CSE-4502/5717 Big Data Analytics

Written Assignment 2 (total 100 marks)

1. (Total 10 marks) **PCA**

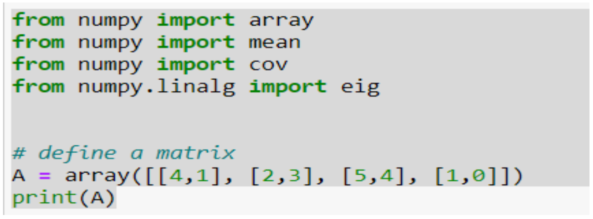
   (a) (5 marks) Consider the following matrix, representing four sample points $X \in R^2$.

$$X = \begin{bmatrix} 4 & 1 \\ 2 & 3 \\ 5 & 4 \\ 1 & 0 \end{bmatrix}$$

We want to represent the data in only one dimension, so we turn to PCA. Please compute the unit-length principal component directions of X, and state which one the PCA algorithm would choose if you request just one principal component. (Suggested steps: center the data, calculate the sample covariance matrix, calculate the eigenvectors and eigenvalues, identify the principal component; please provide the calculation process).

Answer:

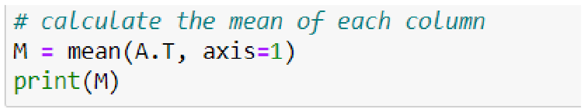
The data points represented in form of matrix =



```
from numpy import array
from numpy import mean
from numpy import cov
from numpy.linalg import eig


# define a matrix
A = array([[4,1], [2,3], [5,4], [1,0]])
print(A)
```

```
[[4 1]
 [2 3]
 [5 4]
 [1 0]]
```

Calculating mean of each column of given matrix =



```
# calculate the mean of each column
M = mean(A.T, axis=1)
print(M)
```

```
[3. 2.]
```

Next, we need to center the values in each column by subtracting the mean column value=

```
# center columns by subtracting column means
C = A - M
print(C)
```

```
[[ 1. -1.]
 [-1.  1.]
 [ 2.  2.]
 [-2. -2.]]
```

The next step is to calculate the covariance matrix of the centered matrix C =

```
# calculate covariance matrix of centered matrix
V = cov(C.T)
print(V)
```

```
[[3.33333333 2.        ]
 [2.         3.33333333]]
```

Finally, we calculate the eigen decomposition of the covariance matrix V. This results in a list of eigenvalues and a list of eigenvectors.

```
# eigendecomposition of covariance matrix
values, vectors = eig(V)
print('eigen vectors = ' , vectors, '\n')
print('eigen values = ' , values)
```

```
eigen vectors =

 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]

eigen values =

[5.33333333 1.33333333]
```

Based on our values from above steps,
let’s suppose that our data set is 2-dimensional with 2 variables x,y and that the eigenvectors and eigenvalues of t he covariance matrix are as follows:

v1 = [0.70710678 -0.70710678] **λ1 = 5.33**

v2 = [0.70710678 0.70710678] λ2 = 1.33

If we rank the eigenvalues in descending order, we get λ1>λ2, which means that the eigenvector that corresponds to the first principal component (PC1) is v1 and the one that corresponds to the second component (PC2) is v2.

The eigenvectors and eigenvalues are taken as the principal components and singular values and used to project the original data as,

```
[[ 2.22044605e-16 -1.41421356e+00]
 [-2.22044605e-16  1.41421356e+00]
 [ 2.82842712e+00  4.44089210e-16]
 [-2.82842712e+00 -4.44089210e-16]]
```

(b) (5 marks) Given the following 3D input data, $X \in R^3$.

$$X = \begin{bmatrix} 1 & 1 & 9 \\ 2 & 4 & 6 \\ 3 & 7 & 4 \\ 4 & 11 & 4 \\ 5 & 9 & 2 \end{bmatrix}$$

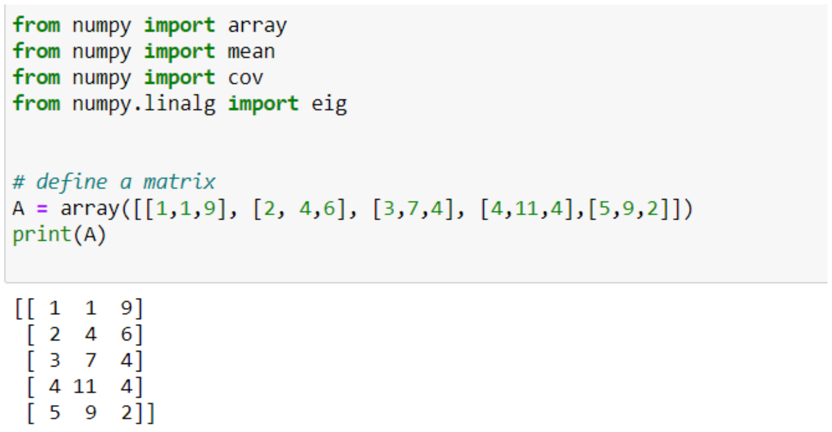
Please compute the principal component which corresponds to the largest eigenvalue. (Suggested steps: center the data, calculate the sample covariance matrix, calculate the eigenvectors and eigenvalues, identify the principal component; please provide the calculation process).

Note: sample covariance matrix is based on:

$$cov_{x,y} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

Answer:

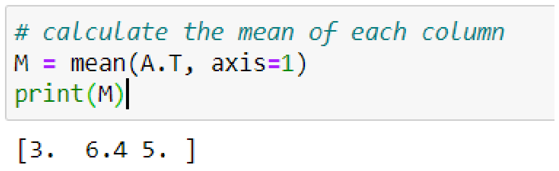
The data points represented in form of matrix =



```
from numpy import array
from numpy import mean
from numpy import cov
from numpy.linalg import eig


# define a matrix
A = array([[1,1,9], [2, 4,6], [3,7,4], [4,11,4],[5,9,2]])
print(A)
```

```
[[ 1  1  9]
 [ 2  4  6]
 [ 3  7  4]
 [ 4 11  4]
 [ 5  9  2]]
```

Calculating mean of each column of given matrix =



```
# calculate the mean of each column
M = mean(A.T, axis=1)
print(M)
```

```
[3.  6.4 5. ]
```

Next, we need to center the values in each column by subtracting the mean column value=

```
# center columns by subtracting column means
C = A - M
print(C)
```

```
[[-2.  -5.4  4. ]
 [-1.  -2.4  1. ]
 [ 0.   0.6 -1. ]
 [ 1.   4.6 -1. ]
 [ 2.   2.6 -3. ]]
```

The next step is to calculate the covariance matrix of the centered matrix C =

```
# calculate covariance matrix of centered matrix
V = cov(C.T)
print(V)
```

```
[[ 2.5   5.75 -4.  ]
 [ 5.75 15.8  -9.25]
 [-4.   -9.25  7.  ]]
```

Finally, we calculate the eigen decomposition of the covariance matrix V. This results in a list of eigenvalues and a list of eigenvectors.

```
# eigendecomposition of covariance matrix
values, vectors = eig(V)
print('eigen vectors = ' , vectors, '\n')
print('eigen values = ' , values)
```

```
eigen vectors =  [[ 0.31047214  0.91384283 -0.26172187]
 [ 0.80139329 -0.10354986  0.58910629]
 [-0.5112493   0.39264324  0.76449686]]

eigen values =  [23.92868148  0.12980618  1.24151234]
```

Based on our values from above steps,
the eigenvectors and eigenvalues of the covariance matrix are as follows(in descending order):

