CS 536 : Perceptrons and SVMs-Assignment № 3

Aravinda Reddy Dandu, Rutgers University 10/24/2020

Contents

1 Perceptrons

1.1 Data generation

Given method to generate data:

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.

Data is generated using the below function. This function serves as a general purpose data generator for all further operations in perceptrons

```
# Function to generate perceptron data. Takes epsilon, size of data and dimension as inputs
def gen_perceptron_data(e, m, k):
    dataset = {}
    # Initializing dataset storage with header
    for x in range(1, k + 1):
        dataset['x' + str(x)] = []
    dataset['y'] = []
    # Looping until dataset reaches required size which is m
    while len(dataset['x1']) < m:
        # Drawing k samples out of normal distribution using mean 0 and variance 1
        z = np.random.normal(0, 1, k)
        # Taking euclidian norm of Z
        z_norm = np.linalg.norm(z, 2)
        # Dividing Z vector with norm of Z to get X
        x = z / z_norm
        y = None
        # checking if absolute value is greater than epsilon. Continuing otherwise
        if abs(x[len(x) - 1]) > e:
            # Assigning positive if xk is positive(will obviously be greater than e). Negative else
            if x[len(x) - 1] > e:
```

```
                y = 1
            else:
                y = -1
        else:
            continue
        # Including X vector in dataset
        for index, xk in enumerate(x):
            dataset['x' + str(index + 1)].append(xk)
        # Appending Y values to last column
        dataset['y'].append(y)
    # Returning a pandas dataframe
    dataset = pd.DataFrame.from_dict(dataset)
    return dataset
```

A sample data generated using k as 5, m as *100* and ϵ as 0.2 looks like below

	x1	x2	x3	x4	x5	y
0	-0.637690	-0.254533	-0.469553	-0.382885	0.401851	1
1	-0.105606	-0.212329	0.316824	-0.087033	-0.914227	-1
2	-0.125693	0.171695	0.715603	0.206205	-0.632545	-1
3	-0.272733	-0.556376	0.396314	0.612019	-0.290568	-1
4	0.778924	-0.269206	0.089614	-0.439899	-0.345346	-1
..	...	...	...	...	...	..
95	-0.149478	-0.248669	0.672143	0.054015	-0.679062	-1
96	-0.690397	0.511306	-0.000118	-0.203999	-0.469364	-1
97	-0.398089	0.392455	-0.060761	-0.438334	0.701196	1
98	0.422985	-0.606463	-0.248129	-0.521433	-0.346158	-1
99	0.309706	0.262561	0.355346	-0.035801	0.841184	1

[100 rows x 6 columns]

1.2 Perceptron algorithm

A class is written for Perceptron which has a method to generate a perceptron for any given data. Here we assume that data is linearly separable and try to find weights and bias for a corresponding dataset. Class variables will be weight vector and bias value because these are the only values which define a perceptron.
Initializing with weight vector as 0 and bias as 0, we keep moving the perceptron until all the points are correctly classified.

1.2.1 Information being returned

Number of steps taken to converge : A step is defined as one move where weight and bias value is changed. Every time we encounter a misclassified point, we change the weight and bias an it counts as a *step*
Weight vector and bias : After all the points are correctly classified by a given perceptron, respective weights and bias values will be stored in the class variables.

```
# Main class to fit a perceptron on given data
```

```
class Perceptron():
    def __init__(self):
        pass

    # Function to fit a perceptron on given data
    def fit_perceptron(self, data):
        # Splitting the dataset for features and classifiers
        X_vectors = np.asarray(data.iloc[:, :-1])
        y_values = np.asarray(data.iloc[:, -1:])
        # Getting size of sample and dimension from first split
        m, k = X_vectors.shape

        # Declaring to class variables weights and bias. These will be updated for every mis classified point
        self.weights = np.zeros(shape=(k, 1))
        self.bias_val = 0

        # Initializing number of steps count
        number_of_steps = 0
        # Vector for storing y values calculated by perceptron at that instance
        Calculated_val = [0] * m
        i = 0
        # Running an indefinite loop until all points are correctly classified. Here we know that perceptron will
        # converge because of the dataset we generate
        while True:
            # Looping over the dataset looking for misclassified points
            for i in range(m):
                # Getting the dot product of weights and current X vector. This is w1*x1+ w2*x2 + ..... +wk*xk
                if (np.dot(X_vectors[i].reshape((k, 1)).T, self.weights) + self.bias_val)[0][0] > 0:
                    Calculated_val[i] = 1
                else:
                    Calculated_val[i] = -1
                # If calculated value agrees with y value of vector, perceptron correctly classified the point
                if float(Calculated_val[i]) != float(y_values[i]):
                    # Updating weights and bias if wrongly classified
                    self.weights += np.dot(X_vectors[i].reshape((k, 1)), y_values[i].reshape((1, 1)))
                    self.bias_val += y_values[i]
                    number_of_steps += 1
            # Storing all calculated values in the dataset
            Calculated_val = np.asarray(Calculated_val).reshape((m, 1))
            i += 1
            # If all points in dataset correctly classified, break the loop and return the number of steps taken. The
            # class will have weights and bias values if needed
            if (np.array_equal(y_values, Calculated_val)) or (number_of_steps >= 10000):
```

```
                break
        return number_of_steps
```

1.3 Question 1.1

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?

Here we generate 20 datasets for each value of *m* and vary *m* over a range of 10 to 5000. For each value of *m*, average number of steps taken is calculated.
Code for running iterations and calculating values below. This code also includes logic for bonus question(will be explained later).

```
# Varying M and getting number of steps taken and distance from ideal perceptron in each case
def test_vary_m():
    storage = []
    # Range of m values from 10, 5000 with increments of 100
    for m in tqdm(range(10, 5000, 100)):
        steps = []
        weights = []
        biases = []
        # Looping 20 times and taking average number of steps to get a better estimate
        for i in range(0, 20):
            # Get perceptron data with e = 0.1, m as a variable and k as 5
            data = gen.gen_perceptron_data(0.1, m, 5)
            pt = Perceptron()
            steps.append(pt.fit_perceptron(data))
            # Normalizing all the weights to make them uniform
            den = np.sqrt(np.square(np.linalg.norm(pt.weights)) + np.square(pt.bias_val))
            pt.weights = pt.weights / den
            pt.bias_val = pt.bias_val / den
            # Get weights of perceptron and store them
            weights.append(pt.weights)
            # Get biases of perceptron and store them
            biases.append(pt.bias_val)
        # Calculating average weights using np.mean and axis as 0
        avg_w = np.mean(weights, axis=0)
        # Calculating bias the same way
        avg_b = np.mean(biases, axis=0)
        # Getting the distance from "ideal" perceptron
        dist = get_dist_from_ideal(weights=avg_w, bias=avg_b)
        # Returning all values calculated
        storage.append(str(m) + "," + str(np.average(steps)) + "," + str(dist))
```

```
    return storage
```

