CED16I006  B S CHANDRASHEKAR

# Reading Assignment

I a)
Pseudocode for decision tree classifier
Gigni index-m tells how is the randomness of the variable chosen to split the data
    *so its advised to take the bdata with lower gini index

Algorithm

1Convert the nominal data into ordinal scale
2 calculate the gini index for each variable
3choose the variable with least gini index and make it as root
4 recursive call or loop
5 construct the other branches of the tree based on the decrease in gini index
6classify the leaves of the tree as yes/no
7use the constructed tree to iterate through and classify the given data

Code -rough

```
gini(x), where x varies from 0 to n-1th values given in the table
G- be a biniry graph
gini.sort()
root=gini[0]
i=0
while(i<n)
        branch[2*r]=gini[i]
        branch[2*r+1]=gini[i+1]
Prune_empty nodes()
assign_yes/no_leaves()

traverse_graph(G,k)
```

Ib) NAIVE BASEYAN CLASSIFIER
logic:
1find the posteriory probability and the priory probability for both yes and no occurence of the event .

Use the baseyan theorem to find the occurence of a event to pridict the yes or no of a event
Laplace smoothing : Its to solve problems of zero probability By applying this method, prior probability and conditional probability can be written as: K denotes the number of different values in y and A denotes the number of different values in aj.

Procedure:
Load the dataset
Store the feature matrix of x and y
Splitting the testing model into training and testing
Train the model on dataset
Make the prediction
Compare the predictions

Algorithm:
```
Store the condition in C()
Class probabiliy -c
Conditional probablity -cn
formula=c*cn
If formula yes>no:
        Pridict yes
else :
        no>yes :
        Predict no
```

NEURAL NETWORK CLASSIFIER
Toplogy: hidden layer , input layer,output layer , target values , learning rate , bias ,
- hypothesis weights,

Input layer: its  the layer of classifier which take the array of input
Output layer: its the layer of the classifier which gives the output
Target values;expected output out of the layer
Learning rate : rate at which weights are changed
Bias; its a constant added during each computation
Advantages : ANNs have the ability to learn and model non-linear and complex relationships , which is really important because in real-life, many of the relationships between inputs and outputs are non-linear as well as complex.ANNs have the ability to learn and model non-linear and complex relationships , which is really important because in real-life, many of the relationships between inputs and outputs are non-linear as well as complex.

Disadvantages:
Black Box.,Duration of Development. ,Amount of Data,Computationally Expensive.

b=a1*w1+a2*w2+a3*w3+bias -------------------------(1)

<u>Example</u> :   Activation function:

A = 1/(1 + $e_{-x}$)

Hypothese y=3.x

| Input | output | output before back propogation | squared error |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 2 | 3 | 4 |
| 2 | 4 | 6 | 16 |

Check derivative

+ - increase weight
- -decrease weight
0 - do nothing

Backpropogation starts:
Decomposing the deriavtive to calculatable smaller derivatives= auto differnciation
Rate 60x, loss=30units  so reduce the weight by 0.5
New weight =old weight-derivative*learning rate
Iterate untill convergence

```
h(X) = W0.X0 + W1.X1 + W2.X2
h(X) = sigma(W.X) for all (W, X)
And X1 | X2 | Y
     0  | 0  | 0
     0  | 1  | 0
     1  | 0  | 0
     1  | 1  | 1
or  X1 | X2 | Y
    0  | 0  | 0
    0  | 1  | 1
    1  | 0  | 1
    1  | 1  | 1
```

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log((h_\Theta(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2$$