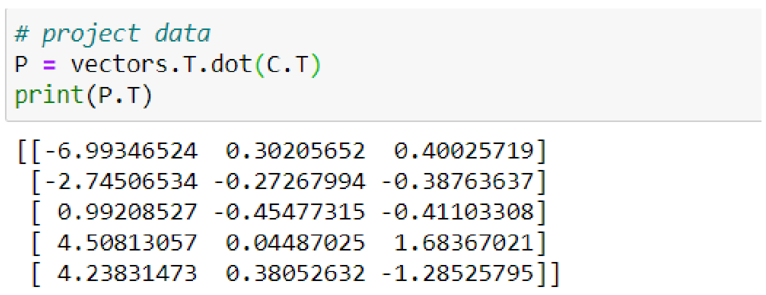
v1 = [0.31047214 0.91384283 -0.26172187]   λ1 = 23.92

v2 = [-0.5112493   0.39264324 0.76449686]   λ2 = 1.24

v3 = [0.80139329 -0.10354986 0.58910629]   λ3 = 0.12

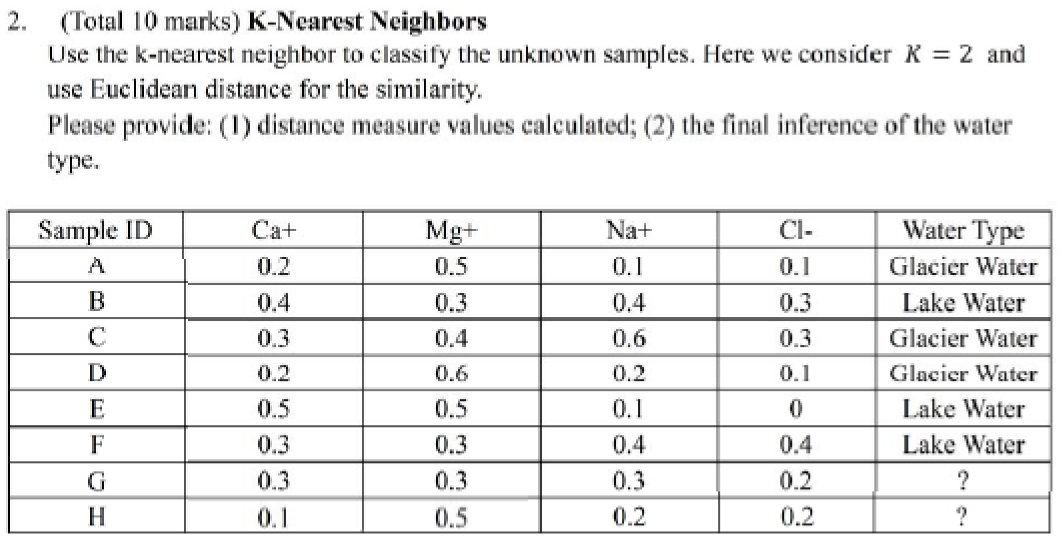
If we rank the eigenvalues in descending order, we get λ1>λ2> λ3, which means that the eigenvector that corresponds to the first principal component (PC1) is v1 and the one that corresponds to the second component (PC2) is v2 and the one that corresponds to the third component (PC3) is v3.

The eigenvectors and eigenvalues are taken as the principal components and singular values and used to project the original data as,



```
# project data
P = vectors.T.dot(C.T)
print(P.T)
```

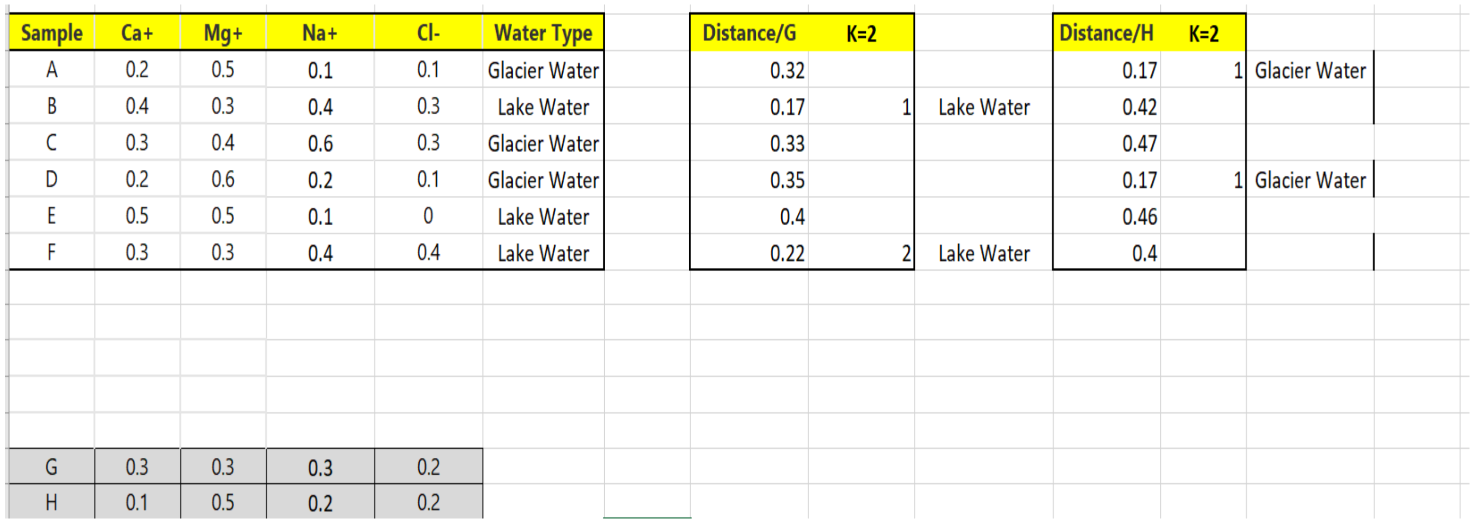
```
[[-6.99346524  0.30205652  0.40025719]
 [-2.74506534 -0.27267994 -0.38763637]
 [ 0.99208527 -0.45477315 -0.41103308]
 [ 4.50813057  0.04487025  1.68367021]
 [ 4.23831473  0.38052632 -1.28525795]]
```



2. (Total 10 marks) **K-Nearest Neighbors**
Use the k-nearest neighbor to classify the unknown samples. Here we consider $K = 2$ and use Euclidean distance for the similarity.
Please provide: (1) distance measure values calculated; (2) the final inference of the water type.

Sample ID	Ca+	Mg+	Na+	Cl-	Water Type
A	0.2	0.5	0.1	0.1	Glacier Water
B	0.4	0.3	0.4	0.3	Lake Water
C	0.3	0.4	0.6	0.3	Glacier Water
D	0.2	0.6	0.2	0.1	Glacier Water
E	0.5	0.5	0.1	0	Lake Water
F	0.3	0.3	0.4	0.4	Lake Water
G	0.3	0.3	0.3	0.2	?
H	0.1	0.5	0.2	0.2	?

Answer:

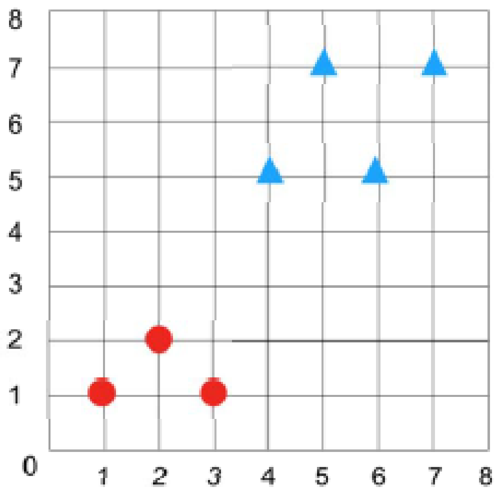


Sample	Ca+	Mg+	Na+	Cl-	Water Type		Distance/G	K=2		Distance/H	K=2	
A	0.2	0.5	0.1	0.1	Glacier Water		0.32			0.17	1	Glacier Water
B	0.4	0.3	0.4	0.3	Lake Water		0.17	1	Lake Water	0.42		
C	0.3	0.4	0.6	0.3	Glacier Water		0.33			0.47		
D	0.2	0.6	0.2	0.1	Glacier Water		0.35			0.17	1	Glacier Water
E	0.5	0.5	0.1	0	Lake Water		0.4			0.46		
F	0.3	0.3	0.4	0.4	Lake Water		0.22	2	Lake Water	0.4		
G	0.3	0.3	0.3	0.2								
H	0.1	0.5	0.2	0.2								

I used Excel for calculation of Euclidian distances. Finally, the water type inference is shown in table above. G is classified as Lake water and H is classified as Glacier water.