This gives below output

```
m,steps,dist
10,5.5,0.019489341654281102
110,18.0,0.003741245883055618
210,20.1,0.0008855192074725559
310,23.0,0.0005059914666722962
410,23.2,0.00025653320008841673
510,22.7,0.00029609120136727437
610,21.8,0.0007503066622136762
710,22.3,0.0003418595277035938
810,24.6,0.00014717323003123932
910,23.1,0.00023137543389946573
1010,26.6,0.0009667901065258704
1110,25.2,0.00012727281759664525
1210,26.7,0.0004519138170351651
1310,28.6,0.000380831454051946
1410,27.7,0.0003024906779735875
1510,29.0,0.0009825533419468453
1610,27.1,0.0019581086302622006
1710,26.7,0.00012581869760750503
1810,27.3,0.0005850925989935943
1910,25.9,0.00019724733504510606
2010,29.6,0.00022531452223603434
2110,25.9,0.0005475133482714749
2210,30.0,0.00034486506066963763
2310,27.5,0.0006426021847137075
2410,27.2,0.00040621419264086177
2510,31.1,0.00043452634846506466
2610,30.5,0.0005569729728738002
2710,29.7,0.0006764970999968959
2810,25.5,0.00033635171409906995
2910,28.0,0.0004318544179080017
3010,28.5,0.0005534896992075317
3110,27.8,0.0003803624637624665
3210,28.6,0.00038719553227329473
3310,29.8,0.00011749664108536564
3410,27.2,0.00015566645416848981
3510,27.4,0.0002740062616987382
3610,27.9,0.00027755673650489975
3710,27.1,0.0005468490446676837
3810,28.4,0.0002611695784311267
3910,28.4,0.00020052087857995798
4010,31.8,0.0007349533386939369
4110,27.6,0.00036789631161839004
4210,30.3,8.138274817110553e-05
4310,28.6,0.0005782565382362639
4410,31.7,0.00035775107189336107
```

```
4510,33.0,0.000587761788711138
4610,29.3,0.0001972873130329835
4710,31.8,4.509583525873702e-05
4810,27.3,0.0002902779624813425
4910,33.1,0.00011546399748538787
```

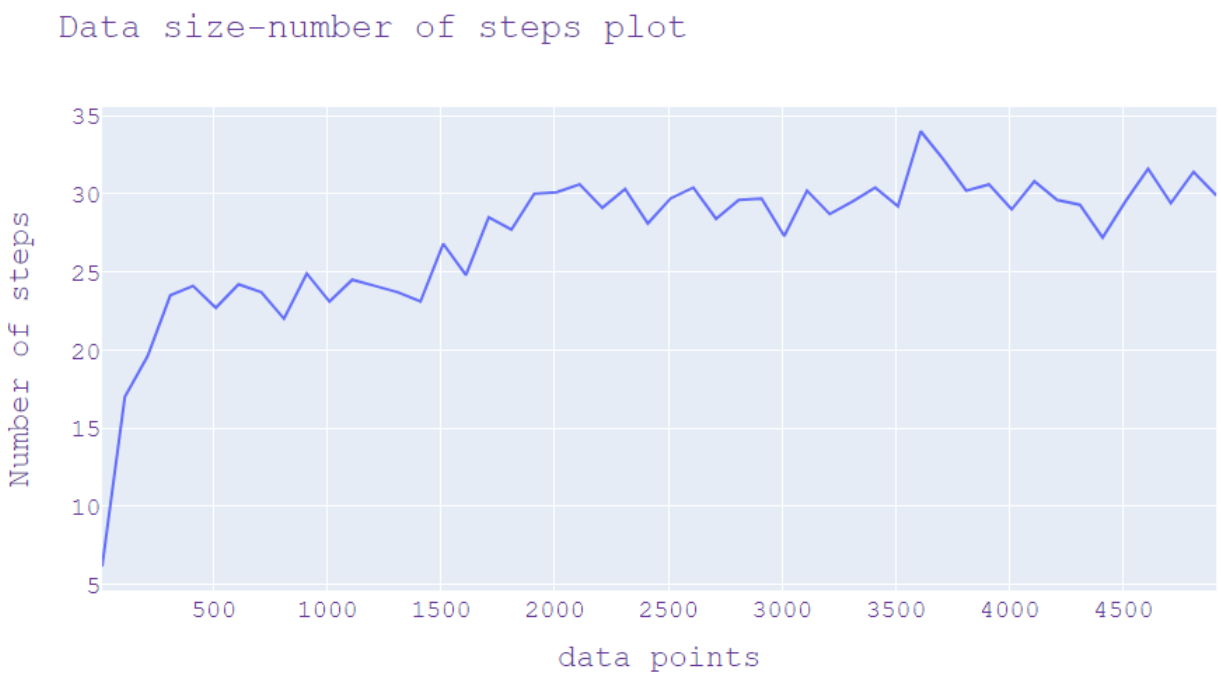
Plotting using plotly

```
import plotly.graph_objects as go
import pandas as pd
cols=['m', 'steps', 'dist']
data= pd.read_csv("D:/Study/ML/Perceptrons_SVMs/data_m_var.csv",usecols=cols)
fig = go.Figure()

fig.add_trace(
    go.Scatter(
        x=data['m'],
        y=data['steps'],
    ))


fig.update_layout(
    title="Data size-number of steps plot",
    xaxis_title="data points",
    yaxis_title="Number of steps",
    font=dict(
        family="Courier New, monospace",
        size=18,
        color="RebeccaPurple"
    )
)
```

Number of steps taken versus data size will give below graph



We can see that number of steps will converge at somewhere **between 25 and 35**. Even after increasing *m* drastically, number of steps will remain constant. This proves that perceptron complexity is independent of data size

Coming to the first few data points where it starts rising and converges, these data points are between 10 and $\tilde{3}50$. Reason why these *m* values have lesser number of steps is *Probability that at least one point falling at the separation line(Gap that we kept while generating dataset) is very less with less data points*. As points are far from separator, it gets easier to classify. Once a data point is located at the separation line, number of steps needed will remain regardless of number of data points.

This can be proven programmatically by repeatedly generating datasets of size 100 and checking number of times it is less than 25(converging value).

```
1  def test_m_100():
2      storage = []
3      for m in [100]:
4          steps = []
5          for i in range(0, 1000):
6              data = gen.gen_perceptron_data(0.1, m, 5)
7              pt = Perceptron()
8              nu = pt.fit_perceptron(data)
9              steps.append(nu)
10         storage.append(steps)
11         # storage.append(str(m) + "," + str(np.max(steps)))
12     steps = storage[0]
13     lesscnt = 0
14     for n in steps:
15         if n < 25:
16             lesscnt += 1
17     print('Number of times steps is less than 20 is ' + str(lesscnt) + ' out of ↩
           1000')
18     print('maximum steps is ' + str(max(steps)))
```

```
Number of times steps is less than 20 is 909 out of 1000
maximum steps is 40
```

Here we can see 909/1000 times, number of steps is less than 25. This shows, probability if points falling on separation line is low. This is why many steps is not required to converge as gap is large.

1.4 Question 1.2

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?

