mathbf{0}$ **[i=A,B,C,D]**, we get:

$$1 - 9\alpha_A + \alpha_B - \alpha_C + \alpha_D = 0$$
$$1 + \alpha_A - 9\alpha_B + \alpha_C - \alpha_D = 0$$
$$1 - \alpha_A + \alpha_B - 9\alpha_C + \alpha_D = 0$$
$$1 + \alpha_A - \alpha_B + \alpha_C - 9\alpha_D = 0$$

or (rearranging),

$$9\alpha_A - \alpha_B + \alpha_C - \alpha_D = 1$$
$$-\alpha_A + 9\alpha_B - \alpha_C + \alpha_D = 1$$
$$\alpha_A - \alpha_B + 9\alpha_C - \alpha_D = 1$$
$$-\alpha_A + \alpha_B - \alpha_C + 9\alpha_D = 1$$

Solving by Inverse Matrix Method:

A.X=B

$$A = \begin{bmatrix} 9 & -1 & 1 & -1 \\ -1 & 9 & -1 & 1 \\ 1 & -1 & 9 & -1 \\ -1 & 1 & -1 & 9 \end{bmatrix} \qquad \text{B=} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} 11/96 & 1/96 & -1/96 & 1/96 \\ 1/96 & 11/96 & 1/96 & -1/96 \\ -1/96 & 1/96 & 11/96 & 1/96 \\ 1/96 & -1/96 & 1/96 & 11/96 \end{bmatrix}$$

$$X = A^{-1}B = \begin{bmatrix} 11/96 & 1/96 & -1/96 & 1/96 \\ 1/96 & 11/96 & 1/96 & -1/96 \\ -1/96 & 1/96 & 11/96 & 1/96 \\ 1/96 & -1/96 & 1/96 & 11/96 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1/8 \\ 1/8 \\ 1/8 \\ 1/8 \end{bmatrix}$$

Therefore, $\alpha_A = \alpha_B = \alpha_C = \alpha_D = 1/8$

Therefore, all 4 inputs are support vectors, so optimum value of the $Obj\ Func(\underline{\alpha}) = 1/4$

Hence,

$$\frac{1}{2}\|\underline{w}^*\|^2 = \frac{1}{4}$$

or,

$$\|\underline{w}^*\| = \frac{1}{\sqrt{2}}$$

$$\underline{w}^* = \frac{1}{8}\left[-\varphi\left(\underline{y}_A\right) + \varphi\left(\underline{y}_B\right) - \varphi\left(\underline{y}_C\right) + \varphi\left(\underline{y}_D\right)\right]$$

Referencing

$$\varphi\left(\underline{y}\right) = [1, y_1^2, \sqrt{2}y_1y_2, y_2^2, \sqrt{2}y_1, \sqrt{2}y_2]^T$$

and

A = ((1, 1), -1)
B = ((-1, 1), 1)
C = ((-1, -1), -1)
D = ((1, -1), 1)

$$\underline{w}^* = \frac{1}{8}\left[-\begin{bmatrix}1\\1\\\sqrt{2}\\1\\\sqrt{2}\\\sqrt{2}\end{bmatrix} + \begin{bmatrix}1\\1\\-\sqrt{2}\\1\\-\sqrt{2}\\\sqrt{2}\end{bmatrix} - \begin{bmatrix}1\\1\\\sqrt{2}\\1\\-\sqrt{2}\\-\sqrt{2}\end{bmatrix} + \begin{bmatrix}1\\1\\-\sqrt{2}\\1\\\sqrt{2}\\-\sqrt{2}\end{bmatrix}\right]$$

or,

$$\underline{w}^* = \frac{1}{8}\begin{bmatrix}0\\0\\-4\sqrt{2}\\0\\0\\0\end{bmatrix} = \begin{bmatrix}0\\0\\-\sqrt{2}/2\\0\\0\\0\end{bmatrix} = \begin{bmatrix}0\\0\\-1/\sqrt{2}\\0\\0\\0\end{bmatrix}$$

The first element of w* is the bias:

$$bias = w_1^* = 0$$

The separating function is

$$\underline{w}^{*T}\varphi(\underline{x}) = 0$$

or,

$$\begin{bmatrix}0 & 0 & \frac{-1}{\sqrt{2}} & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}1\\{x_1}^2\\\sqrt{2}x_1x_2\\{x_2}^2\\\sqrt{2}x_1\\\sqrt{2}x_2\end{bmatrix} = 0$$

or,

$$-x_1x_2 = 0$$

or,

$$classify\left(\underline{x}\right) = sign(-\boldsymbol{x_1} * \boldsymbol{x_2})$$

Hence we found the separator using the dual SVM.