3. (Total 20 marks) **SVM**

(a) (12 marks) Consider the following training points. Circles are classified as positive examples with label + 1 and triangles are classified as negative examples with label −1.



i) Which points are the support vectors?

ii) If we add the sample point x = [5, 1] with label -1 (triangle) to the training set, which points are the support vectors?

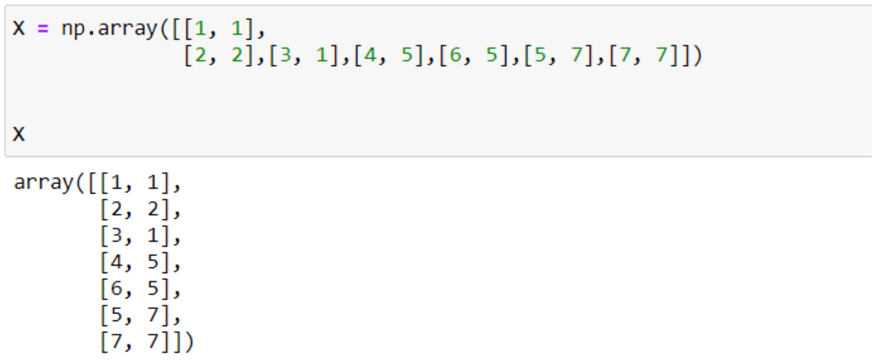
iii) Recall that an SVM has the form of $y_i(w^T \cdot X_i + b) \geq 1$, where y_i's are the labels and X_i's are the training data points. What is the geometric relationship between the weight vector w and the decision boundary of SVM?

Answer(a):

ID	**X1**	**X2**	**y**
1	**1**	**1**	**1**
2	**2**	**2**	**1**
3	**3**	**1**	**1**
4	**4**	**5**	**-1**
5	**6**	**5**	**-1**
6	**5**	**7**	**-1**

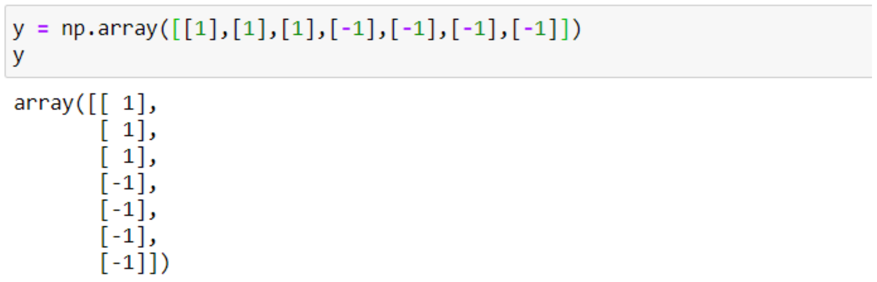
7	**7**	**7**	**-1**

(i) Creating numpy array of given data points:



```
X = np.array([[1, 1],
              [2, 2],[3, 1],[4, 5],[6, 5],[5, 7],[7, 7]])


X
```

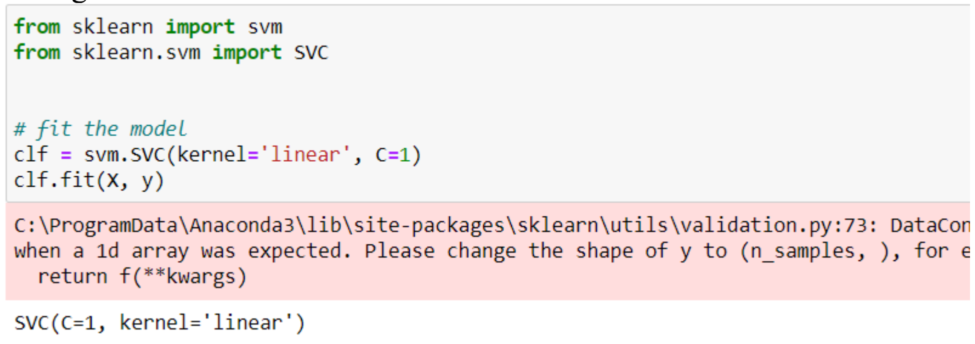
```
array([[1, 1],
       [2, 2],
       [3, 1],
       [4, 5],
       [6, 5],
       [5, 7],
       [7, 7]])
```



```
y = np.array([[1],[1],[1],[-1],[-1],[-1],[-1]])
y
```

```
array([[ 1],
       [ 1],
       [ 1],
       [-1],
       [-1],
       [-1],
       [-1]])
```

Fitting SVM model:



```
from sklearn import svm
from sklearn.svm import SVC


# fit the model
clf = svm.SVC(kernel='linear', C=1)
clf.fit(X, y)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:73: DataCon
when a 1d array was expected. Please change the shape of y to (n_samples, ), for e
  return f(**kwargs)
```

```
SVC(C=1, kernel='linear')
```

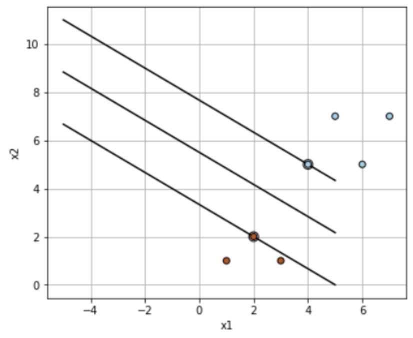
Getting margin and separating hyperplane:

```python
# get the separating hyperplane
w = clf.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(-5, 5)
yy = a * xx - (clf.intercept_[0]) / w[1]
```

```python
margin = 1 / np.sqrt(np.sum(clf.coef_ ** 2))
yy_down = yy - np.sqrt(1 + a ** 2) * margin
yy_up = yy + np.sqrt(1 + a ** 2) * margin
```

Plotting the points with hyperplanes and given support vectors:

```python
plt.figure(1, figsize=(6, 5))
plt.clf()
plt.plot(xx, yy, "k-")
plt.plot(xx, yy_down, "k-")
plt.plot(xx, yy_up, "k-")


plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=80,
 facecolors="none", zorder=10, edgecolors="k")
plt.scatter(X[:, 0], X[:, 1], c=y, zorder=10, cmap=plt.cm.Paired,
 edgecolors="k")
plt.xlabel("x1")
plt.ylabel("x2")
plt.grid()
plt.show()
```



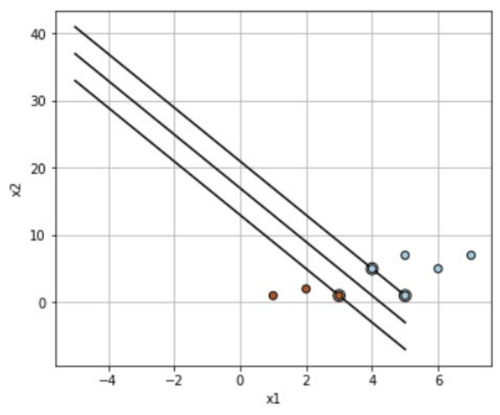
The points that are falling on hyperplanes and are support vectors are, [2 2] & [4 5].

```
: clf.support_vectors_

: array([[4., 5.],
         [2., 2.]])
```

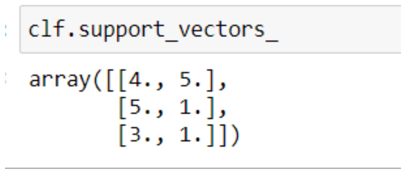
(ii)

If we add sample point x = [5 1] with label (-1) triangle, to the training set.

Solving in similar way as before:



The points that are falling on hyperplanes and are support vectors are, [3 1] , [4 5] and [5 1].



```
clf.support_vectors_
```

```
array([[4., 5.],
       [5., 1.],
       [3., 1.]])
```

(iii)

For a linear classifier:

$$w^T x + b \ < 0 \ \ => predict - 1$$

$$w^T x + b \ >= 0 \ \ => predict \ + 1$$

Assuming for a training data to be " linearly separable" → there is some weight w and b(bias) that separates positive training examples,

$$w^T x^{(i)} \ + b \ < \ 0 \ \ for \ \ y^{(i)} \ \ = -1$$

$$w^T x^{(i)} \ + b \ >= \ 0 \ \ for \ \ y^{(i)} \ \ = +1$$

If there are possible hyperplanes, we need to decided which one will lead to best generalization performance. Here we define Margin, which is the distance from hyperplane to closest training example. We have to choose hyperplane (w,b) to maximize the margin.

