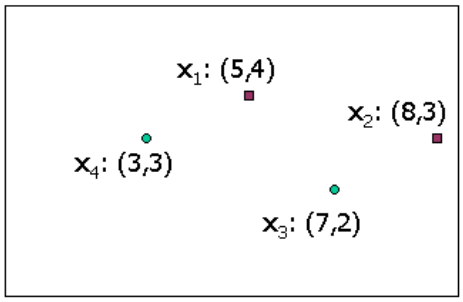
Solution of Gradiance Homework – 4

– By Anirban Haldar

Machine Learning

Some problems on Nearest Neighbours, Hyperplane, SVM etc.

1. **The figure below shows two positive points (purple squares) and two negative points (green circles):**



That is, the training data set consists of:

(x1,y1) = ((5,4),+1)
(x2,y2) = ((8,3),+1)
(x3,y3) = ((7,2),-1)
(x4,y4) = ((3,3),-1)

This data set is separable. If we call the horizontal axis of the space u and the vertical axis v, then a form of the decision boundary is v=c+au. Note that in this form, c is the intersection of the boundary with the vertical (v) axis, and a is the slope.

In all cases, if (u,v) is one of the points x1 or x2, then the point will be above the boundary (that is, if x1 = (u,v), then v ≥ c+au, and similarly for x2). Likewise, x3 and x4 must be below the boundary.

Various values of a and c could be chosen, but there are significant constraints on what a and c can be. Deduce those constraints, and then find a possible value of a and c in the list below.

a) v = 8 - u
b) v = 4 - u/9
c) v = 7 - 2u/3
d) v = 7 - 4u/7

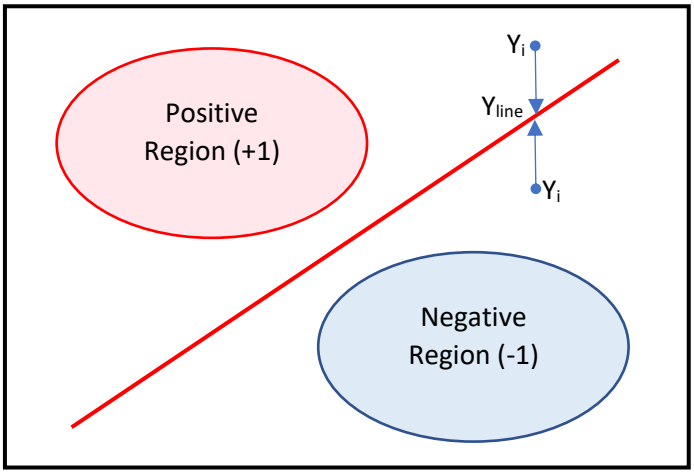
⇨ For simple binary classification, where we divide some points into two classes using a hyperplane or a line (in 2D), we call the region above the plane or line as positive class or (+1) and the points lies below the plane belongs to negative or (-1) class.

Consider a two-dimensional space, where we have a classifying line as $y_{line} = mx + c$. We have data points as, (x_1, y_1), (x_2, y_2)…(x_n, y_n). Now, the classifying line will be valid only if

For every positive point (x_i, y_i), $y_{line} = mx_i + c \rightarrow y_i - y_{line} >= 0$

And for every negative point (x_i, y_i), $y_{line} = mx_i + c \rightarrow y_i - y_{line} <= 0$

We can visualise it as,



Now, using the above logic, we will check every line equations given for every points using a code snippet as

```
points = np.array([[5, 4, +1],
                   [8, 3, +1],
                   [7, 2, -1],
                   [3, 3, -1]])

y_actual = points[:, 2]

# Option 1 : v = 8 - u
y_opt1 = []
for point in points:
    y = 8 - point[0]
    if point[1] - y > 0: y_opt1.append(1)
    else: y_opt1.append(-1)
print('Result of Option 1 :',  (y_actual == y_opt1).all())

# Option 2 : v = 4 - u/9
y_opt2 = []
for point in points:
    y = 4 - point[0] / 9
    if point[1] - y > 0: y_opt2.append(1)
    else: y_opt2.append(-1)
print('Result of Option 2 :',  (y_actual == y_opt2).all())

# Option 3 : v = 7 - 2u/3
y_opt3 = []
for point in points:
    y = 7 - 2 * point[0] / 3
    if point[1] - y > 0: y_opt3.append(1)
    else: y_opt3.append(-1)
print('Result of Option 3 :',  (y_actual == y_opt3).all())

# Option 4 : v = 7 - 4u/7
y_opt4 = []
for point in points:
    y = 7 - 4 * point[0] / 7
    if point[1] - y > 0: y_opt4.append(1)
    else: y_opt4.append(-1)
print('Result of Option 4 :',  (y_actual == y_opt4).all())
```

```
Result of Option 1 : False
Result of Option 2 : False
Result of Option 3 : True
Result of Option 4 : False
```