Unit Z1:

```
      h(x) = W0.X0 + W1.X1 + W2.X2= 1 . 0 + 1 . 0 + 1 . 0 = 1 =a
z = f(a) = a   =>   z = f(1) = 1

   delta_0 = w . delta_1 . f'(z)
W := W - alpha . J'(W)
J'(W) = Z . delta
Consider exor gate for calculation
nit Z2:      h(x) = W0.X0 + W1.X1 + W2.X2    = 1 . 1 + 1 . 0 + 1 . 0=
1 = a
   z = f(a) = a   =>   z = f(1) = 1
Unit Z3:
      h(x) = W0.X0 + W1.X1 + W2.X2           = 1 . 1 + 1 . 0 + 1 . 0=
1 =   z = f(a) = a   =>   z = f(1) = 1
Unit D0:
      h(x) = W0.Z0 + W1.Z1 + W2.Z2 + W3.Z3           = 1 . 1 + 1 . 1
+ 1 . 1 + 1 . 1= 4 = z = f(a) = a   =>   z = f(4) = 4

delta_D0 = total_loss = -4

W11 := 1.4
W21 := 1.4
W31 := 1.4 after calculation from the above formula
V01 := 1.4
V02 := 1.4
V03 := 1.4
```

## NEAREST NEIGHBOUR CLASSIFIER

Kth nearest neighbour

1Given the coordinates of all the points

2Both different category of points and necessary point

3Sort all the points based on the ditace from the point c

4Based on the maximum occurence of points among the top k points

5We decide the category of the point

II)

to maximize the function f, you define a new function F such that F=1/(1+f).

Herer  $f(x)= x^3-2x^2+x$ within a range of (0,31).

```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    return (1/(x*x*x+2*x*x+x))



x = np.arange(-10, 10, 0.1)
plt.plot(x, f(x))
plt.show()

from scipy import optimize

# The default (Nelder Mead)
print(optimize.minimize(f, x0=0))
print(optimize.minimize(f, x0=0, method="L-BFGS-B"))
.
  if (phi_aj > phi0 + c1*a_j*derphi0) or (phi_aj >= phi_lo):
      fun: array([inf])
 hess_inv: array([[1]])
      jac: array([-inf])
  message: 'Desired error not necessarily achieved due to precision
loss.'
     nfev: 321
      nit: 0
     njev: 103
   status: 2
  success: False
        x: array([0.])
genetic.py:5: RuntimeWarning: divide by zero encountered in
true_divide
  return (1/(x*x*x+2*x*x+x))
      fun: array([inf])
 hess_inv: <1x1 LbfgsInvHessProduct with dtype=float64>
      jac: array([-inf])
  message: b'ABNORMAL_TERMINATION_IN_LNSRCH'
     nfev: 42
      nit: 0
   status: 2
  success: False
        x: array([0.])
```

From the graph the point is at 0.0
Selection:selectin of fittest set for mating
Crossover:exchange of chromosomes while mating
Mutation: change in the species with higher survival rate

III) BUCKET BRIDGE CLASSIFIER
In the bucket bridge classifier there two things
1auction and claearing house
Based on the auction the objects will be selecetd for operation
Using clearhouse segraegation among objects is done
Using auction
Strenght of the classifier at i is by
si(t+1)=si(t)-pi(t)-Ti(t)+Ri(t)

Gain coffecient
Bss=Cbid/k *Rss=Cbid/Cbid+Cixx

Strenght at n time
S(n)=(1-k)nS(0)+ sumR(j)(1-K)n-j-1

IV) CONFUSION MATRIX **:** A confusion matrix is a summary of prediction results on a classification problem

- Positive (P) : Observation is positive (for example: is an apple).
- Negative (N) : Observation is not positive (for example: is not an apple).
- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.
- **Classification Rate/Accuracy:**Classification Rate or Accuracy is given by the relation:
- **Recall:**

$$\textit{Recall} = \frac{TP}{TP + FN}$$

- 
- Recall can be defined as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized (a small number of FN).**Precision:**

$$\textit{Precision} = \frac{TP}{TP + FP}$$

- 
- To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labelled as positive is

$$\textit{F - measure} = \frac{2*Recall*Precision}{Recall + Precision}$$

indeed positive (a small number of FP).