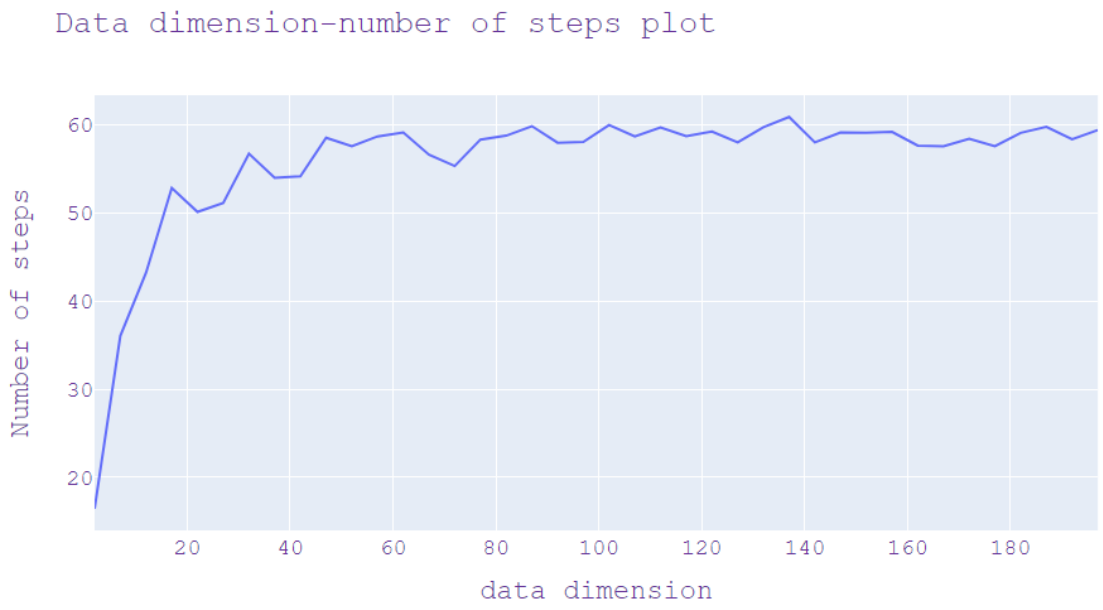
Same as *m*, we vary *k* for range between 1 and 200

```python
# Varying K and getting number of steps taken and distance from ideal perceptron in each case
def test_vary_k():
    storage = []
    # Range of k values from 1, 200 with increments of 5
    for k in tqdm(range(1, 200, 5)):
        steps = []
        weights = []
        biases = []
        # Looping 50 times and taking average number of steps to get a better estimate
        for i in range(0, 50):
            # Get perceptron data with e = 0.05, m as 100 and k as a variable
            data = gen.gen_perceptron_data(0.05, 100, k)
            pt = Perceptron()
            steps.append(pt.fit_perceptron(data))
            den = np.sqrt(np.square(np.linalg.norm(pt.weights)) + np.square(pt.bias_val))
            pt.weights = pt.weights / den
            pt.bias_val = pt.bias_val / den
            # Get weights of perceptron and store them
            weights.append(pt.weights)
            # Get biases of perceptron and store them
            biases.append(pt.bias_val)
        # Calculating average weights and biases using np.mean and axis as 0
        avg_w = np.mean(weights, axis=0)
        avg_b = np.mean(biases, axis=0)
        # Getting the distance from "ideal" perceptron
        dist = get_dist_from_ideal(weights=avg_w, bias=avg_b)
        # Returning all values calculated
        storage.append(str(k) + "," + str(np.average(steps)) + "," + str(dist))
    return storage
```

This gives below data

```
k,steps,dist
1,1.0,0.5857864376269051
6,35.12,0.000792159091595642
11,41.96,0.00031879176668000563
16,48.28,0.002254649180593259
21,51.08,0.003400524838979116
26,54.04,0.0038550581261809582
31,54.64,0.003711919347351471
36,55.0,0.008862846496991766
41,56.28,0.00614584153155933
46,58.12,0.007023159307302922
51,56.56,0.007688828000580725
56,55.84,0.011459824234848238
61,60.0,0.011155824076315938
66,58.6,0.00849497549660349
71,58.8,0.01325382244363926
76,60.8,0.01110629363120088
81,55.12,0.016952300943098672
86,58.72,0.015701964987362517
91,59.64,0.015569634807497248
96,58.8,0.019670416214498727
101,57.04,0.02201314269528506
106,60.72,0.01766211538212445
111,59.68,0.017378414236108863
116,60.32,0.017127343120082607
121,58.52,0.021073563037323682
126,57.0,0.019483854280533577
131,60.88,0.024178362458176155
136,56.84,0.02630801908628984
141,59.56,0.02467376302048073
146,59.64,0.026099787316625253
151,58.36,0.022567438467676185
156,58.52,0.024212178692972354
161,60.36,0.02593934400425663
166,57.76,0.025307979318785707
171,59.16,0.028588054401244144
176,58.8,0.026819771515990912
181,59.36,0.033574121955407696
186,56.84,0.028033870434750377
191,58.68,0.028245189496670018
196,57.72,0.030840883821933104
```

Plotting the output



Here we can see that the line neatly converges to $\tilde{6}0$ and irrespective of increase of dimension, stays there. So, we can infer that complexity is independent of dimension as well.

1.5 Question 1.3

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?

Same way, we generate data by varying ϵ between 0 and 1. Step size is taken as 0.02

```
# Varying M and getting number of steps taken and distance from ideal perceptron in each case
def test_vary_e():
    storage = []
    # Range of e values from 0.01, 1 with increments of 0.02
    for e in tqdm(range(1, 100, 2)):
        steps = []
        weights = []
        biases = []
        e = e / 100
        # Looping 30 times and taking average number of steps to get a better estimate
        for i in range(0, 30):
            # Get perceptron data with e as a variable, m as 100 and k = 5
            data = gen.gen_perceptron_data(e, 100, k=5)
            pt = Perceptron()
            steps.append(pt.fit_perceptron(data))
            den = np.sqrt(np.square(np.linalg.norm(pt.weights)) + np.square(pt.bias_val))
            pt.weights = pt.weights / den
```

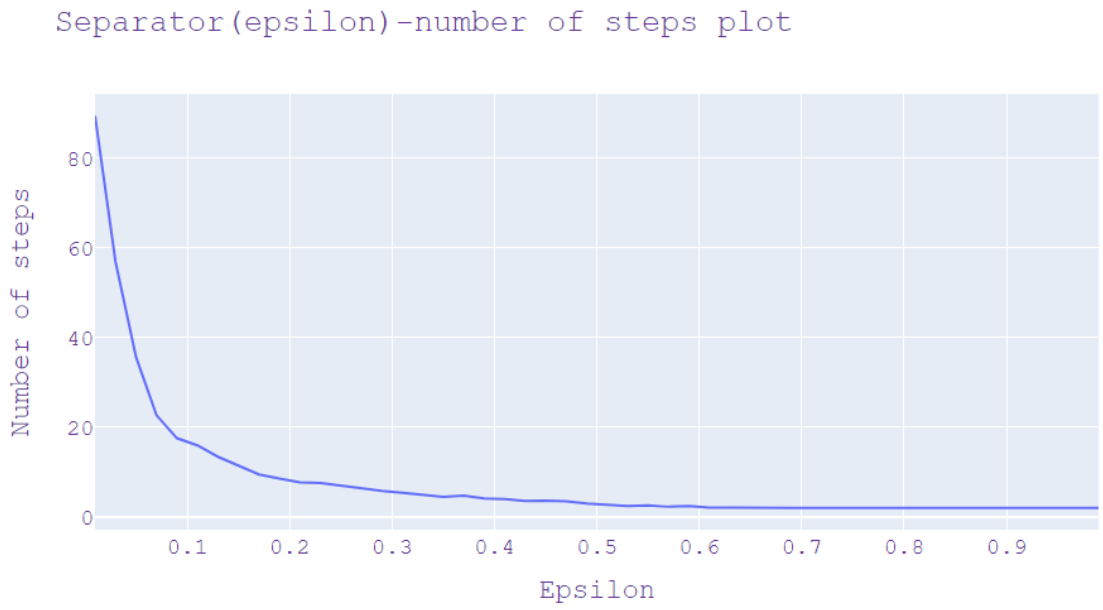
```
            pt.bias_val = pt.bias_val / den
            # Get weights and biases of perceptron and store them
            weights.append(pt.weights)
            biases.append(pt.bias_val)
        # Calculating average weights and biases using np.mean and axis as 0
        avg_w = np.mean(weights, axis=0)
        avg_b = np.mean(biases, axis=0)
        # Getting the distance from "ideal" perceptron
        dist = get_dist_from_ideal(weights=avg_w, bias=avg_b)
        # Returning all values calculated
        storage.append(str(e) + "," + str(np.average(steps)) + "," + str(dist))
    return storage
```