But for an optimized solution, we need to find w, b to minimize,

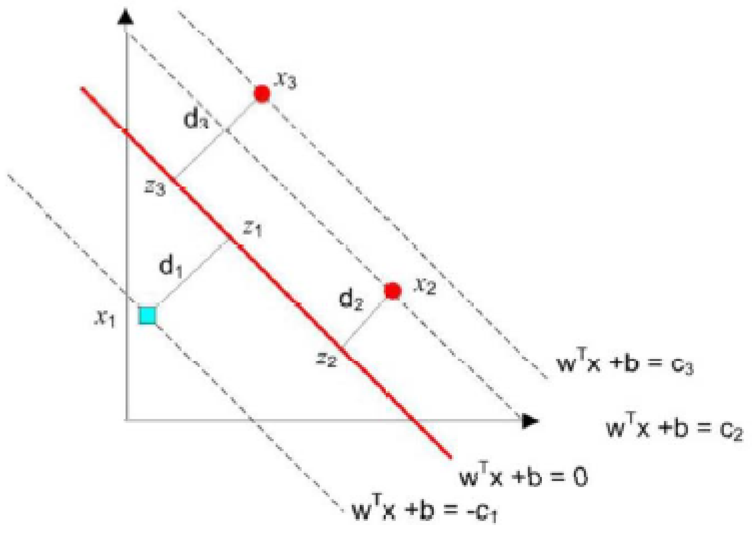
$$L(\vec{w}) = \frac{\| \vec{w} \|^2}{2}$$

Subject to ,

$$w^T x^{(i)} \;+ b \;<\; -1 \;\; for \;\; y^{(i)} \;\; = -1$$

$$w^T x^{(i)} \;+ b \;>=\; +1 \;\; for \;\; y^{(i)} \;\; = +1$$

(b) (8 marks) Consider the 2-dimensional data shown in the following figure. There are three data points, two of them are classified as positive (red circles) and one is negative (blue square). Let the decision boundary of the linear classifier be $w^T x + b = 0$ (shown as a red line in the diagram).



The geometric margin of each data point is the **perpendicular** distance between each data point to the decision boundary (shown as d_1, d_2, and d_3, respectively). Derive an expression for the total geometric margin, $M = d_1 + d_2 + d_3$, as a function of w, c_1, c_2, and c_3. (Hint: you can use the virtual points z_1, z_2 and z_3 as helpers to derive the expression; recall the process in deriving the margin in the lecture and leverage some of the formulas for the expression)

Answer:

Suppose we consider the hyperplane that separates the samples,

$$w^T \ x^i + b \ = 0$$

Let's say each sample is denoted as x and superscript (i) denotes ith sample. In the following section *y* superscripted with (i) represents label corresponding to the ith training example.

Functional margin **of a hyperplane** w.r.t. **i**th training example is defined as:

$$\gamma(\mathrm{i}) = y^i \ (w^T \ x^i \ + b \,)$$

Functional margin *of a hyperplane* w.r.t. the entire dataset is defined as:

$$\gamma \ = \ min_{i=1,2,3} \ \gamma^i$$

Geometric margin **of a hyperplane** w.r.t. **i**th training example is defined as functional margin normalized by norm(**w**):

$$\gamma^i \ = \ y^i \ \frac{w^T x^i \ + b}{||w||}$$

So for all 3 points x1,x2 and x3,

$$\gamma^1 \ = \ y^1 \ \frac{w^T x^1 \ +b}{||w||} \ = \ y^1 \ \frac{-c1}{||w||}$$

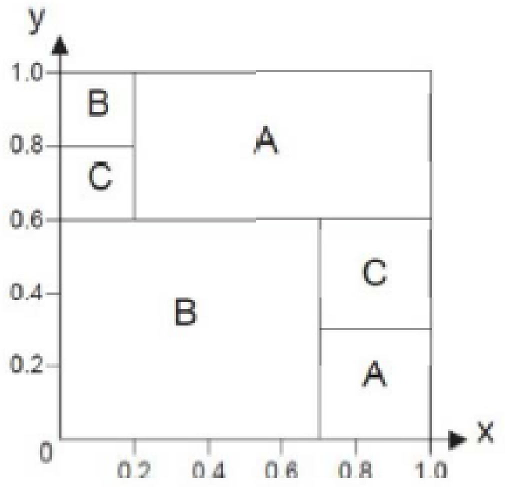
$$\gamma^2 \ = \ y^2 \ \frac{w^T x^2 \ +b}{||w||} \ = y^2 \ \frac{c2}{||w||}$$

$$\gamma^3 \ = \ y^3 \ \frac{w^T x^3 \ +b}{||w||} \ = \ y^3 \ \frac{c3}{||w||}$$

Geometric margin for a hyperplane with respect to the entire dataset is defined as:

$$\gamma \ = \ min_{i=1,2,3} \ \gamma^i$$

4. (Total 15 marks) **Decision Tree**
Consider a training set sampled uniformly from the the two-dimensional space shown in the following figure.



Assume that the training set size is large enough so that the probabilities can be calculated accurately based on the areas of the selected regions. The space is divided into three classes—A, B, and C. For example, we can say probability of class A is $p(A) = 0.3 \times 0.3 + 0.8 \times 0.4 = 0.41$. In this exercise, you will build a decision tree, using the concepts of **information gain** and **entropy**, from the training set. Note that it is slightly different from the examples taught in lecture, as the dataset only has two attributes, X and Y. You might need to repeatedly use some split points in X and Y to finally classify the labels of data points. Each node in the decision tree only checks one attribute (either X or Y).

(1) (4 marks) Given above, please calculate the overall entropy for the data (Info(T)).

(2) (9 marks) Compare the entropy when the data is split at $x <= 0.2$, $x <= 0.7$, and $y <= 0.6$.

(3) (2 marks) Based on your answer in part (b), which attribute split condition do you think should be used as the root of the decision tree.

Answer:

Instance	X	Y	Class
1	0.3	0.3	A
2	0.8	0.4	A
3	0.7	0.6	B
4	0.2	0.2	B
5	0.3	0.3	C
6	0.2	0.2	C

(1)

There are two A samples , two B samples, two C samples Thus,

P(A) = 0.3*0.3 + 0.8*0.4 = 0.41
P(B) = 0.7*0.6 + 0.2*0.2 = 0.46
P(C) = 0.3*0.3 + 0.2*0.2 = 0.13

Probability of A, B and C:

P(A) = 0.41
P(B) = 0.46
P(C) = 0.13

Proportion of examples on each class:

A = 0.41
B = 0.46
C= 0.13

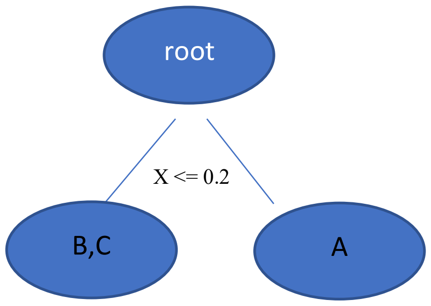
class0 = A
class1 = B
class2 = C

entropy = -(class0 * log2(class0) + class1 * log2(class1) + class2 * log2(class2))
= 1.425

(2)

Instance	X	Y	Class
1	0.3	0.3	A
2	0.8	0.4	A
3	0.7	0.6	B
4	0.2	0.2	B
5	0.3	0.3	C
6	0.2	0.2	C

First splitting at X <= 0.2:



Instance	X	Y	Class
1	0.3	0.3	A
2	0.8	0.4	A
3	0.7	0.6	B
4	0.2	0.2	B
5	0.3	0.3	C
6	0.2	0.2	C

two of instances go into one split and the remaining 4 instances go into the other split. We will calculate the entropy of each split and then make a weighted average over these two entropies.