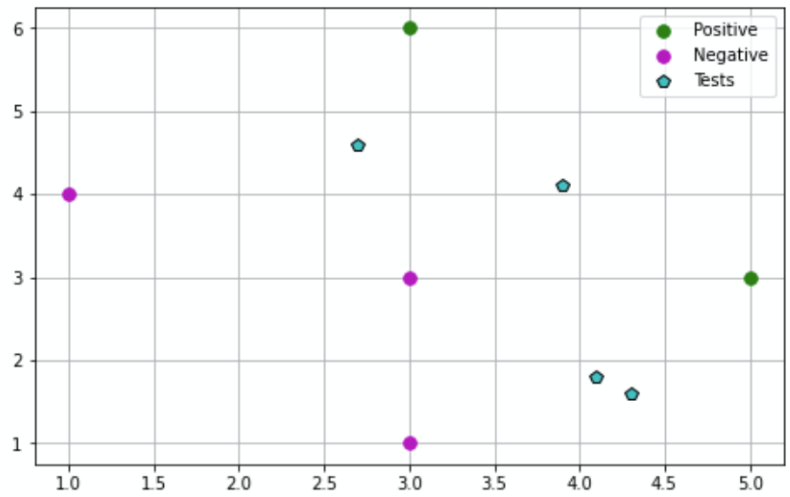
So, the correct equation is v = 7 - 2u/3

2. **Suppose our training set consists of the three negative points (1,4), (3,3), and (3,1) and the two positive points (3,6) and (5,3). If we use nearest-neighbour learning, where we classify a point to be in the class of the nearest member of the training set, what is the boundary between the positive and negative points? Identify in the list below, the point that is classified as positive.**

   a) (4.1, 1.8)
   b) (3.9, 4.1)
   c) (4.3, 1.6)
   d) (2.7, 4.6)

⇨ We can plot the points as

```
negative = np.array([[1, 4], [3, 3], [3, 1]])

positive = np.array([[3, 6], [5, 3]])

tests = np.array([[4.1, 1.8],
                  [3.9, 4.1],
                  [4.3, 1.6],
                  [2.7, 4.6]])

plt.figure(figsize=(8, 5))
plt.plot(positive[:, 0], positive[:, 1], '.g', markersize=15, label='Positive')
plt.plot(negative[:, 0], negative[:, 1], '.m', markersize=15, label='Negative')
plt.plot(tests[:, 0], tests[:, 1], 'pc', markersize=8, label='Tests', markeredgecolor='k')
plt.legend()
plt.grid()
plt.show()
```



Now, to calculate nearest neighbour, for each test point, we will calculate distances from every datapoint (Positive points and Negative points), and we choose the point with least distance as nearest neighbour. Now, the label (Positive or Negative) of the test point will be same as the label of the nearest point.

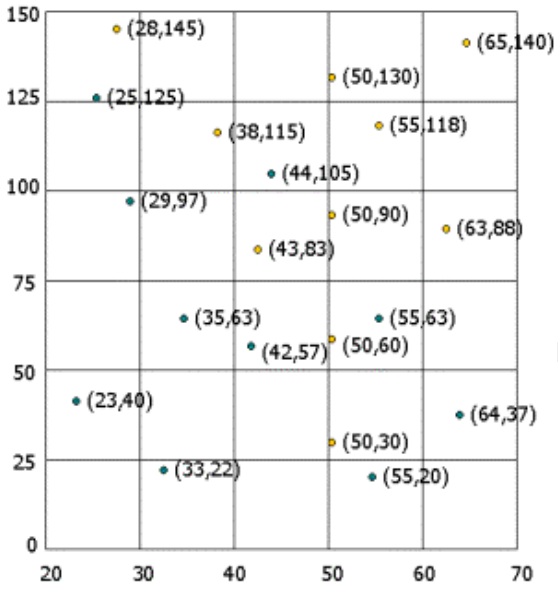
Now, to calculate the distance, we will take the help of a code snippet