This gives below data

```
e,steps,dist
0.01,79.3,0.000251573281694657
0.03,44.166666666666664,0.00038497288786862836
0.05,30.466666666666665,0.0008442735446659691
0.07,23.4,0.0006796210666929297
0.09,18.0,0.00045169585805523825
0.11,15.866666666666667,0.0008552694318976155
0.13,13.333333333333334,0.0006728406190542547
0.15,11.466666666666667,0.0008774147777884127
0.17,9.933333333333334,0.006073310901259595
0.19,9.266666666666667,0.0024628739411135855
0.21,8.4,0.0017496265566388533
0.23,7.4,0.002741258975014222
0.25,6.0,0.0018680011915344326
0.27,5.733333333333333,0.003606969573007876
0.29,5.066666666666666,0.002600249430509449
0.31,5.266666666666667,0.006418991432606572
0.33,5.033333333333333,0.010954987395197702
0.35,4.733333333333333,0.003567297494209923
0.37,4.2,0.0034202039673193866
0.39,4.0,0.005934374635189128
0.41,3.7333333333333334,0.0014837974666425764
0.43,3.6666666666666665,0.0021682842515879323
0.45,3.3333333333333335,0.009082218527788934
0.47,3.4,0.0014736669621873565
0.49,3.0,0.004366047165908399
0.51,3.066666666666667,0.009538339063128794
0.53,2.6666666666666665,0.004030965159470492
0.55,2.4,0.008314647429324758
0.57,2.466666666666667,0.011931785718988914
0.59,2.1333333333333333,0.002715749944516967
0.61,2.2,0.005419412874700856
0.63,2.1,0.014677085603064775
0.65,2.0,0.0028529555732480626
0.67,2.0,0.008392828839299626
```

```
0.69,2.0,0.005282135462813741
0.71,2.0,0.009546609580059049
0.73,2.0,0.010786362991078912
0.75,2.0,0.004168047709009224
0.77,2.0,0.015064499366818236
0.79,2.0,0.018148851550467797
0.81,2.0,0.01448580727980522
0.83,2.0,0.002171327448825599
0.85,2.0,0.0005352089681034504
0.87,2.0,0.002831933180653132
0.89,2.0,0.0038489144130229116
0.91,2.0,0.0020277901924322084
0.93,2.0,0.0019955801434043555
0.95,2.0,0.0013001818080805889
0.97,2.0,0.0006986272544607081
0.99,2.0,0.00010029932603725317
```

Plotting the above data gives this graph. Using plotly



We can clearly say that *larger the gap, easier to find a perceptron*. Here for smaller ϵ values, it takes larger number of steps as gap is small. For higher values, gap increases and it gets easier to solve.

1.6 Bonus Question

A method to calculate distance of any perceptron from the ideal perceptron is written. We normalize the input vector to calculate distance.

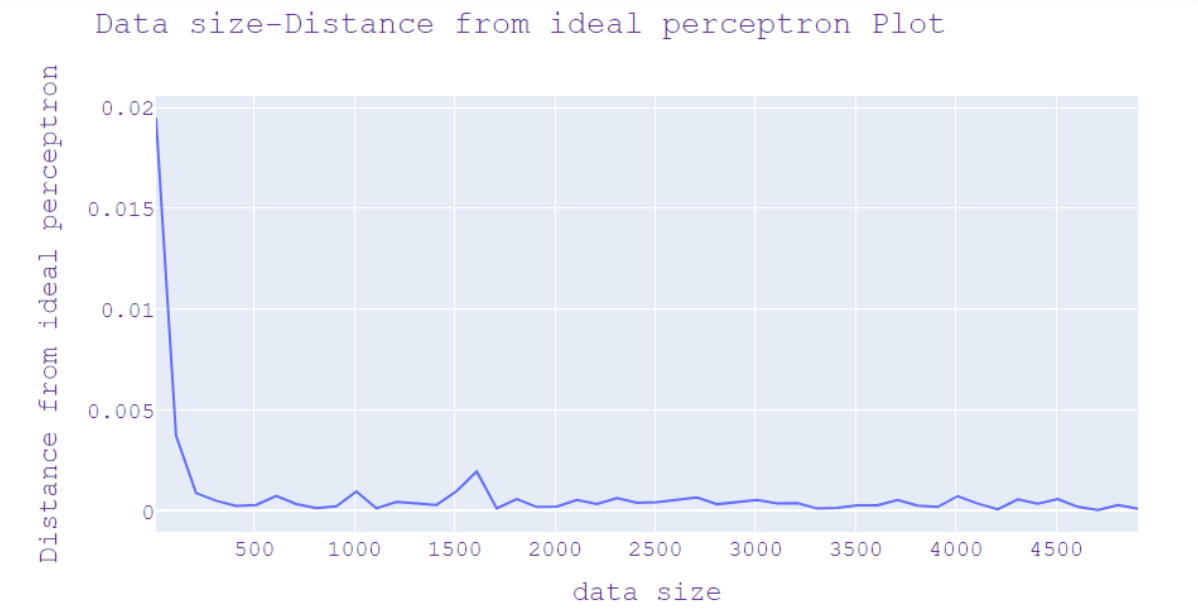
```python
# Method to get distance of any perceptron my the ideal perceptron
def get_dist_from_ideal(weights, bias):
    # getting dimension from input
    k = len(weights)
    # Calculating the denominator for normalizing the input perceptron to 1
    den = np.sqrt(np.square(np.linalg.norm(weights)) + np.square(bias))
```

```
    weights = weights / den
    bias = bias / den
    # Defining ideal perceptron
    ideal_weights = np.zeros(shape=(k, 1))
    ideal_weights[k - 1] = 1
    ideal_bias_val = 0
    # Calculating distance
    distance = np.square(np.linalg.norm(weights - ideal_weights)) + np.square(np.↩
        linalg.norm(bias - ideal_bias_val))
    return distance
```

In all the methods to calculate number of steps while varying different parameters, we also store the weights and biases.
An important point to note is that, we have to normalize each weight vector before finding average of that specific parameter value. Will be explained below