entropy1 = -(1/2)*np.log2(1/2) - (1/2)*np.log2(1/2) = 1.0
entropy2 = -(2/4)*np.log2(2/4) - (1/4)*np.log2(1/4) - (1/4)*np.log2(1/4) = 1.5
entropy = (2/6)*entropy1 + (4/6)*entropy2 = 1.42

Secondly, splitting at X <= 0.7:

Instance	X	Y	Class
1	0.3	0.3	A
2	0.8	0.4	A
3	0.7	0.6	B
4	0.2	0.2	B
5	0.3	0.3	C
6	0.2	0.2	C

1 of instances go into one split and the remaining 5 instances go into the other split. We will calculate the entropy of each split and then make a weighted average over these two entropies.

entropy1 = -(1/1)*np.log2(1/1)
entropy2 = -(1/5)*np.log2(1/5) - (2/5)*np.log2(2/5) - (2/5)*np.log2(2/5)
entropy = (1/6)*entropy1 + (5/6)*entropy2 = 1.42

Secondly, splitting at X <= 0.7:

Instance	X	Y	Class
1	0.3	0.3	A
2	0.8	0.4	A
3	0.7	0.6	B
4	0.2	0.2	B
5	0.3	0.3	C
6	0.2	0.2	C

1 of instances go into one split and the remaining 5 instances go into the other split. We will calculate the entropy of each split and then make a weighted average over these two entropies.

entropy1 = -(1/1)*np.log2(1/1)
entropy2 = -(1/5)*np.log2(1/5) - (2/5)*np.log2(2/5) - (2/5)*np.log2(2/5)
entropy = (1/6)*entropy1 + (5/6)*entropy2 = 1.26

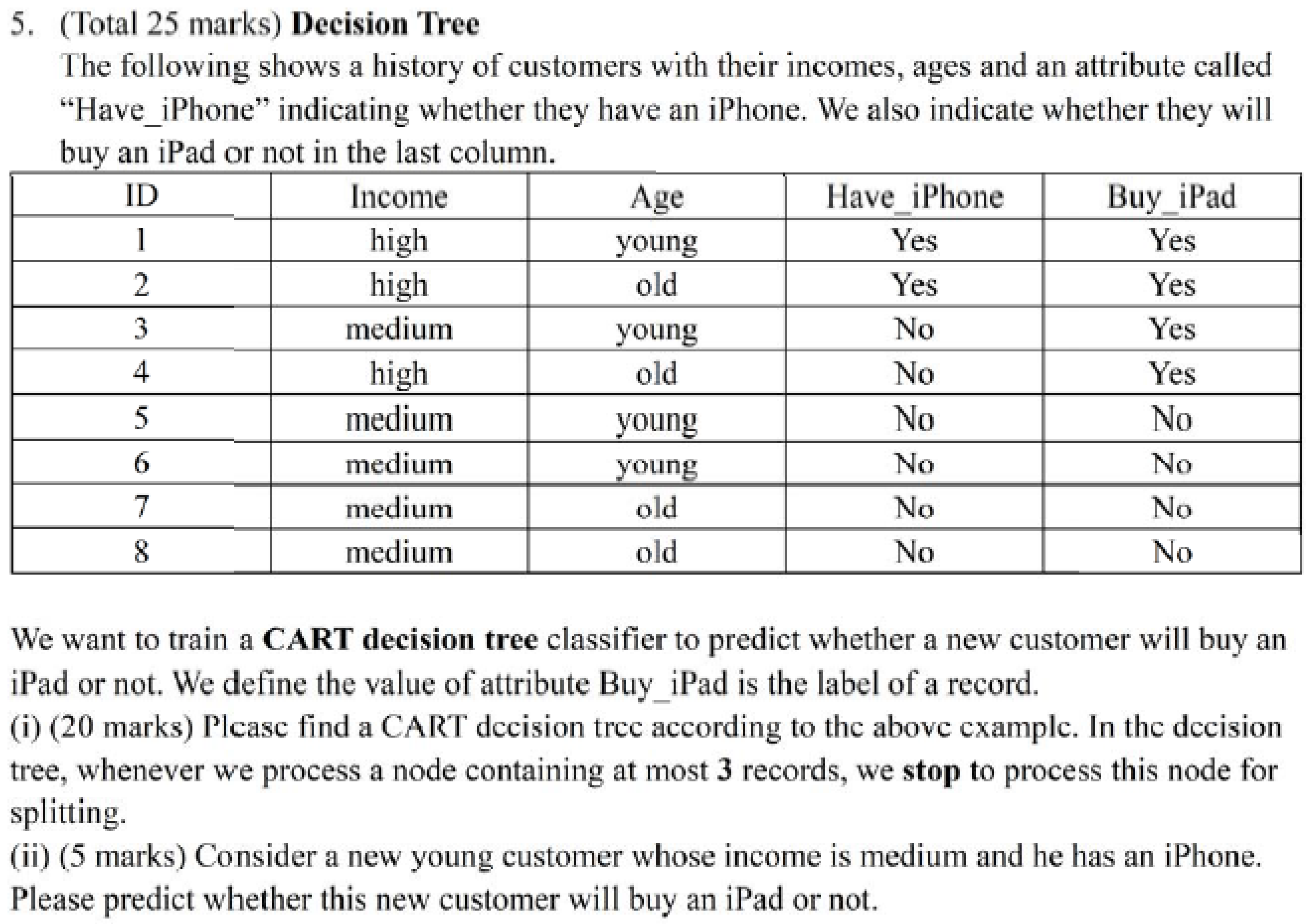
Thirdly, splitting at Y <= 0.6:

Instance	X	Y	Class
1	0.3	0.3	A
2	0.8	0.4	A
3	0.7	0.6	B
4	0.2	0.2	B
5	0.3	0.3	C
6	0.2	0.2	C

1 of instances go into one split and the remaining 5 instances go into the other split. We will calculate the entropy of each split and then make a weighted average over these two entropies.

entropy = -(2/6)*np.log2(2/6) - (2/6)*np.log2(2/6) - (2/6)*np.log2(2/6)
entropy = (1/6)*entropy1 + (5/6)*entropy2 = 1.58

(2) Based on above , we will consider split condition Y <= 0.6 , as root of the decision tree.



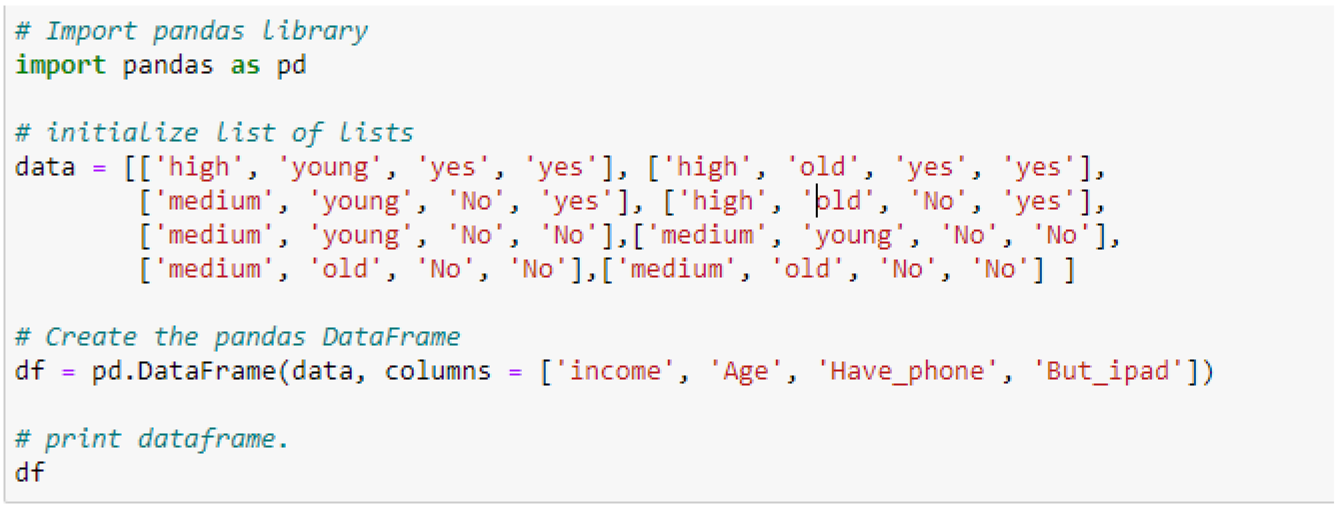
5. (Total 25 marks) **Decision Tree**
The following shows a history of customers with their incomes, ages and an attribute called "Have_iPhone" indicating whether they have an iPhone. We also indicate whether they will buy an iPad or not in the last column.

