CS M146 Fall 2023 Homework 2 Solutions

UCLA ID: 605900342
Name: Caleb Traxler
Collaborators: N/A

By turning in this assignment, I agree by the UCLA honor code and declare that all of this is my own work.

Problem 1 (Decision Trees)

.

You are checking the weather and would like to know whether a day is comfortable or not (IsComfortable). This is determined by factors such as whether a day is sunny (IsSunny), whether a day is hot (IsHot), etc. You have the following data to consider.

Example	IsSunny	IsHot	IsHumid	IsWindy	IsComfortable
A	0	0	0	0	0
B	0	0	1	0	0
C	1	1	0	1	0
D	1	0	0	1	1
E	0	1	1	0	1
F	0	0	1	1	1
G	0	0	0	1	1
H	1	0	1	1	1
X	1	1	1	1	?
Y	0	1	0	1	?
Z	1	1	0	0	?

You know whether or not a day A through H is comfortable, but you do not know about X through Z. For questions (a) - (d), consider only days A through H. For all calculations in this problem, use base 2 for the logarithms.

1. What is the entropy of IsComfortable?

   Entropy tells us how impure a collection of data is, i.e. Entropy is the measurement of homogeneity. It returns us the information about an arbitrary data set that how impure/non-homogeneous the data set it.

   Entropy(IsComfortable) $= -(P_{(1)} log_2 P_{(1)} + P_{(0)} log_2 P_{(0)})$,

   such that $P_{(1)}$ is the portion of (1) examples and $P_{(0)}$ is the portion of (0) examples.

   Entropy(IsComfortable) $= -(\frac{5}{8} log_2 \frac{5}{8} + \frac{3}{8} log_2 \frac{3}{8}) = 0.954434$

   Which concludes, the data-set is 95.4434% impure or 95.4434% non-homogeneous.

2. Which attribute should you choose as the root of a decision tree?

I need to evaluate each attribute using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected as the root of the tree. Then I need to find which node will be next after the chosen root. Thus, I will calculate the information gain of each attribute to determine which will be the root of the decision tree.

First I will calculate the entropy's of all the attributes:

Entropy(IsComfortable) $= -(\frac{5}{8}log_2\frac{5}{8} + \frac{3}{8}log_2\frac{3}{8}) = 0.954434$

Entropy(IsSunny) $= -(\frac{3}{8}log_2\frac{3}{8} + \frac{5}{8}log_2\frac{5}{8}) = 0.954434$

Entropy(IsHot) $= -(\frac{2}{8}log_2\frac{2}{8} + \frac{6}{8}log_2\frac{6}{8}) = 0.811278$

Entropy(IsHumid) $= -(\frac{4}{8}log_2\frac{4}{8} + \frac{4}{8}log_2\frac{4}{8}) = 1$

Entropy(IsWindy) $= -(\frac{5}{8}log_2\frac{5}{8} + \frac{3}{8}log_2\frac{3}{8}) = 0.954434$

Next, recall that given the Entropy is the measure of impurity in a collection of data-set, now I can measure the effectiveness of an attribute in classifying the training set. The measure is called information gain, which is the expected reduction in entropy cause by partitioning the data set according to this attribute.

$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$,

where $Values(A)$ is the all possible values for attribute A, and S_v is the subset of S for which attribute A has value v.

- IsSunny Information Gain Calculation:

$values(IsSunny) = 1, 0$

$S(Outcome) = [5+, 3-]$ from IsComfortable

$S(1) = [2+, 1-]$

$S(0) = [3+, 2-]$

$Entropy(S(Outcome)) = -(\frac{5}{8}log_2(\frac{5}{8}) + \frac{3}{8}log_2(\frac{3}{8})) = 0.9544$

$Entropy(S(1)) = -(\frac{2}{3}log_2(\frac{2}{3}) + \frac{1}{3}log_2(\frac{1}{3})) = 0.9183$

$Entropy(S(0)) = -(\frac{3}{5}log_2(\frac{3}{5}) + \frac{2}{5}log_2(\frac{2}{5})) = 0.97095$

$Gain(S, A) = Entropy(S) - ((\frac{3}{8})(0.9183) + (\frac{5}{8})(0.970951))$

$= 0.9544 - ((\frac{3}{8})(0.9183) + (\frac{5}{8})(0.970951)) \approx 0.00319323$

- IsHot Information Gain Calculation:

$values(IsHot) = 1, 0$

$S(Outcome) = [5+, 3-]$ from IsComfortable

$S(1) = [1+, 1-]$

$S(0) = [4+, 2-]$

$Entropy(S(Outcome)) = -(\frac{5}{8}log_2(\frac{5}{8}) + \frac{3}{8}log_2(\frac{3}{8})) = 0.9544$