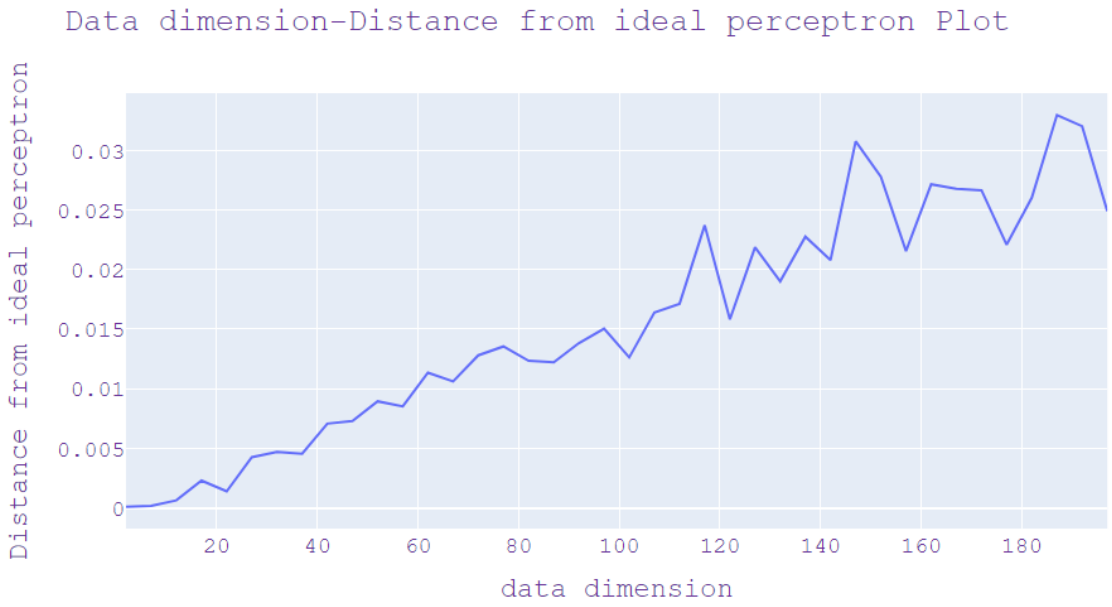
1.6.1 Varying *m*

While varying, *m*, distance from ideal vector converges very fast at 200 data size to a small value of 0.0008855192074725559. For smaller values of m, error is greater because gap γ is larger. Because of this, perceptron algorithm stops after correctly classifying points and gap still remains.
Plotting values gives below graph



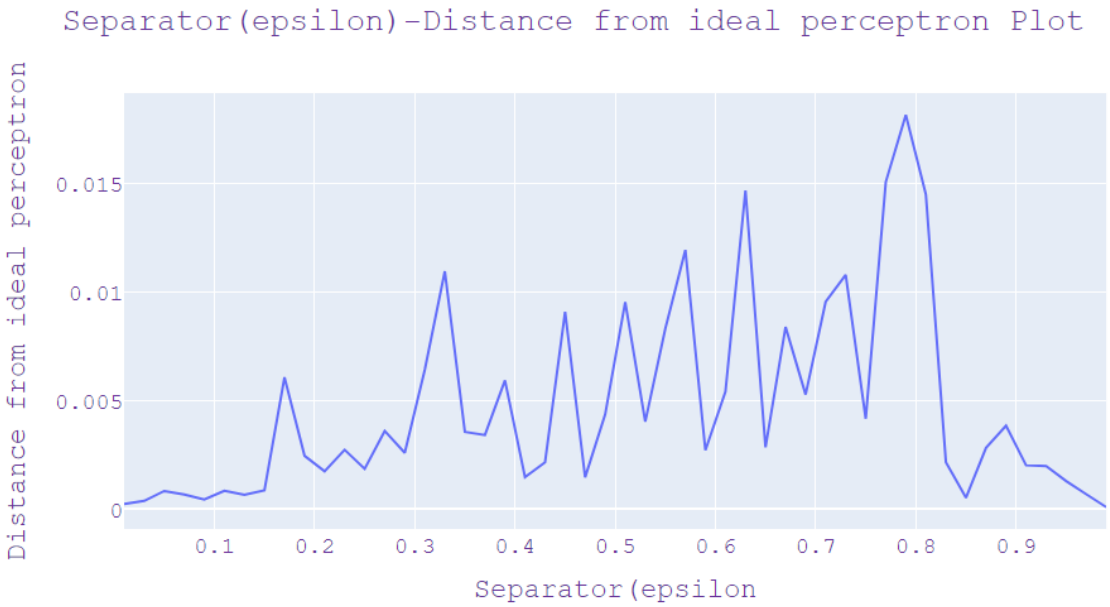
1.6.2 Varying *k*

While varying, *k* error rate slowly increases. My intuition is that- For a fixed size of m=100, increasing the dimension, makes larger gap in higher dimension and perceptron easily converges at some farther value from the ideal one which is at the centre. For higher values of *k*, it stays at $\tilde{0}$.02 for many values



1.6.3 Varying ϵ

While varying ϵ, distance from ideal vector doesn't make any sense. It varies wildly from 0 to 1.



2 Support Vector Machines

SVMs

Saturday, October 24, 2020 1:15 PM

HW-
Perceptro...

SVMs

1) Suppose you had a data set in two dimensions that satisfied the following: the positive class all lay within a certain radius of a point, the negative class all lay outside that radius.

   – Show that under the feature map $\phi(x_1, x_2) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$ (or equivalently, with the kernel $K(\underline{x}, \underline{y}) = (1 + \underline{x}.\underline{y})^2$), a linear separator can always be found in this embedded space, *regardless of radius and where the data is centered*.

   – In fact show that if there is an ellipsoidal separator, regardless of center, width, orientation (and dimension!), a separator can be found in the quadratic feature space using this kernel.

2) As an extension of the previous problem, suppose that the two dimensional data set satisfied the following: the positive class lay within one of two (disjoint) ellipsoidal regions, and the negative class was everywhere else. Argue that the kernel $K(\underline{x}, \underline{y}) = (1 + \underline{x}.\underline{y})^4$ will recover a separator.

3) Suppose that the two dimensional data set is distributed like the following: the positive class lays in a circle centered at some point, the negative class lies in a circular band surrounding it of some radius, and then additional positive points lie outside that radius. Argue that the kernel $K(\underline{x}, \underline{y}) = (1 + \underline{x}.\underline{y})^4$ will recover a separator.

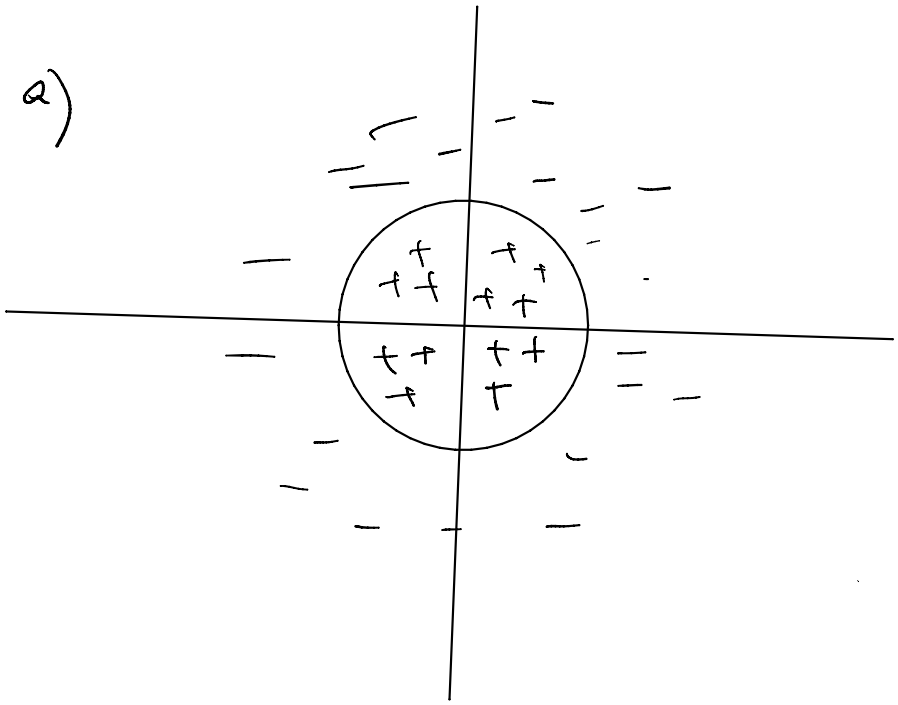
4) Consider the XOR data (located at $(\pm 1, \pm 1)$). Express the dual SVM problem and find a separator using

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

   – $K(\underline{x}, \underline{y}) = \exp(-||\underline{x} - \underline{y}||^2)$.