ID	Income	Age	Have_iPhone	Buy_iPad
1	high	young	Yes	Yes
2	high	old	Yes	Yes
3	medium	young	No	Yes
4	high	old	No	Yes
5	medium	young	No	No
6	medium	young	No	No
7	medium	old	No	No
8	medium	old	No	No

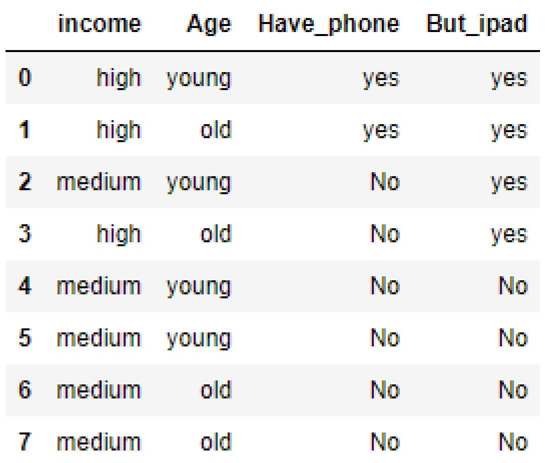
We want to train a **CART decision tree** classifier to predict whether a new customer will buy an iPad or not. We define the value of attribute Buy_iPad is the label of a record.
(i) (20 marks) Please find a CART decision tree according to the above example. In the decision tree, whenever we process a node containing at most **3** records, we **stop** to process this node for splitting.
(ii) (5 marks) Consider a new young customer whose income is medium and he has an iPhone. Please predict whether this new customer will buy an iPad or not.

Answer:

(i) Creating a dataframe of given dataset:



```python
# Import pandas library
import pandas as pd

# initialize list of lists
data = [['high', 'young', 'yes', 'yes'], ['high', 'old', 'yes', 'yes'],
        ['medium', 'young', 'No', 'yes'], ['high', 'old', 'No', 'yes'],
        ['medium', 'young', 'No', 'No'],['medium', 'young', 'No', 'No'],
        ['medium', 'old', 'No', 'No'],['medium', 'old', 'No', 'No'] ]

# Create the pandas DataFrame
df = pd.DataFrame(data, columns = ['income', 'Age', 'Have_phone', 'But_ipad'])

# print dataframe.
df
```



	income	Age	Have_phone	But_ipad
0	high	young	yes	yes
1	high	old	yes	yes
2	medium	young	No	yes
3	high	old	No	yes
4	medium	young	No	No
5	medium	young	No	No
6	medium	old	No	No
7	medium	old	No	No

Encoding column values in form of integers:

```
dataset_encoded=df.iloc[:,0:4]
le=LabelEncoder()

for i in dataset_encoded:
    dataset_encoded[i]=le.fit_transform(dataset_encoded[i])

print(dataset_encoded)
print()
print(df)
```

```
   income  Age  Have_phone  But_ipad
0       0    1           1         1
1       0    0           1         1
2       1    1           0         1
3       0    0           0         1
4       1    1           0         0
5       1    1           0         0
6       1    0           0         0
7       1    0           0         0

   income    Age Have_phone But_ipad
0    high  young        yes      yes
1    high    old        yes      yes
2  medium  young         No      yes
3    high    old         No      yes
4  medium  young         No       No
5  medium  young         No       No
6  medium    old         No       No
7  medium    old         No       No
```

Defining feature and target columns:-

Features: income, Age, Have_phone
Target: Buy_ipad

```
#Feature Set
X=dataset_encoded.iloc[:,0:3].values
X
#Label Set
y=dataset_encoded.iloc[:,3].values
y
```

```
array([1, 1, 1, 1, 0, 0, 0, 0])
```

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test
```

```
X_train.shape, X_test.shape
```

```
((5, 3), (3, 3))
```

Instantiating model with gini index:

```
# instantiate the DecisionTreeClassifier model with criterion gini index

clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=0)


# fit the model
clf_gini.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=4, random_state=0)
```

```
y_pred_gini = clf_gini.predict(X_test)
```

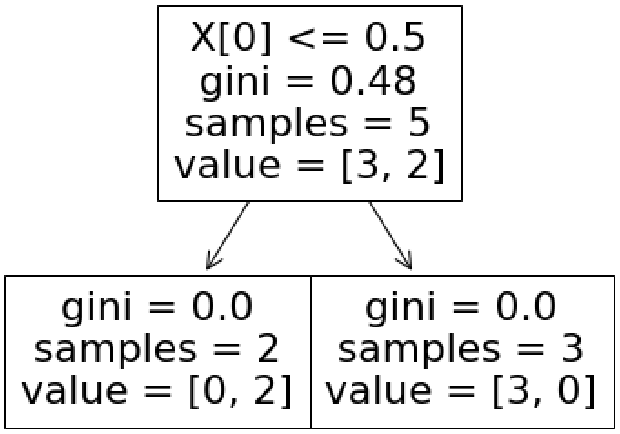
```
from sklearn.metrics import accuracy_score

print('Model accuracy score with criterion gini index: {0:0.4f}'. format(accuracy_score(y_test, y_pred_gini)))
```

```
Model accuracy score with criterion gini index: 0.6667
```

Plotting the decision tree:

```
plt.figure(figsize=(6,5))

from sklearn import tree

tree.plot_tree(clf_gini.fit(X_train, y_train))
```

```
[Text(167.4, 203.85000000000002, 'X[0] <= 0.5\ngini = 0.48\nsamples = 5\nvalue = [3, 2]'),
 Text(83.7, 67.94999999999999, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(251.10000000000002, 67.94999999999999, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]')]
```

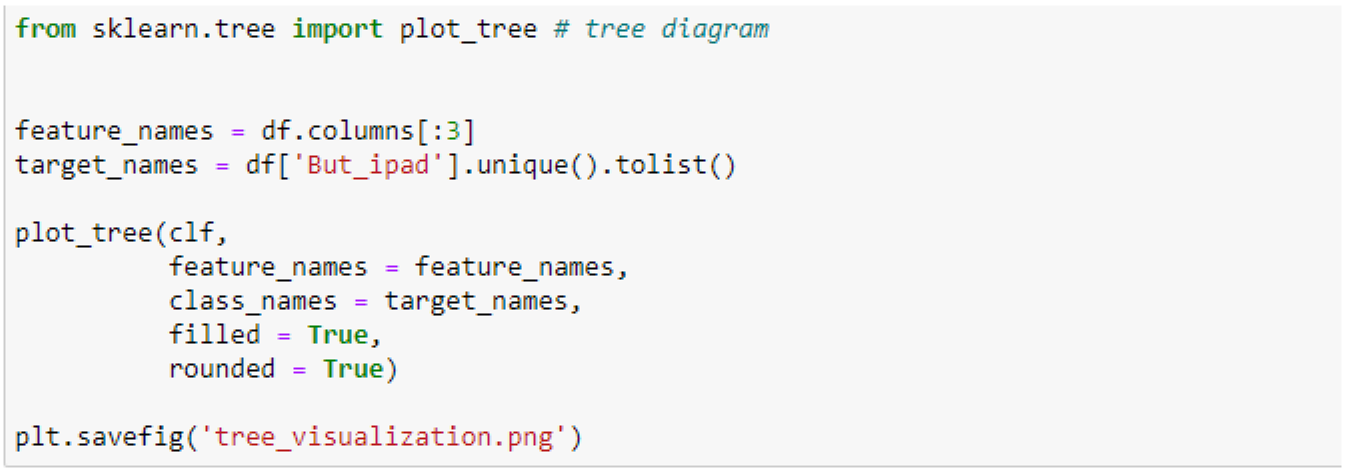


Test representation of tree:

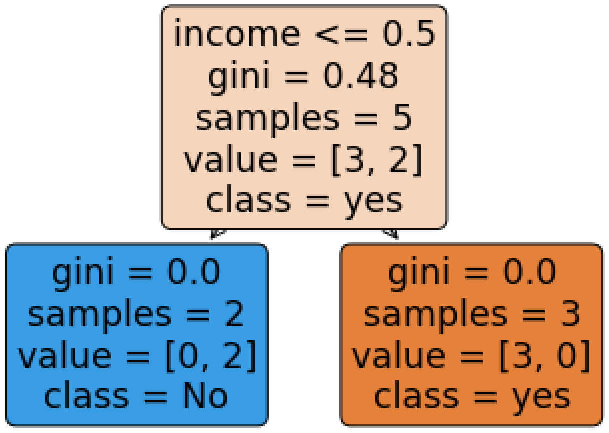
```
text_representation = tree.export_text(clf)
print(text_representation)
```