$Entropy(S(1)) = -(\frac{1}{2}log_2(\frac{1}{2}) + \frac{1}{2}log_2(\frac{1}{2})) = 1$

$Entropy(S(0)) = -(\frac{4}{6}log_2(\frac{4}{6}) + \frac{2}{6}log_2(\frac{2}{6})) = 0.9183$

$Gain(S, A) = Entropy(S) - ((\frac{2}{8})(1) + (\frac{6}{8})(0.9183))$

$= 0.9544 - ((\frac{2}{8})(1) + (\frac{6}{8})(0.9183)) \approx 0.01568$

- IsHumind Information Gain Calculation:

$values(IsHumid) = 1, 0$

$S(Outcome) = [5+, 3-]$ from IsComfortable

$S(1) = [3+, 1-]$

$S(0) = [2+, 2-]$

$Entropy(S(Outcome)) = -(\frac{5}{8}log_2(\frac{5}{8}) + \frac{3}{8}log_2(\frac{3}{8})) = 0.9544$

$Entropy(S(1)) = -(\frac{3}{4}log_2(\frac{3}{4}) + \frac{1}{4}log_2(\frac{1}{4})) = 0.81128$

$Entropy(S(0)) = -(\frac{2}{4}log_2(\frac{2}{4}) + \frac{2}{4}log_2(\frac{2}{4})) = 1$

$Gain(S, A) = Entropy(S) - ((\frac{4}{8}(0.81128) + (\frac{4}{8})(1))$

$= 0.9544 - (\frac{4}{8}(0.81128) + (\frac{4}{8})(1)) \approx 0.04876$

- IsWindy Information Gain Calculation:

$values(IsWindy) = 1, 0$

$S(Outcome) = [5+, 3-]$ from IsComfortable

$S(1) = [4+, 1-]$

$S(0) = [1+, 2-]$

$Entropy(S(Outcome)) = -(\frac{5}{8}log_2(\frac{5}{8}) + \frac{3}{8}log_2(\frac{3}{8})) = 0.9544$

$Entropy(S(1)) = -(\frac{4}{5}log_2(\frac{4}{5}) + \frac{1}{5}log_2(\frac{1}{5})) = 0.72193$

$Entropy(S(0)) = -(\frac{1}{3}log_2(\frac{1}{3}) + \frac{2}{3}log_2(\frac{2}{3})) = 0.9183$

$Gain(S, A) = Entropy(S) - ((\frac{5}{8})(0.72193) + (\frac{3}{8})(0.9183))$

$= 0.9544 - ((\frac{5}{8})(0.72193) + (\frac{3}{8})(0.9183)) \approx 0.1588$

Information Gain for IsSunny: 0.00319323

Information Gain for IsHot: 0.01568

Information Gain for IsHumid: 0.04876

Information Gain for IsWindy: 0.1588

Overall, I see that the attribute that has the largest Information Gain quantity is the IsWindy attribute. Thus I should choose this attribute as the root of my decision tree. Next I should choose the IsHumid attribute as the next in the decision tree, then the IsHot attribute and lastly the IsSunny attribute as the last root int he decision tree based on their information gain values.

3. What is the information gain of the attribute you chose in the previous question?

   The information gain of the attribute that I chose in the previous question is, IsWindy with an information gain of 0.1588.

   - IsWindy Information Gain Calculation:

   $values(IsWindy) = 1, 0$

   $S(Outcome) = [5+, 3-]$ from IsComfortable

   $S(1) = [4+, 1-]$

   $S(0) = [1+, 2-]$

   $Entropy(S(Outcome)) = -(\frac{5}{8}log_2(\frac{5}{8}) + \frac{3}{8}log_2(\frac{3}{8})) = 0.9544$

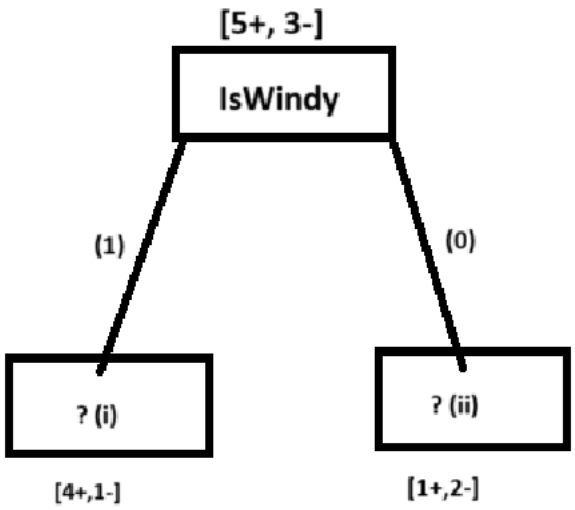
   $Entropy(S(1)) = -(\frac{4}{5}log_2(\frac{4}{5}) + \frac{1}{5}log_2(\frac{1}{5})) = 0.72193$