   For each, determine the regions of (x_1, x_2) space where points will be classified as positive or negative. Given that each produces a distinct separator, how might you decide which of the two was preferred?

1) a)



Above diagram represents data being separated by the non-linear circular separator.

equation of a circle is

$$x^2 + y^2 = r^2$$

Changing x to x_1 and y to x_2 to avoid confusion with sign and y

$$x_1^2 + x_2^2 = r^2$$

$$x_1^2 + x_2^2 = r^2$$

Let

$$f(x) = x_1^2 + x_2^2 - r^2$$

if this is < 0, point lies inside the circle. otherwise point lies outside the circle.

So, we can define our seperator as

$$x_1^2 + x_2^2 - r^2 < 0 \quad +1$$

$$x_1^2 + x_2^2 - r^2 > 0 \quad -1$$

Our sign function is the above function which is circle inequation.

Here we needed x_1^2 and x_2^2 order features which will recover a seperator

Both of these can be found in the feature set $(1, x_1, x_2, x_1 x_2, x_1^2, x_2^2)$.

So, a feature map in the above format will recover a circular seperator

will recover a circular separator.

b)

An ellipse is represented by following equation

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

$$b^2x^2 + a^2y^2 = a^2b^2$$

Switching variables again

$$b^2x_1^2 + a^2x_2^2 = a^2b^2$$

if we have to shift the origin, equation becomes

$$b^2(x_1 - z_1)^2 + a^2(x_2 - z_2)^2 = a^2b^2$$

Here we can write the sign function as

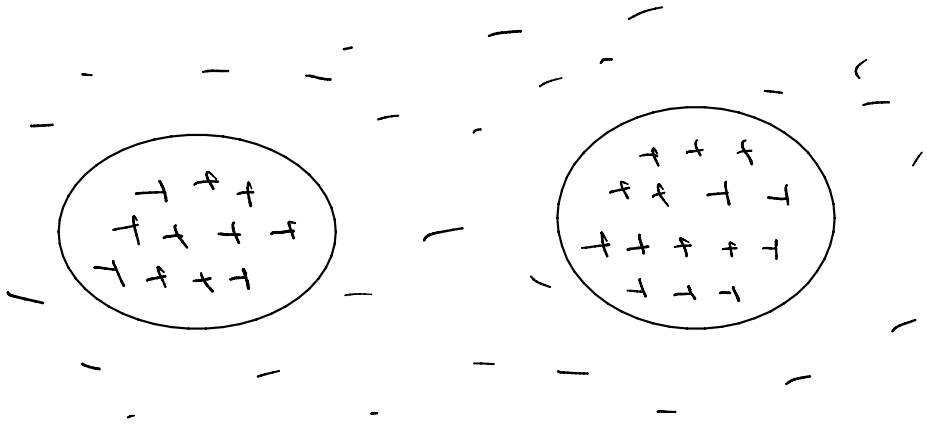
$$b^2(x_1 - z_1)^2 + a^2(x_2 - z_2)^2 - a^2b^2 < 0 \rightarrow +1$$

$$b^2(x_1 - z_1)^2 + a^2(x_2 - z_2)^2 - a^2b^2 > 0 \rightarrow -1$$

This separator also has the highest order of x_1 and x_2 as 2 and no other terms other than its first order.

So $\{x_1, x_2, x_1x_2, x_1^2, x_2^2\}$ feature space will recover if there is a seperator as defined above.

2)



From the discussion board, it is

From the discussion board, it is understood that data is separated in the above format.

We have two ellipses, both containing positive data and negatives everywhere else

Let first ellipse be

$$b_1^2 (x_1 - z_1)^2 + a_1^2 (x_2 - z_2)^2 = a_1^2 b_1^2$$

This can be defined as

$$K_1 \Rightarrow b_1^2 (x_1 - z_1)^2 + a_1^2 (x_2 - z_2)^2 - a_1^2 b_1^2$$

Second be

$$b_2^2 (x_1 - z_3)^2 + a_2^2 (x_2 - z_4)^2 = a_2^2 b_2^2$$

$$K_2 \Rightarrow b_2^2 (x_1 - z_3)^2 + a_2^2 (x_2 - z_4)^2 - a_2^2 b_2^2$$

Let a function K be defined as product of K_1 and K_2

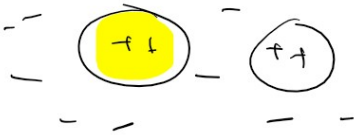
Same way we represent two circles next to each other by product of thier equations, we define

$$K = K_1 \times K_2$$

Here we have 3 classes of points

class 1) Inside the first ellipse

inequations will be



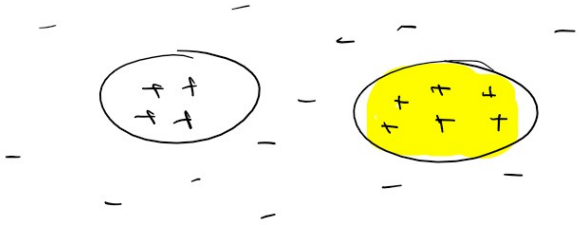
$K_1 < 0$

$K_2 > 0$

$\Rightarrow K_1 K_2 < 0$ } +ve

Data is positive

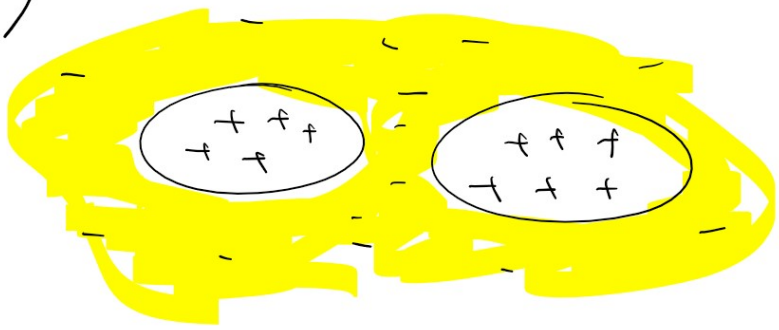
class 2) Inside second ellipse



Same as inside first ellipse

$$\left.\begin{array}{l} K_1 > 0 \\ K_2 < 0 \end{array}\right\} \Rightarrow K_1 K_2 < 0 \Big\} \text{ +ve}$$

Class 3)



Here

$$\left.\begin{array}{l} K_1 > 0 \\ K_2 > 0 \end{array}\right\} \Rightarrow K_1 K_2 > 0 \Big\} \text{ -ve}$$

↙

Points are -ve

So, our separator function can just be the below

$K_1 K_2 < 0 \quad +ve$

$K_1 K_2 > 0 \quad -ve$

$K_1 K_2$ will look like

$$\left(b_1^2(x_1^2 - z_1^2) + a_1^2(x_2^2 - z_2^2) - a_1^2 b_1^2\right) \times$$

$$\left((b_2^2(x_1^2 - z_3^2) + a_2^2(x_2^2 - z_4^2) - a_2^2 b_2^2\right)$$

This raises to powers of 4 and Variables will be

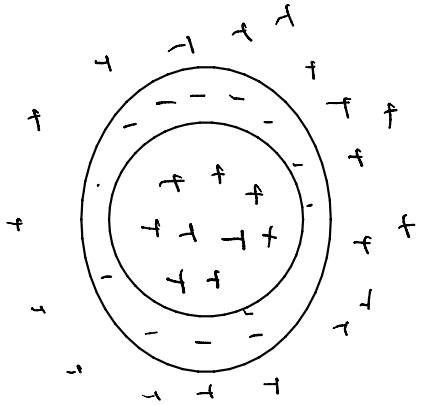
$$\{x_1^4, x_2^4, x_1^2 x_2^2\}$$

All these features can be recovered from Kernel with dimension 4.

from Kernel with dimension 4.