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

VI)a) K  MEANS CLUSTERING  :: Psudocode for the algorithm

- Initialize the k with the random values
- For a given number of iterations
- Iterate through the items :
- Find the mena closeset to the item

- Assign item to the mean
- Update mean

```
def CalculateMeans(k,items,maxIterations=100000):

    # Find the minima and maxima for columns
    cMin, cMax = FindColMinMax(items);

    # Initialize means at random points
    means = InitializeMeans(items,k,cMin,cMax);

    # Initialize clusters, the array to hold
    # the number of items in a class
    clusterSizes= [0 for i in range(len(means))];

    # An array to hold the cluster an item is in
    belongsTo = [0 for i in range(len(items))];

    # Calculate means
    for e in range(maxIterations):

        # If no change of cluster occurs, halt
        noChange = True;
        for i in range(len(items)):

            item = items[i];

            # Classify item into a cluster and update the
            # corresponding means.
            index = Classify(means,item);

            clusterSizes[index] += 1;
            cSize = clusterSizes[index];
            means[index] = UpdateMean(cSize,means[index],item);

            # Item changed cluster
            if(index != belongsTo[i]):
                noChange = False;

            belongsTo[i] = index;

        # Nothing changed, return
        if (noChange):
            break;

    return means;
```

UPDATE MEAN

```
def UpdateMean(n,mean,item):
    for i in range(len(mean)):
        m = mean[i];
        m = (m*(n-1)+item[i])/float(n);
        mean[i] = round(m, 3);

    return mean;
```

B)**Algorithm:**
*1. Initialize: select k random points out of the n data points as the medoids.*

*2. Associate each data point to the closest medoid by using any common distance metric methods.*

*3. While the cost decreases:*

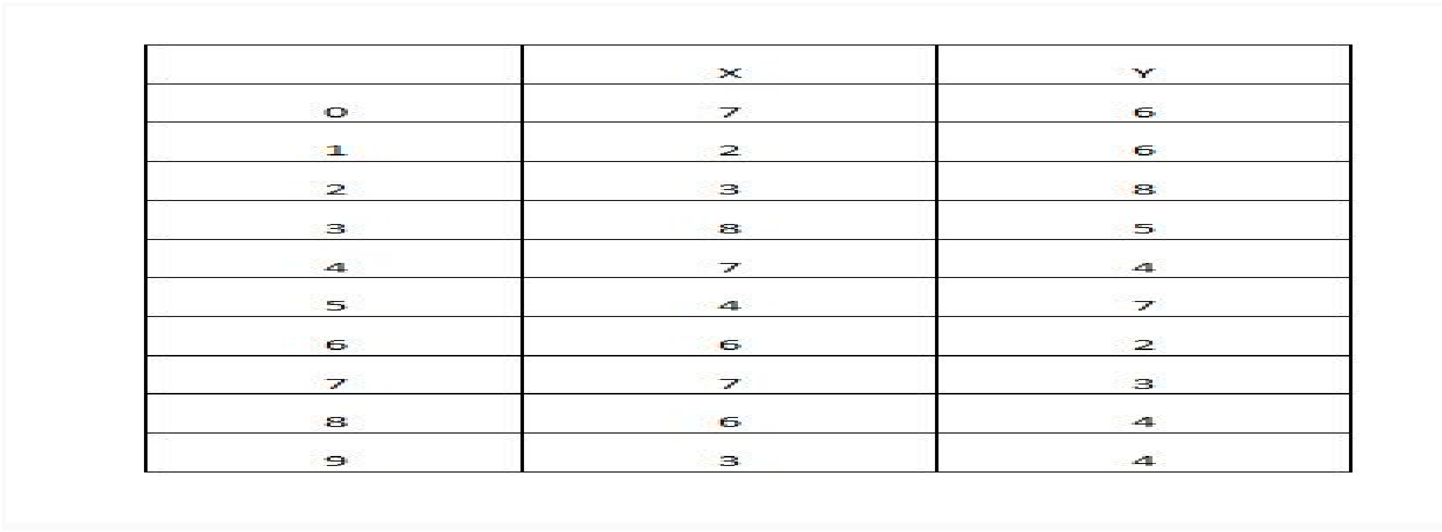
*For each medoid m, for each data o point which is not a medoid:*

*1. Swap m and o, associate each data point to the closest medoid, recompute the cost.*

*2. If the total cost is more than that in the previous step, undo the swap.*

```
def UpdateMediods(n,mean,item):
    for i in range(len(mediods)):
        m = mediods[i];
        m = (m*(n-1)+item[i])/float(n);
        mediods[i] = round(m, 3);

    return mean;
```



| | X | Y |
|---|---|---|
| 0 | 7 | 6 |
| 1 | 2 | 6 |
| 2 | 3 | 8 |
| 3 | 8 | 5 |
| 4 | 7 | 4 |
| 5 | 4 | 7 |
| 6 | 6 | 2 |
| 7 | 7 | 3 |
| 8 | 6 | 4 |
| 9 | 3 | 4 |

**Step #1:** k = 2

Let the randomly selected 2 medoids be C1 –(3, 4) and C2 –(7, 4).