The separators are the the axes. A point will be classified as negative in the 1st and 3rd quadrant and a point will be classified as positive in the 2nd and 4th quadrant.

Part 2:

For the Gaussian-like (no sigma) Kernel: $K\left(\underline{x},\underline{y}\right) = exp\left(-\left\|\underline{x}-\underline{y}\right\|^2\right)$

Referencing

A = ((1, 1), -1)
B = ((-1, 1), 1)
C = ((-1, -1), -1)
D = ((1, -1), 1)

After finding the differenence of the 2 vectors, we find its norm^2 (so inner or dot product)

$$K(A,A) = e^{-(0)} = 1$$
$$K(A,B) = e^{-(4)}$$
$$K(A,C) = e^{-(8)}$$
$$K(A,D) = e^{-(4)}$$

$$K(B,A) = e^{-(4)}$$
$$K(B,B) = e^{-(4)} = 1$$
$$K(B,C) = e^{-(4)}$$
$$K(B,D) = e^{-(8)}$$

$$K(C,A) = e^{-(8)}$$
$$K(C,B) = e^{-(4)}$$
$$K(C,C) = e^{-(0)} = 1$$
$$K(C,D) = e^{-(4)}$$

$$K(D,A) = e^{-(4)}$$
$$K(D,B) = e^{-(8)}$$
$$K(D,C) = e^{-(4)}$$
$$K(D,D) = e^{-(0)} = 1$$

Our initial(general) objective function:

$$\max_{\alpha_A,\alpha_B,\alpha_C,\alpha_D}[\alpha_A+\alpha_B+\alpha_C+\alpha_D]-\frac{1}{2}[\alpha_A K(A,A)\alpha_A-\alpha_A K(A,B)\alpha_B+\alpha_A K(A,C)\alpha_C-\alpha_A K(A,D)\alpha_D]$$
$$-\frac{1}{2}[-\alpha_B K(B,A)\alpha_A+\alpha_B K(B,B)\alpha_B-\alpha_B K(B,C)\alpha_C+\alpha_B K(B,D)\alpha_D]$$
$$-\frac{1}{2}[\alpha_C K(C,A)\alpha_A-\alpha_C K(C,B)\alpha_B+\alpha_C K(C,C)\alpha_C-\alpha_C K(C,D)\alpha_D]$$
$$-\frac{1}{2}[-\alpha_D K(D,A)\alpha_A+\alpha_D K(D,B)\alpha_B-\alpha_D K(D,C)\alpha_C+\alpha_D K(D,D)\alpha_D]$$

$$(s.t.)\alpha_A-\alpha_B+\alpha_C-\alpha_D=0$$

$$\alpha_A,\alpha_B,\alpha_C,\alpha_D\geq 0$$

Now our objective function becomes:

$$\max_{\alpha_A,\alpha_B,\alpha_C,\alpha_D}[\alpha_A+\alpha_B+\alpha_C+\alpha_D]-\frac{1}{2}[\alpha_A{}^2-e^{-4}\alpha_A\alpha_B+e^{-8}\alpha_A\alpha_C-e^{-4}\alpha_A\alpha_D]$$
$$-\frac{1}{2}[-e^{-4}\alpha_B\alpha_A+\alpha_B{}^2-e^{-4}\alpha_B\alpha_C+e^{-8}\alpha_B\alpha_D]-\frac{1}{2}[e^{-8}\alpha_C\alpha_A-e^{-4}\alpha_C\alpha_B+\alpha_C{}^2-e^{-4}\alpha_C\alpha_D]$$
$$-\frac{1}{2}[-e^{-4}\alpha_D\alpha_A+e^{-8}\alpha_D\alpha_B-e^{-4}\alpha_D\alpha_C+\alpha_D{}^2]$$

or,

$$\max_{\alpha_A,\alpha_B,\alpha_C,\alpha_D} \alpha_A + \alpha_B + \alpha_C + \alpha_D - \frac{1}{2}[{\alpha_A}^2 - 2e^{-4}\alpha_A\alpha_B + 2e^{-8}\alpha_A\alpha_C - 2e^{-4}\alpha_A\alpha_D + {\alpha_B}^2 - 2e^{-4}\alpha_B\alpha_C + 2e^{-8}\alpha_B\alpha_D + {\alpha_C}^2 - 2e^{-4}\alpha_C\alpha_D + {\alpha_D}^2]$$