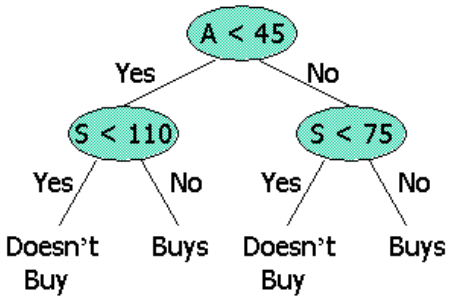
```
points = [(1, 4, 'Negative'),
          (3, 3, 'Negative'),
          (3, 1, 'Negative'),
          (3, 6, 'Positive'),
          (5, 3, 'Positive')]

tests = [(4.1, 1.8), (3.9, 4.1), (4.3, 1.6), (2.7, 4.6)]


result = []
for i in tests:
    x = []
    for j in points:
        # Calculating Euclidian Distance
        dist = ((i[0]-j[0]) ** 2 + (i[1]-j[1]) ** 2) ** 0.5
        x.append([dist, j[2]])
    # Sorting the distances to get the nearest neighbour
    x.sort()
    # Loading the label of nearest neighbour for each option
    result.append([i, x[0][1]])

result
```

```
[[(4.1, 1.8), 'Negative'],
 [(3.9, 4.1), 'Negative'],
 [(4.3, 1.6), 'Negative'],
 [(2.7, 4.6), 'Positive']]
```

So, we can see the point (2.7, 4.6) will be classified as a Positive point.

3. **Below we see a set of 20 points and a decision tree for classifying the points.**





To be precise, the 20 points represent (Age,Salary) pairs of people who do or do not buy gold jewelry. Age (abbreviated A in the decision tree) is the x-axis, and Salary (S in the tree) is the y-axis. Those that do are represented by gold points, and those that do not by green points. The 10 points of gold-jewelry buyers are:

(28,145), (38,115), (43,83), (50,130), (50,90), (50,60), (50,30), (55,118), (63,88), and (65,140).

The 10 points of those that do not buy gold jewelry are:

(23,40), (25,125), (29,97), (33,22), (35,63), (42,57), (44, 105), (55,63), (55,20), and (64,37).

Some of these points are correctly classified by the decision tree and some are not. Determine the classification of each point, and then indicate in the list below the point that is misclassified.

a) (25,125)
b) (55,63)
c) (63,88)
d) (29,97)

⇨ According to the given question, we have

Buyers = (28,145), (38,115), (43,83), (50,130), (50,90), (50,60), (50,30), (55,118), (63,88), and (65,140)

Not Buyers = (23,40), (25,125), (29,97), (33,22), (35,63), (42,57), (44, 105), (55,63), (55,20), and (64,37)