So, $(1 + \underline{x}\,\underline{y})^4$ will recover a Separator in the given case.

3)



Again, two inequations of circle

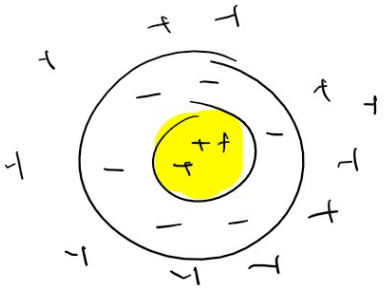
$K_1 \Rightarrow (x_1 - a_1)^2 + (x_2 - b_1)^2 - r_1^2 \rightarrow$ Inner circle

$K_2 \Rightarrow (x_1 - a_2)^2 + (x_2 - b_2)^2 - r_2^2 \rightarrow$ outer circle

We have 3 classes of points again

We have 3 classes of points again

1) Inside inner circle



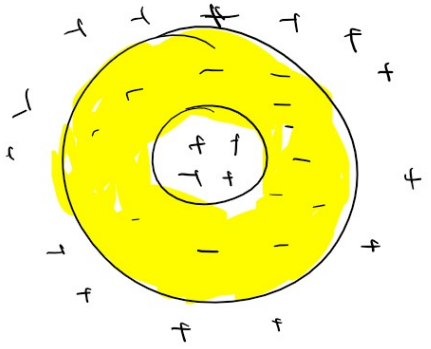
Point is inside both circles, so

$$\left.\begin{array}{l} K_1 < 0 \\ K_2 < 0 \end{array}\right\} \; K_1 K_2 > 0 \Rightarrow +ve$$

↓

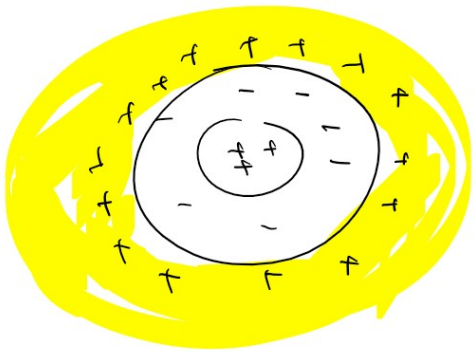
Data is +ve

2) Inside outer circle, outside inner circle



$K_1 > 0$
$K_2 < 0$
} $K_1 K_2 < 0 \Rightarrow$ -ve

3) outside both circles,



$K_1 > 0$
$K_2 > 0$
} $K_1 K_2 > 0$ } +ve

So our classifier can be

$K = K_1 K_2$ } > 0 +ve
< 0 -ve

K can be expressed as

K can be expressed as

$$K \Rightarrow$$

$$\left((x_1 - a_1)^2 + (x_2 - b_1)^2 - r_1^2 \right) \times$$

$$\left((x_2 - a_2)^2 + (x_2 - b_2)^2 - r_2^2 \right)$$

This has

$$x_1^4, x_2^4, x_1^2 x_2^2$$ terms in it.

All of these can be retrieved using Kernel

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

4)

a) written on paper. will be attached along with this submission.

b) In a, we derived expanded form of the kernel function as

$$\max_{\alpha_1 \ldots \alpha_4} (\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4) - \frac{1}{2}\Big[K_{11} y_1^2 \alpha_1^2 + K_{22} y_2^2 \alpha_2^2 + K_{33} y_3^2 \alpha_3^2 + K_{44} y_4^2 \alpha_4^2 + 2\big[y_1 y_2 \alpha_1 \alpha_2 K_{12} + y_2 y_3 \alpha_2 \alpha_3 K_{23} + y_3 y_4 \alpha_3 \alpha_4 K_{34} + y_4 y_1 \alpha_4 \alpha_1 K_{41} + y_1 y_3 \alpha_1 \alpha_3 K_{13} + y_2 y_4 \alpha_2 \alpha_4 K_{24}\big]\Big]$$

Now we find kernel values using

$$K(\underline{x}, \underline{y}) = \exp\left(-\|\underline{x} - \underline{y}\|^2\right)$$

$$K_{11} = \exp\left(-\|(-1\ -1) - (-1\ -1)\|^2\right)$$

$$= \exp(0) = 1 \quad \text{same for } K_{22}, K_{33}, K_{44}$$

$$K_{12} = \exp\left(-\left\|(-1\ -1) - (-1\ +1)\right\|^2\right)$$

$$= \exp\left(-\left\|0\ \ -2\right\|^2\right)$$

$$= \exp\left(-(2)^2\right) = e^{-4}$$

$$K_{23} = \exp\left(-\left\|(-1\ 1) - (1\ -1)\right\|^2\right)$$

$$= \exp\left(-\left(-2\ \ 2\right)\right|^2)$$

$$= e^{-8}$$

$$K_{34} = \exp\left(-\left\|(+1\ -1) - (1\ 1)\right\|^2\right)$$

$$= e^{-4}$$

$$K_{41} = \exp\left(-\left\|(1\ 1) - (-1\ -1)\right\|^2\right)$$

$$= e^{-8}$$

$$K_{13} = \exp\left(-\|(1\ -1) - (1\ -1)\|^2\right)$$

$$= e^{-4}$$

$$K_{24} = \exp\left(-\|(-1\ 1) - (1\ 1)\|^2\right)$$

$$= e^{-4}$$

Using these

$$\max_{\alpha_1 \ldots \alpha_4} (\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4) - \frac{1}{2}\Big[K_{11}y_1^2\alpha_1^2 + K_{22}y_2^2\alpha_2^2 + K_{33}y_3^2\alpha_3^2 + K_{44}y_4^2\alpha_4^2 + 2\big[y_1y_2\alpha_1\alpha_2K_{12} + y_2y_3\alpha_2\alpha_3K_{23} + y_3y_4\alpha_3\alpha_4K_{34} + y_4y_1\alpha_4\alpha_1K_{41} + y_1y_3\alpha_1\alpha_3K_{13} + y_2y_4\alpha_2\alpha_4K_{24}\big]\Big]$$

$$\min_{\alpha_1 \ldots \alpha_4} \sum\alpha - \frac{1}{2}\Big[1(\Sigma\alpha^2) + 2\big[e^{-4}(-\alpha_1\alpha_2$$

$\alpha_1 \ldots \alpha_4$

$- \alpha_3\alpha_4 - \alpha_1\alpha_3 - \alpha_2\alpha_4) + e^{-8}(\alpha_2\alpha_3 + \alpha_4\alpha_1)$

d - 1/2((a^2 + b^2 + c^2 + d^2)+ (2*(exp(-4)((-a*b)-(c*d) - (a*c)-(b*d)) +exp(-8)((b*c)+ (d*a))))), {a,b,c,d }]