```
|--- feature_0 <= 0.50
|   |--- class: 1
|--- feature_0 >  0.50
|   |--- class: 0
```

Detailed tree diagram:



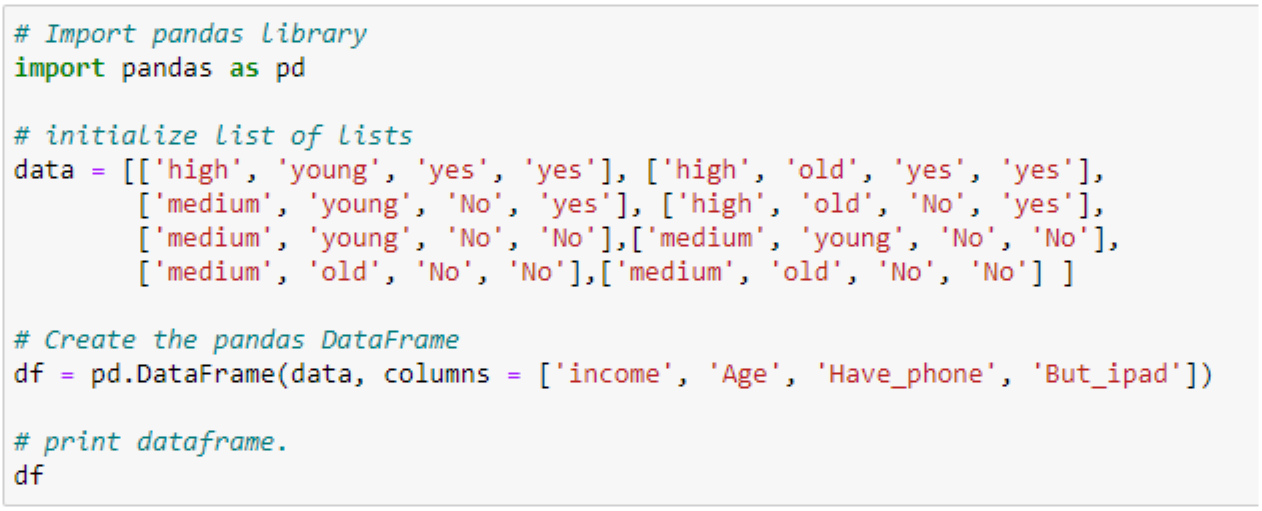
```
from sklearn.tree import plot_tree # tree diagram


feature_names = df.columns[:3]
target_names = df['But_ipad'].unique().tolist()

plot_tree(clf,
          feature_names = feature_names,
          class_names = target_names,
          filled = True,
          rounded = True)

plt.savefig('tree_visualization.png')
```



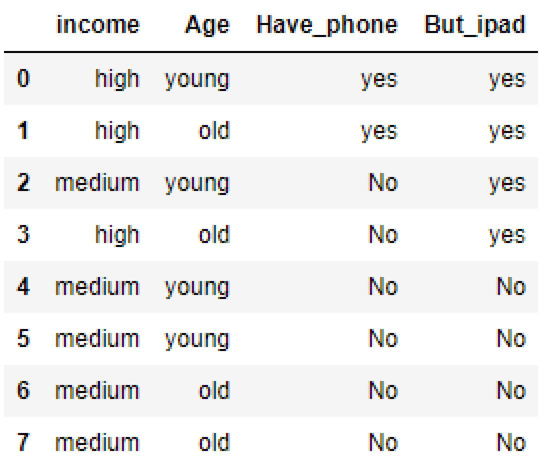
(ii)

To predict if a new young customer whose income is medium and he has an iphone, to buy an ipad or not?

Showing step by step implementation:



```python
# Import pandas library
import pandas as pd

# initialize list of lists
data = [['high', 'young', 'yes', 'yes'], ['high', 'old', 'yes', 'yes'],
        ['medium', 'young', 'No', 'yes'], ['high', 'old', 'No', 'yes'],
        ['medium', 'young', 'No', 'No'],['medium', 'young', 'No', 'No'],
        ['medium', 'old', 'No', 'No'],['medium', 'old', 'No', 'No'] ]

# Create the pandas DataFrame
df = pd.DataFrame(data, columns = ['income', 'Age', 'Have_phone', 'But_ipad'])

# print dataframe.
df
```



	income	Age	Have_phone	But_ipad
0	high	young	yes	yes
1	high	old	yes	yes
2	medium	young	No	yes
3	high	old	No	yes
4	medium	young	No	No
5	medium	young	No	No
6	medium	old	No	No
7	medium	old	No	No

Trying encoding of values of features for prediction:

```
dataset_encoded=df.iloc[:,0:4]
le=LabelEncoder()

for i in dataset_encoded:
    dataset_encoded[i]=le.fit_transform(dataset_encoded[i])

print(dataset_encoded)
print()
print(df)
```

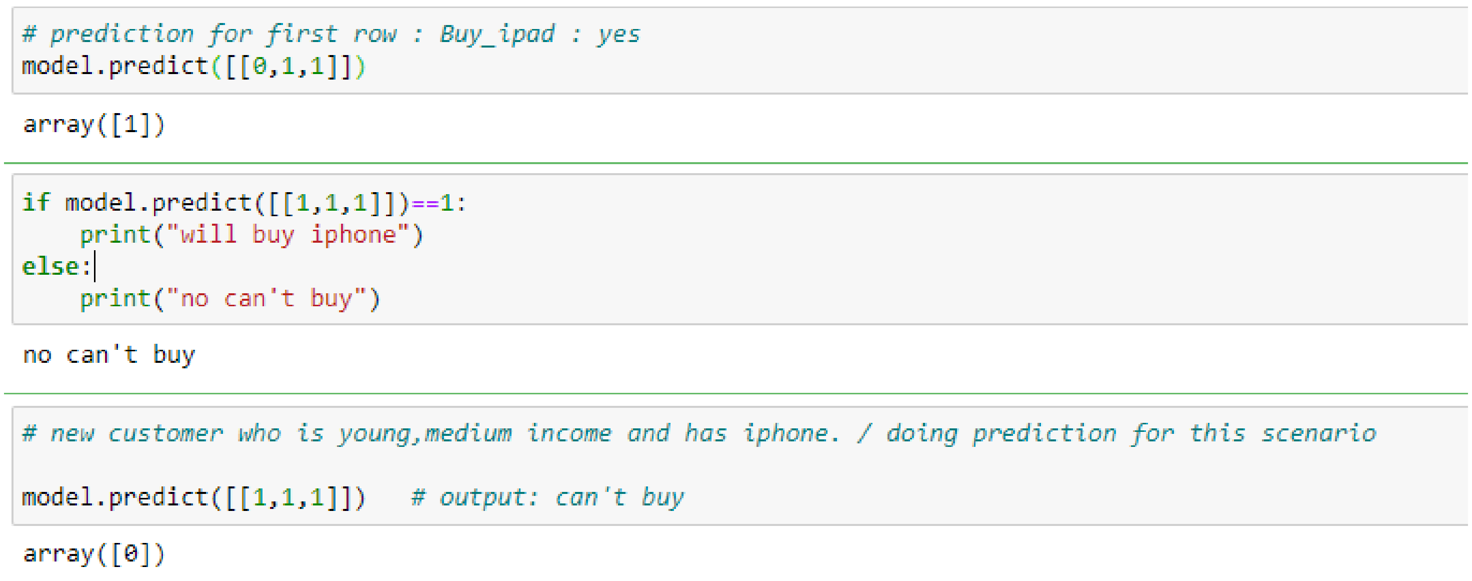
```
   income  Age  Have_phone  But_ipad
0       0    1           1         1
1       0    0           1         1
2       1    1           0         1
3       0    0           0         1
4       1    1           0         0
5       1    1           0         0
6       1    0           0         0
7       1    0           0         0

   income    Age Have_phone But_ipad
0    high  young        yes      yes
1    high    old        yes      yes
2  medium  young         No      yes
3    high    old         No      yes
4  medium  young         No       No
5  medium  young         No       No
6  medium    old         No       No
7  medium    old         No       No
```