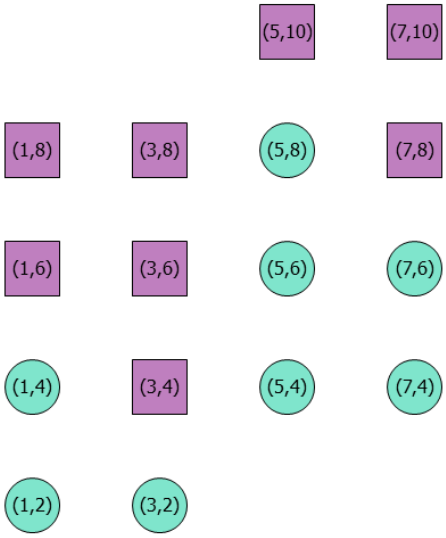
But some points are misclassified in the above two classes. To find the misclassified values, we can run the algorithm again on all 20 values using a code snippet as

```
Values = [(28,145,'Buy'), (38,115,'Buy'), (43,83,'Buy'), (50,130,'Buy'), (50,90,'Buy'),
          (50,60,'Buy'), (50,30,'Buy'), (55,118,'Buy'), (63,88,'Buy'), (65,140,'Buy'),
          (23,40,'Not Buy'), (25,125,'Not Buy'), (29,97,'Not Buy'), (33,22,'Not Buy'), (35,63,'Not Buy'),
          (42,57,'Not Buy'), (44, 105,'Not Buy'), (55,63,'Not Buy'), (55,20,'Not Buy'), (64,37,'Not Buy')]

for item in Values:
    if item[0] < 45:
        if item[1] < 110:
            print(f'For ({item[0]}, {item[1]})\tActual : Not Buy  Given : {item[2]}    \t',
                  'Verdict :', 'Correct' if item[2] == 'Not Buy' else 'Misclassified')
        else:
            print(f'For ({item[0]}, {item[1]})\tActual : Buy      Given : {item[2]}    \t',
                  'Verdict :', 'Correct' if item[2] == 'Buy' else 'Misclassified')
    else:
        if item[1] < 75:
            print(f'For ({item[0]}, {item[1]})\tActual : Not Buy  Given : {item[2]}    \t',
                  'Verdict :', 'Correct' if item[2] == 'Not Buy' else 'Misclassified')
        else:
            print(f'For ({item[0]}, {item[1]})\tActual : Buy      Given : {item[2]}    \t',
                  'Verdict :', 'Correct' if item[2] == 'Buy' else 'Misclassified')
```

```
For (28, 145)   Actual : Buy      Given : Buy          Verdict : Correct
For (38, 115)   Actual : Buy      Given : Buy          Verdict : Correct
For (43, 83)    Actual : Not Buy  Given : Buy          Verdict : Misclassified
For (50, 130)   Actual : Buy      Given : Buy          Verdict : Correct
For (50, 90)    Actual : Buy      Given : Buy          Verdict : Correct
For (50, 60)    Actual : Not Buy  Given : Buy          Verdict : Misclassified
For (50, 30)    Actual : Not Buy  Given : Buy          Verdict : Misclassified
For (55, 118)   Actual : Buy      Given : Buy          Verdict : Correct
For (63, 88)    Actual : Buy      Given : Buy          Verdict : Correct
For (65, 140)   Actual : Buy      Given : Buy          Verdict : Correct
For (23, 40)    Actual : Not Buy  Given : Not Buy      Verdict : Correct
For (25, 125)   Actual : Buy      Given : Not Buy      Verdict : Misclassified
For (29, 97)    Actual : Not Buy  Given : Not Buy      Verdict : Correct
For (33, 22)    Actual : Not Buy  Given : Not Buy      Verdict : Correct
For (35, 63)    Actual : Not Buy  Given : Not Buy      Verdict : Correct
For (42, 57)    Actual : Not Buy  Given : Not Buy      Verdict : Correct
For (44, 105)   Actual : Not Buy  Given : Not Buy      Verdict : Correct
For (55, 63)    Actual : Not Buy  Given : Not Buy      Verdict : Correct
For (55, 20)    Actual : Not Buy  Given : Not Buy      Verdict : Correct
For (64, 37)    Actual : Not Buy  Given : Not Buy      Verdict : Correct
```

So, we can see there are four misclassified points as -> (43, 83), (50, 60), (50, 30) & (25, 125)

4. Consider the following training set of 16 points. The eight purple squares are positive examples, and the eight green circles are negative examples.



We propose to use the diagonal line with slope +1 and intercept +2 as a decision boundary, with positive examples above and negative examples below. However, like any linear boundary for this training set, some examples are misclassified. We can measure the goodness of the boundary by computing all the slack variables that exceed 0, and then using them in one of several objective functions. In this problem, we shall only concern ourselves with computing the slack variables, not an objective function.

To be specific, suppose the boundary is written in the form w.x+b=0, where w = (-1,1) and b = -2. Note that we can scale the three numbers involved as we wish, and so doing changes the margin around the boundary. However, we want to consider this specific boundary and margin.

Determine the slack for each of the 16 points. Then, identify the correct statement in the list below.

a) The slack for (5,6) is 2.
b) The slack for (3,8) is 0.
c) The slack for (3,6) is 2.
d) The slack for (1,2) is 2.