Extended Keyboard | Upload | Examples | Random

Input interpretation:

maximize $a + b + c + d - \frac{1}{2}\left(\left(a^2 + b^2 + c^2 + d^2\right) + 2\left(\exp(-4)\left(-a\,b - c\,d - a\,c - b\,d\right) + \exp(-8)\left(b\,c + d\,a\right)\right)\right)$ for a, b, c, d

Global maxima:

(no global maxima found)

Local maximum: Exact form | More digits

$\max\left\{a + b + c + d - \frac{1}{2}\left(\left(a^2 + b^2 + c^2 + d^2\right) + 2\left(\exp(-4)\left(-a\,b - c\,d - a\,c - b\,d\right) + \exp(-8)\left(b\,c + d\,a\right)\right)\right)\right\} \approx 2.07533$ at $(a, b, c, d) \approx (1.03766, 1.03766, 1.03766, 1.03766)$

Download Page | POWERED BY THE WOLFRAM LANGUAGE

Using wolfram solver, we get α values as

$(\alpha_1\ \alpha_2\ \alpha_3\ \alpha_4) \cong (1.037, 1.037, 1.037, 1.037)$

Calculating bias

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

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

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

$$= 1.037\left[-K_{1x} + K_{2x} + K_{3x} - K_{4x} \right]$$

Using $i = 1$

$$= 1.037\left[-1 + e^{-4} + e^{-4} - e^{-8} \right]$$

$$= -0.9993 \cong 1$$

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

$$= 0$$

So separator is $= 0$

$$1.0 \cdot \left[-K_{1n} + K_{2n} + K_{3n} - K_{4n} \right] + 0$$

where

$$K_{1n} = \exp\left(- \| x_1 - x \|^2 \right)$$

svm_part_1

2.4) XOR data is given by

	①	②	y
n_1	-1	-1	-1
n_2	-1	+1	1
n_3	1	-1	1
n_4	1	1	-1

↓ data points

Kernel function for SVM is given by

$$\underset{\alpha}{\text{Max}} \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} \alpha_i y^i K(x^i, x^j) y^j \alpha_j$$

$$s.t \quad \sum_{i=1}^{M} \alpha_i y^i = 0$$

$$\forall \alpha_i \geq 0$$

Substituting, for first we find Kernel values

K is given by $(1 + x^i x^j)^2$

$$\text{Max}(\alpha_1+\alpha_2+\alpha_3+\alpha_4) - \frac{1}{2}\Big[K_{11} y_1^2 \alpha_1^2 + K_{22} y_2^2 \alpha_2^2 + K_{33} y_3^2 \alpha_3^2 + K_{44} y_4^2 \alpha_4^2 + 2 y_1 y_2 \alpha_1 \alpha_2 K_{12} + 2 y_2 y_3 \alpha_2 \alpha_3 K_{23} + 2 y_3 y_4 \alpha_3 \alpha_4 K_{34} + 2 y_4 y_1 K_{41} \alpha_4 \alpha_1 + 2 y_1 y_3 \alpha_1 \alpha_3 K_{13} + 2 y_2 y_4 \alpha_2 \alpha_4 K_{24}$$

To find values,

$$K_{11} = \left(1 + \underline{x}_1 \underline{x}_1\right)^2 = \left(1 + \begin{bmatrix} -1 \\ -1 \end{bmatrix} \cdot \begin{bmatrix} -1 & -1 \end{bmatrix}\right)^2$$

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

for all K_{22}, K_{33}, K_{44}; dot product with itself results in 2, thus $K = 9$

$$K_{12} = 1 + \left(\begin{bmatrix} -1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 1 \end{bmatrix}\right)^2 = 1 + (-1) = (1+0)^2 = 1$$

$$K_{23} = \left(1 + \left(\begin{bmatrix} -1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \end{bmatrix}\right)\right)^2 = (1-1-1)^2 = (-1)^2 = 1$$

$$K_{14} = \left(1 + \left(\begin{bmatrix} -1 \\ -1 \end{bmatrix} (1 \quad 1)\right)\right)^2 = (1+(-1)-1)^2$$

$$= 1$$

∴ all will be 1

and $y_1 = -1 \quad y_2 = 1 \quad y_3 = 1 \quad y_4 = -1$

So, substituting in equation gives (all squares are 1)

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

$$\left. + 2(-\alpha_1\alpha_2 + \alpha_2\alpha_3 - \alpha_3\alpha_4 + \alpha_4\alpha_1 - \alpha_1\alpha_3 - \alpha_2\alpha_4) \right]$$

Now we have a maximization function with $\alpha_1, \alpha_2, \alpha_3, \alpha_4$

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

$$\left. + 2(-\alpha_1\alpha_2 + \alpha_2\alpha_3 - \alpha_3\alpha_4 + \alpha_4\alpha_1 - \alpha_1\alpha_3 - \alpha_2\alpha_4) \right\}$$

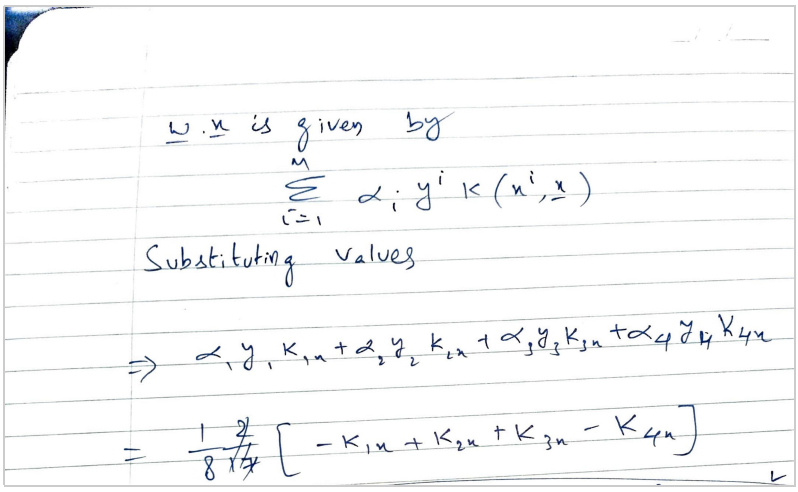
Substituting this in Wolfram Solver, I got

Hessian is at $\frac{4}{17}, \frac{1}{4}$ and

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = \left(\frac{2}{17}, \frac{2}{17}, \frac{2}{17}, \frac{2}{17} \right)$$

$$\left(\frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8} \right)$$

Using this values, we make our sign function.


$\underline{w} \cdot \underline{x}$ is given by
$\sum_{i=1}^{M} \alpha_i y^i K(x^i, \underline{x})$
Substituting values
$\Rightarrow \alpha_1 y_1 K_{1n} + \alpha_2 y_2 K_{2n} + \alpha_3 y_3 K_{3n} + \alpha_4 y_4 K_{4n}$
$= \frac{1}{8} [-K_{1n} + K_{2n} + K_{3n} - K_{4n}]$

Finding bias

$$b = y' - \underline{\omega} \cdot \underline{x}$$

Using $i=1$

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

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

$$= \frac{1}{8}(-8) = -1$$

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

bias is 0

So, seperator is

$$\Rightarrow \frac{1}{8}\left[-K_{1x} + K_{2x} + K_{3x} - K_{4x}\right] + 0$$

Here x is $(1 + x_i \cdot \underline{x})^2$