**Step #2:** Calculating cost.

Each point is assigned to the cluster of that medoid whose dissimilarity is less.

The points 1, 2, 5 go to cluster C1 and 0, 3, 6, 7, 8 go to cluster C2.

The cost C = (3 + 4 + 4) + (3 + 1 + 1 + 2 + 2)

**C = 20**

**Step #3:** Now randomly select one non-medoid point and recalculate the cost.

Let the randomly selected point be (7, 3). The dissimilarity of each non-medoid point with the medoids – C1 (3, 4) and C2 (7, 3) is calculated and tabulated.

Each point is assigned to that cluster whose dissimilarity is less. So, the points 1, 2, 5 go to cluster C1 and 0, 3, 6, 7, 8 go to cluster C2.

The cost C = (3 + 4 + 4) + (2 + 2 + 1 + 3 + 3)

**C = 22**

Swap Cost = Present Cost – Previous Cost

= 22 – 20 = **2 >0**

As the swap cost is not less than zero, we undo the swap. Hence (3, 4) and (7, 4) are the final medoids. The clustering would be in the following way

C)**Agglomerative Clustering**

```
given a dataset (d1, d2, d3, ....dN) of size N
# compute the distance matrix
for i=1 to N:
   # as the distance matrix is symmetric about
   # the primary diagonal so we compute only lower
   # part of the primary diagonal
   for j=1 to i:
      dis_mat[i][j] = distance[di, dj]
each data point is a singleton cluster
repeat
   merge the two cluster having minimum distance
   update the distance matrix
untill only a single cluster remains
```

```
from sklearn.cluster import AgglomerativeClustering
import numpy as np

# randomly chosen dataset
X = np.array([[1, 2], [1, 4], [1, 0],
              [4, 2], [4, 4], [4, 0]])

# here we need to mention the number of clusters
# otherwise the result will be a single cluster
# containing all the data
clustering = AgglomerativeClustering(n_clusters = 2).fit(X)

# print the class labels
print(clustering.labels_)
```

V) DATA

| | I | think | therefore | am | Can | you | don’t | know | who |
|---|---|---|---|---|---|---|---|---|---|

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| “I think, therefore I am” | 2 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| “Can you think?” | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| “I don’t think, therefore I don’t know who I am” | 3 | 1 | 1 | 1 | 0 | 0 | 2 | 1 | 1 |

**Cosine similarity** is measure of number of common words in two observations, scaled by length of sentences. Cosine distance is computed as

$$d_{ij} = 1 - \frac{\sum_{w}(x_{iw}x_{jw})}{\sqrt{\sum_{w}(x_{iw})^2}\sqrt{\sum_{w}(x_{jw})^2}} \qquad d_{12} = 1 - \frac{1}{\sqrt{2^2+1+1+1}\sqrt{1+1+1}} = 1 - \frac{1}{\sqrt{7}\sqrt{3}} = 0.782$$

**Tanimato coefficien**t extends idea of Cosine distance and changes the normalization figure in denominator. Tanimato distance is computed as

$$d_{ij} = 1 - \frac{\sum_{w}(x_{iw}x_{jw})}{\sum_{w}(x_{iw})^2 + \sum_{w}(x_{jw})^2 - \sum_{w}(x_{iw}x_{jw})}$$

$$d_{12} = 1 - \frac{1}{(2^2+1+1+1)+(1+1+1)-1} = 1 - \frac{1}{7+3-1} = 0.889$$

and $d_{13} = 1 - \frac{9}{7+18-9} = 0.438$ and $d_{23} = 1 - \frac{1}{3+18-1} = 0.95$