Applying $\frac{\boldsymbol{\partial(Obj\ Func(\underline{\alpha}))}}{\boldsymbol{\partial\alpha_i}} = \mathbf{0}$ **[i=A,B,C,D]**, we get:

$$\begin{gathered}
1-\alpha_A + e^{-4}\alpha_B - e^{-8}\alpha_C + e^{-4}\alpha_D = 0\\
1+e^{-4}\alpha_A - \alpha_B + e^{-4}\alpha_C - e^{-8}\alpha_D = 0\\
1-e^{-8}\alpha_A + e^{-4}\alpha_B - \alpha_C + e^{-4}\alpha_D = 0\\
1+e^{-4}\alpha_A - e^{-8}\alpha_B + e^{-4}\alpha_C - \alpha_D = 0
\end{gathered}$$

or (rearranging),

$$\begin{gathered}
\alpha_A - e^{-4}\alpha_B + e^{-8}\alpha_C - e^{-4}\alpha_D = 1\\
-e^{-4}\alpha_A + \alpha_B - e^{-4}\alpha_C + e^{-8}\alpha_D = 1\\
e^{-8}\alpha_A - e^{-4}\alpha_B + \alpha_C - e^{-4}\alpha_D = 1\\
-e^{-4}\alpha_A + e^{-8}\alpha_B - e^{-4}\alpha_C + \alpha_D = 1
\end{gathered}$$

Solving by Inverse Matrix Method:

A.X=B

$$A = \begin{bmatrix} 1 & -e^{-4} & e^{-8} & -e^{-4} \\ -e^{-4} & 1 & -e^{-4} & e^{-8} \\ e^{-8} & -e^{-4} & 1 & -e^{-4} \\ -e^{-4} & e^{-8} & -e^{-4} & 1 \end{bmatrix} \qquad \text{B=}\begin{bmatrix}1\\1\\1\\1\end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} \frac{e^{16}}{e^{16}-2e^8+1} & \frac{e^{12}}{e^{16}-2e^8+1} & \frac{e^{8}}{e^{16}-2e^8+1} & \frac{e^{12}}{e^{16}-2e^8+1} \\ \frac{e^{12}}{e^{16}-2e^8+1} & \frac{e^{16}}{e^{16}-2e^8+1} & \frac{e^{12}}{e^{16}-2e^8+1} & \frac{e^{8}}{e^{16}-2e^8+1} \\ \frac{e^{8}}{e^{16}-2e^8+1} & \frac{e^{12}}{e^{16}-2e^8+1} & \frac{e^{16}}{e^{16}-2e^8+1} & \frac{e^{12}}{e^{16}-2e^8+1} \\ \frac{e^{12}}{e^{16}-2e^8+1} & \frac{e^{8}}{e^{16}-2e^8+1} & \frac{e^{12}}{e^{16}-2e^8+1} & \frac{e^{16}}{e^{16}-2e^8+1} \end{bmatrix}$$

$$X = A^{-1}B = \begin{bmatrix} \frac{e^{16}}{e^{16}-2e^8+1} & \frac{e^{12}}{e^{16}-2e^8+1} & \frac{e^{8}}{e^{16}-2e^8+1} & \frac{e^{12}}{e^{16}-2e^8+1} \\ \frac{e^{12}}{e^{16}-2e^8+1} & \frac{e^{16}}{e^{16}-2e^8+1} & \frac{e^{12}}{e^{16}-2e^8+1} & \frac{e^{8}}{e^{16}-2e^8+1} \\ \frac{e^{8}}{e^{16}-2e^8+1} & \frac{e^{12}}{e^{16}-2e^8+1} & \frac{e^{16}}{e^{16}-2e^8+1} & \frac{e^{12}}{e^{16}-2e^8+1} \\ \frac{e^{12}}{e^{16}-2e^8+1} & \frac{e^{8}}{e^{16}-2e^8+1} & \frac{e^{12}}{e^{16}-2e^8+1} & \frac{e^{16}}{e^{16}-2e^8+1} \end{bmatrix} \begin{bmatrix}1\\1\\1\\1\end{bmatrix} = \begin{bmatrix} \frac{e^8}{e^8-2e^4+1} \\ \frac{e^8}{e^8-2e^4+1} \\ \frac{e^8}{e^8-2e^4+1} \\ \frac{e^8}{e^8-2e^4+1} \end{bmatrix}$$