⇨ According to the question, we have 16 points, out of which 8 should in the positive side of the classifying line and rest should be in the negative side. The equation of the classifying line is

$y = m * x + b$

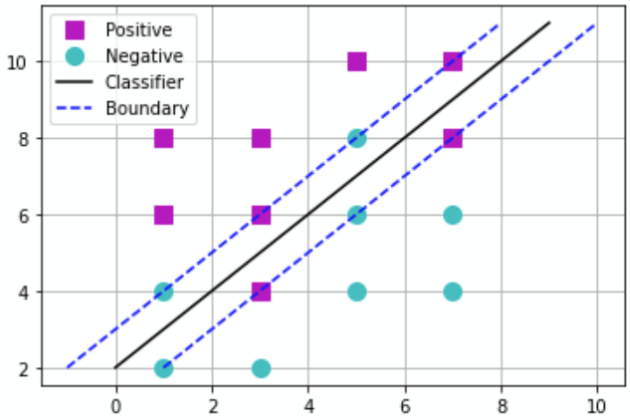
$y = 1 * x + 2$

$-x + y - 2 = 0$

So, we have weights as w = [-1, 1] and bias b = -2. Now, we consider margin = 1 on both side.

We can plot the points along with the classifying line and margins as

```
points = np.array([[ 5, 10,  1], [ 7, 10,  1], [ 1,  8,  1], [ 3,  8,  1],
                   [ 7,  8,  1], [ 1,  6,  1], [ 3,  6,  1], [ 3,  4,  1],
                   [ 5,  8, -1], [ 5,  6, -1], [ 7,  6, -1], [ 1,  4, -1],
                   [ 5,  4, -1], [ 7,  4, -1], [ 1,  2, -1], [ 3,  2, -1]])

plt.plot(points[:8, 0], points[:8, 1], 'sm', markersize=10, label='Positive')
plt.plot(points[8:, 0], points[8:, 1], 'oc', markersize=10, label='Negative')
plt.plot([0, 9], [2, 11], 'k', label='Classifier')
plt.plot([1, 10], [2, 11], '--b', label='Boundary')
plt.plot([-1, 8], [2, 11], '--b')
plt.legend()
plt.grid()
plt.show()
```



Now, here, we can see some positive points have crossed the classifier line and misclassified as negative points, and same happened with some negative points too. So, we introduce a variable named “slack variable”, which determines, how much a point is deviated from its original class.

Slack variable value == 0 → The point is in its own class and it is above or on the margin line.

0 < value <= 1 → The point is in its own class but it is located between the margin and the classifying line

value > 1 → The point lies completely outside of its own class.

Now, to calculate the value of slack variable, we need some values as, class = y_n (either +1 or -1), weights (here, w = [-1, 1]), bias (b = -2) and the points as x. The equation is as,

$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \geq 1 - \xi_n \quad \text{if} \quad y_n = +1$$
$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \leq -1 + \xi_n \quad \text{if} \quad y_n = -1$$
$$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n \quad \forall n$$

Here, ξ_n is the slack variable.

Now, we will calculate slack value for every point using a code snippet as,

```
points = np.array([[ 5, 10,  1], [ 7, 10,  1], [ 1,  8,  1], [ 3,  8,  1],
                   [ 7,  8,  1], [ 1,  6,  1], [ 3,  6,  1], [ 3,  4,  1],
                   [ 5,  8, -1], [ 5,  6, -1], [ 7,  6, -1], [ 1,  4, -1],
                   [ 5,  4, -1], [ 7,  4, -1], [ 1,  2, -1], [ 3,  2, -1]])

w = [-1, 1]  # Weights
b = -2       # Bias

slacks = []
for i in points:
    # calculating the value yn(w.x + b)
    value = i[2] * (w[0]*i[0] + w[1]*i[1] + b)
    if value > 1: slacks.append([i.tolist(), 0])
    else: slacks.append([i.tolist(), 1 - value])

print('\n'.join(map(str, slacks)))
```

```
[[5, 10, 1], 0]
[[7, 10, 1], 0]
[[1, 8, 1], 0]
[[3, 8, 1], 0]
[[7, 8, 1], 2]
[[1, 6, 1], 0]
[[3, 6, 1], 0]
[[3, 4, 1], 2]
[[5, 8, -1], 2]
[[5, 6, -1], 0]
[[7, 6, -1], 0]
[[1, 4, -1], 2]
[[5, 4, -1], 0]
[[7, 4, -1], 0]
[[1, 2, -1], 0]
[[3, 2, -1], 0]
```

So, by using slack variable values, we can see the points (7, 8), (3, 4), (5, 8) & (1, 4) have value = 2, so these points are misclassified. And the slack value of (3, 8) is 0.