Tanimato distance between sentence 1 and 2 is

**Jaccard distance** is one minus Jaccard similarity, which is number of common words in two sentences divided by total number of unique words. This is particularly popular distance metric for text comparison, because it is fairly fast to compute as we are essentially counting number of common and unique words without complex

$$d_{ij} = 1 - \frac{|x \cap y|}{|x \cup y|} = 1 - \frac{|x \cap y|}{|x| + |y| - |x \cap y|}$$

mathematical computation.

$$d_{12} = 1 - \frac{1}{6} = 0.833$$

Note that Jaccard's metric ignores number of times each word occurs and essentially treats $x_{iw}$ as binary indicator. $d_{13} = 1 - \frac{4}{7} = 0.429$ and $d_{23} = 1 - \frac{1}{9} = 0.889$.

**Sorensen–Dice coefficient** is variation of Jaccard's and computed as

$$d_{ij} = 1 - \frac{2|x \cap y|}{|x| + |y|}$$

which makes $d_{12} = 1 - \frac{2*1}{4+3} = 0.714$ for us.

**Hamming distance** is most commonly used for equal length documents, and is equal to number of places changes are required to convert one document into another. This is computationally expensive metric as it also takes into account order of words. Sentence 1 can be converted into sentence 2 in following series of operations –

| **Action** | **Sentence** | **Step number** |
|---|---|---|
| – | I think therefore I am | 0 |
| Drop first "I" | think therefore I am | 1 |

| | | |
|---|---|---|
| Drop “therefore” | think I am | 2 |
| Drop second “I” | think am | 3 |
| Drop “am” | think | 4 |
| Insert “can” | Can think | 5 |
| Insert “you” | Can you think | 6 |

Thus, hamming distance d12 = 6, and d13 = 4 (insert “don’t”, “don’t”, “know”, “who”) and d23 = 11 (drop “can”, “you”; insert “I”, “don’t”, “therefore”, “I”, “don’t”, “know”, “who”, “I”, “am”).

**String Edit distance** is generalization of Hamming distance and is O(n2)computational operation. While Hamming distance permits only ‘insert’ and ‘drop’ operation, String Edit distance permits third operation of ‘replace’ which is equivalent to one ‘insert’ and ‘drop’ combined. Further, Hamming distance computation weighs each operation (insert or drop) equally, String Edit distance can weight insert, replace, and delete separately, in counting number of operations. However, commonly applied variant works with equal weight for all three actions. String Edit distance between Sentence 1 and sentence 2 is 5 as shown in table below.

| **Action** | **Sentence** | **Step number** |
|---|---|---|
| – | I think therefore I am | 0 |
| Replace first “I” with “you” | you think therefore I am | 1 |
| Drop “therefore” | you think I am | 2 |
| Drop second “I” | you think am | 3 |
| Drop “am” | you think | 4 |

|  |  |  |
|---|---|---|
| Insert “can” | Can you think | 5 |

**Metrics intended for real-valued vector spaces:**

| identifier | class name | args | distance function |
|---|---|---|---|
| **“euclidean”** | **EuclideanDistance** | ● | `sqrt(sum((x - y)^2))` |
| **“manhattan”** | **ManhattanDistance** | ● | `sum(\|x - y\|)` |
| **“chebyshev”** | **ChebyshevDistance** | ● | `max(\|x - y\|)` |
| **“minkowski”** | **MinkowskiDistance** | **p** | `sum(\|x - y\|^p)^(1/p)` |
| **“wminkowski”** | **WMinkowskiDistance** | **p, w** | `sum(\|w * (x - y)\|^p)^(1/p)` |
| **“seuclidean”** | **SEuclideanDistance** | **V** | `sqrt(sum((x - y)^2 / V))` |
| **“mahalanobis”** | **MahalanobisDistance** | **V or VI** | `sqrt((x - y)' V^-1 (x - y))` |

```
Import math

x=input(“enter the first number ”)  ;y=input(“enter the second number ”)
```

```
print(sqrt(sm((x - y)^2)))
```

```
print(sum(|x - y|))
```

```
print(max(|x - y|))
```

```
print(sum(|x - y|^p)^(1/p))
```

```
print(sum(|w * (x - y)|^p)^(1/p)))
```