Therefore,

$$\alpha_A = \alpha_B = \alpha_C = \alpha_D = \frac{e^8}{e^8 - 2e^4 + 1} = \frac{e^8}{(e^4-1)^2} \cong 1.03766$$

Therefore, all 4 inputs are support vectors, so optimum value of the

$$Obj\ Func(\underline{\alpha}) = \frac{1}{4}$$

Hence,

$$\frac{1}{2}\left\|\underline{w}^*\right\|^2 = \frac{1}{4}$$

or,

$$\left\|\underline{w}^*\right\| = \frac{1}{\sqrt{2}}$$

$$\underline{w}^* = \frac{e^8}{(e^4-1)^2}\left[-\varphi\left(\underline{y_A}\right) + \varphi\left(\underline{y_B}\right) - \varphi\left(\underline{y_C}\right) + \varphi\left(\underline{y_D}\right)\right]$$

Referencing

$$\varphi\left(\underline{y}\right) = [1, y_1^2, \sqrt{2}y_1y_2, y_2^2, \sqrt{2}y_1, \sqrt{2}y_2]^T$$

and

A = ((1, 1), -1)
B = ((-1, 1), 1)
C = ((-1, -1), -1)
D = ((1, -1), 1)

$$\underline{w}^* = \frac{e^8}{(e^4-1)^2}\left[-\begin{bmatrix}1\\1\\\sqrt{2}\\1\\\sqrt{2}\\\sqrt{2}\end{bmatrix} + \begin{bmatrix}1\\1\\-\sqrt{2}\\1\\-\sqrt{2}\\\sqrt{2}\end{bmatrix} - \begin{bmatrix}1\\1\\\sqrt{2}\\1\\-\sqrt{2}\\-\sqrt{2}\end{bmatrix} + \begin{bmatrix}1\\1\\-\sqrt{2}\\1\\\sqrt{2}\\-\sqrt{2}\end{bmatrix}\right]$$

or,

$$\underline{w}^* = \frac{e^8}{(e^4-1)^2}\begin{bmatrix}0\\0\\-4\sqrt{2}\\0\\0\\0\end{bmatrix} = \begin{bmatrix}0\\0\\\frac{-4\sqrt{2}e^8}{(e^4-1)^2}\\0\\0\\0\end{bmatrix}$$

The first element of w* is the bias:

$$bias = w_1^* = 0$$

The separating function is

$$\underline{w}^{*T}\varphi(\underline{x}) = 0$$

or,

$$\begin{bmatrix}0 & 0 & \frac{-4\sqrt{2}e^8}{(e^4-1)^2} & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}1\\{x_1}^2\\\sqrt{2}x_1x_2\\{x_2}^2\\\sqrt{2}x_1\\\sqrt{2}x_2\end{bmatrix} = 0$$

or,

$$-\frac{8e^8}{(e^4-1)^2}x_1x_2 = 0$$

or,

$$classify\left(\underline{x}\right) = sign\left(-\frac{8e^8}{(e^4-1)^2}x_1 * x_2\right)$$

Hence we found the separator using the dual SVM.

References:

1. Lecture Notes and Videos.
2. https://tohtml.com/python/
3. https://www.mathsisfun.com/algebra/circle-equations.html
4. https://www.mathwarehouse.com/ellipse/equation-of-ellipse.php
5. https://mathworld.wolfram.com/Ellipsoid.html
6. https://math.libretexts.org/Bookshelves/Algebra/Map%3A_College_Algebra_(OpenStax)/08%3A_Analytic_Geometry/8.02%3A_The_Ellipse#:~:text=Writing%20Equations%20of%20Ellipses%20Not%20Centered%20at%20the%20Origin,-Like%20the%20graphs&text=If%20an%20ellipse%20is%20translated,by%20(y%E2%88%92k)

***