5. **Suppose we use the transformation $\Phi((x,y)) = (x^2,y^2,xy)$. That is, the two-dimensional vector (x,y) is turned into a three-dimensional vector (x^2,y^2,xy). There is a kernel function K(u,v) for this transformation; the function such that $K(u,v) = \Phi(u).\Phi(v)$. The kernel function measures the similarity, in the transformed space, between the vectors u and v.**

   Let us consider the eight vectors from the original space: (1,2), (1,-2), (-1,2), (-1,-2), (2,1), (2,-1), (-2,1), and (-2,-1). Your task is to compute the kernel function for each pair of these vectors. Note that the task is simpler than it looks, because pairs of original vectors transform to the same vector, so there are only four different vectors in the transformed space.

   Now, find in the list below the pair of vectors that are MOST similar.

   a) (-1,-2) and (2,-1)
   b) (-2,1) and (-1,2)
   c) (-2,-1) and (-1,-2)
   d) (1,-2) and (-1,2)

   ⇨ Now, to solve this Question, we first have to calculate the three dimensional vector for each two dimensional data points using the transformation $\Phi((x,y)) = (x^2,y^2,xy)$ as

Φ((1, 2)) = [1, 4, 2]
Φ((1, -2)) = [1, 4, -2]
…….

Now, we have to find the similarity between all combinations of the two dimensional values using the kernel function K(u, v). The kernel function is nothing but the dot product of the two vectors as K(u,v) = Φ(u).Φ(v). Now, we are taking help of a simple program to compute similarity as

```
# Given 8 tw dimensional values
points = np.array([[ 1, 2], [ 1,-2],
                   [-1, 2], [-1,-2],
                   [ 2, 1], [ 2,-1],
                   [-2, 1], [-2,-1]])

# Here we are defining an empty matrix
similarity = np.zeros((len(points), len(points)), dtype='int8')

for i in range(len(points)):
    for j in range(len(points)):
        # Now, for each combination, we are calculating the three dimenaional vectors
        vector_1 = np.array([points[i, 0] ** 2, points[i, 1] ** 2, points[i, 0] * points[i, 1]])
        vector_2 = np.array([points[j, 0] ** 2, points[j, 1] ** 2, points[j, 0] * points[j, 1]])
        # Finally, we are loading the value of Dot product of vectors
        similarity[i, j] = vector_1.dot(vector_2)

pd.DataFrame(similarity,
             index=list(map(str, points.tolist())),
             columns=list(map(str, points.tolist())))
```

	[1, 2]	[1, -2]	[-1, 2]	[-1, -2]	[2, 1]	[2, -1]	[-2, 1]	[-2, -1]
[1, 2]	21	13	13	21	12	4	4	12
[1, -2]	13	21	21	13	4	12	12	4
[-1, 2]	13	21	21	13	4	12	12	4
[-1, -2]	21	13	13	21	12	4	4	12
[2, 1]	12	4	4	12	21	13	13	21
[2, -1]	4	12	12	4	13	21	21	13
[-2, 1]	4	12	12	4	13	21	21	13
[-2, -1]	12	4	4	12	21	13	13	21

Now, we are checking each option to find MOST similarity

```
print('Similarity of (-1,-2) and ( 2,-1) is :', similarity[3, 5])
print('Similarity of (-2, 1) and (-1, 2) is :', similarity[6, 2])
print('Similarity of (-2,-1) and (-1,-2) is :', similarity[7, 3])
print('Similarity of ( 1,-2) and (-1, 2) is :', similarity[1, 2])
```

```
Similarity of (-1,-2) and ( 2,-1) is : 4
Similarity of (-2, 1) and (-1, 2) is : 12
Similarity of (-2,-1) and (-1,-2) is : 12
Similarity of ( 1,-2) and (-1, 2) is : 21
```

So, most similar pair is (1,-2) and (-1,2)