```
#Feature Set
X=dataset_encoded.iloc[:,0:3].values
X
#Label Set
y=dataset_encoded.iloc[:,3].values
y
```

```
array([1, 1, 1, 1, 0, 0, 0, 0])
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=2)

model=DecisionTreeClassifier(criterion='gini')

model.fit(X_train,y_train)
```

```
DecisionTreeClassifier()
```



```
# prediction for first row : Buy_ipad : yes
model.predict([[0,1,1]])
```

```
array([1])
```

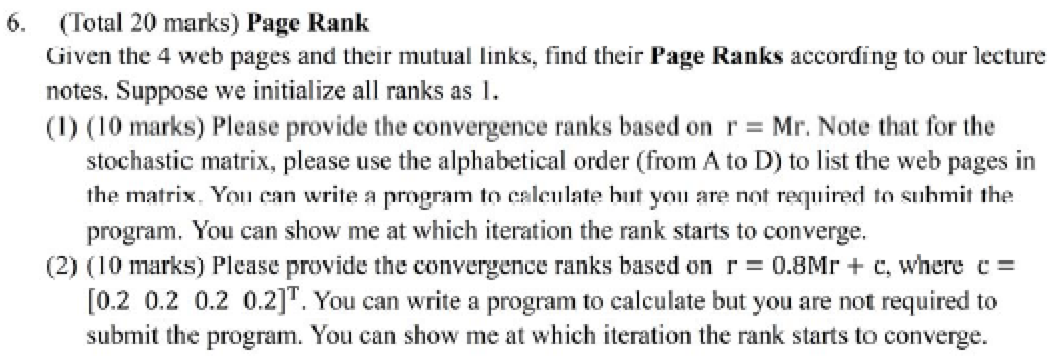
```
if model.predict([[1,1,1]])==1:
    print("will buy iphone")
else:
    print("no can't buy")
```

```
no can't buy
```

```
# new customer who is young,medium income and has iphone. / doing prediction for this scenario

model.predict([[1,1,1]])   # output: can't buy
```

```
array([0])
```

After prediction it’s predicted that a new young customer with medium income having iPhone will not buy an iPad.

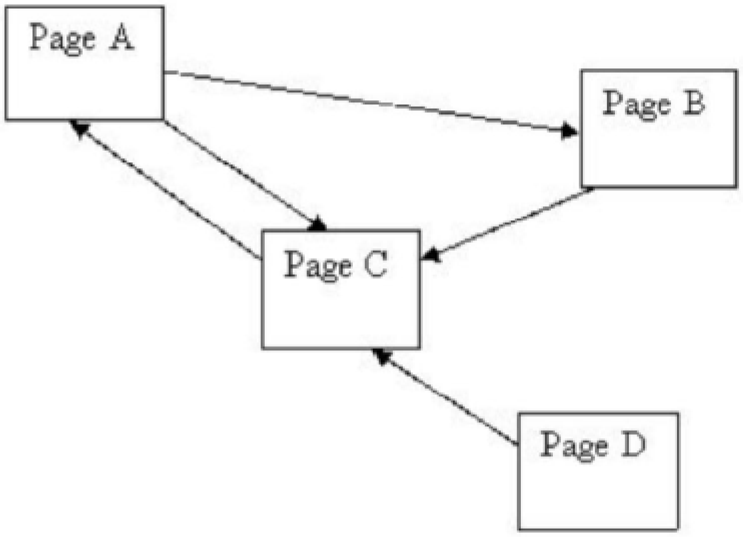


6. (Total 20 marks) **Page Rank**

Given the 4 web pages and their mutual links, find their **Page Ranks** according to our lecture notes. Suppose we initialize all ranks as 1.

(1) (10 marks) Please provide the convergence ranks based on $r = Mr$. Note that for the stochastic matrix, please use the alphabetical order (from A to D) to list the web pages in the matrix. You can write a program to calculate but you are not required to submit the program. You can show me at which iteration the rank starts to converge.

(2) (10 marks) Please provide the convergence ranks based on $r = 0.8Mr + c$, where $c = [0.2\ \ 0.2\ \ 0.2\ \ 0.2]^T$. You can write a program to calculate but you are not required to submit the program. You can show me at which iteration the rank starts to converge.



Answer:

(1)

From my code, I could see total 8 iterations with values changing by smaller amount each time.

All iterations:

```
[[0.303125]
 [0.196875]
 [0.409375]
 [0.090625]]


[[0.40472656]
 [0.18558594]
 [0.35292969]
 [0.05675781]]


[[0.34955127]
 [0.22156982]
 [0.37931787]
 [0.04956104]]


[[0.37045191]
 [0.19659101]
 [0.38492536]
 [0.04803172]]


[[0.3748933 ]
 [0.2051488 ]
 [0.37225116]
 [0.04770674]]


[[0.36405117]
 [0.20696733]
 [0.38134382]
 [0.04763768]]


[[0.37176525]
 [0.20234475]
 [0.37826699]
 [0.04762301]]


[[0.36914683]
 [0.20562012]
 [0.37761316]
 [0.04761989]]
```

The last vector output in this iterative process is the final PageRank.

Given this vector, we can see that Page C has the highest rank. This is expected because Page C has the most inbound links of any page. But notice that Page A has a very high rank, too, despite having only one inbound link! This is because this single inbound link comes from Page C, which happens to be the most important page.

(2)

From my code, I could see total 16 iterations with values changing by smaller amount each time.

All iterations:

```
[[0.26]
 [0.16]
 [0.36]
 [0.06]]


[[0.3084]
 [0.1244]
 [0.2524]
 [0.0204]]


[[0.213056]
 [0.134496]
 [0.234016]
 [0.011136]]


[[0.19536704]
 [0.09337664]
 [0.20097344]
 [0.00815424]]


[[0.16738831]
 [0.08475638]
 [0.15945769]
 [0.00660956]]


[[0.13307018]
 [0.07245936]
 [0.14026446]
 [0.00550403]]


[[0.11682535]
 [0.05784186]
 [0.11580935]
 [0.00461379]]


[[0.09652114]
 [0.0506038 ]
 [0.09687729]
 [0.00387366]]
```

```
[[0.08075532]
 [0.04186195]
 [0.08234499]
 [0.00325349]]


[[0.06860885]
 [0.03503499]
 [0.06852454]
 [0.00273286]]


[[0.05711522]
 [0.02973912]
 [0.05776711]
 [0.00229558]]


[[0.04814197]
 [0.02477437]
 [0.04856567]
 [0.00192829]]


[[0.0404723 ]
 [0.02087655]
 [0.04069605]
 [0.00161976]]


[[0.03391744]
 [0.01754952]
 [0.03425076]
 [0.0013606 ]]


[[0.02854351]
 [0.01470988]
 [0.02874949]
 [0.0011429 ]]


[[0.02395963]
 [0.01237744]
 [0.02414534]
 [0.00096004]]

iteration: 16
```

The last vector output in this iterative process is the final PageRank.

Given this vector, we can see that Page C has the highest rank. This is expected because Page C has the most inbound links of any page. But notice that Page A has a very high rank, too, despite having only one inbound link! This is because this single inbound link comes from Page C, which happens to be the most important page.