   $Entropy(S(0)) = -(\frac{1}{3}log_2(\frac{1}{3}) + \frac{2}{3}log_2(\frac{2}{3})) = 0.9183$

$Gain(S, A) = Entropy(S) - ((\frac{5}{8})(0.72193) + (\frac{3}{8})(0.9183))$
$= 0.9544 - ((\frac{5}{8})(0.72193) + (\frac{3}{8})(0.9183)) \approx 0.1588$

4. Using ID3 algorithm, build a decision tree to classify days as comfortable or not.

   The ID3 algorithm will use the attributes as its root to build the decision tree. Then it will calculate the information gain to find the next node. Here I will use IsWindy as the root node and from there I will extend the two branches corresponding to (1) and (0). From there I will be able to create two new tables, which I can further use to make some predictions after calculating their entropy's and information gains.



(i)

Example	IsSunny	IsHot	IsHumid	IsComfortable
C	1	1	0	0
D	1	0	0	1
F	0	0	1	1
G	0	0	0	1
H	1	0	1	1

(ii)

Example	IsSunny	IsHot	IsHumid	IsComfortable
A	0	0	0	0
B	0	0	1	0
E	0	1	1	1

(i) Now I will compute the entropy's of each attribute and further compute the information gains.

- IsSunny (i) Information Gain Calculation:

$values(IsSunny) = 1, 0$

$S(Outcome) = [4+, 1-]$ from IsComfortable

$S(1) = [2+, 1-]$

$S(0) = [2+, 0-]$

$Entropy(S(Outcome)) = -(\frac{4}{5}log_2(\frac{4}{5}) + \frac{1}{5}log_2(\frac{1}{5})) = 0.7219281$

$Entropy(S(1)) = -(\frac{2}{3}log_2(\frac{2}{3}) + \frac{1}{3}log_2(\frac{1}{3})) = 0.918296$

$Entropy(S(0)) = -(\frac{2}{2}log_2(\frac{2}{2}) + \frac{0}{2}log_2(\frac{0}{2})) = 0$

$Gain(S, A) = Entropy(S) - ((\frac{3}{5})(0.918296) + (\frac{2}{5})(0))$
$= 0.7219281 - ((\frac{3}{5})(0.918296) + (\frac{2}{5})(0)) \approx 0.1709505$

- IsHot (i) Information Gain Calculation:
$values(IsHot) = 1, 0$
$S(Outcome) = [4+, 1-]$ from IsComfortable
$S(1) = [0+, 1-]$
$S(0) = [4+, 0-]$
$Entropy(S(Outcome)) = -(\frac{4}{5}log_2(\frac{4}{5}) + \frac{1}{5}log_2(\frac{1}{5})) = 0.7219281$
$Entropy(S(1)) = -(\frac{0}{1}log_2(\frac{0}{1}) + \frac{1}{1}log_2(\frac{1}{1})) = 0$
$Entropy(S(0)) = -(\frac{4}{4}log_2(\frac{4}{4}) + \frac{0}{4}log_2(\frac{0}{4})) = 0$
$Gain(S, A) = Entropy(S) - ((\frac{1}{5})(0) + (\frac{4}{5})(0))$
$= 0.7219281 - 0 \approx 0.7219281$

- IsHumid (i) Information Gain Calculation:
$values(IsHumid) = 1, 0$
$S(Outcome) = [4+, 1-]$ from IsComfortable
$S(1) = [2+, 0-]$
$S(0) = [2+, 1-]$
$Entropy(S(Outcome)) = -(\frac{4}{5}log_2(\frac{4}{5}) + \frac{1}{5}log_2(\frac{1}{5})) = 0.7219281$
$Entropy(S(1)) = -(\frac{2}{2}log_2(\frac{2}{2}) + \frac{0}{2}log_2(\frac{0}{2})) = 0$
$Entropy(S(0)) = -(\frac{2}{3}log_2(\frac{2}{3}) + \frac{1}{3}log_2(\frac{1}{3})) = 0.918296$
$Gain(S, A) = Entropy(S) - ((\frac{2}{5})(0) + (\frac{3}{5})(0.918296))$
$= 0.7219281 - ((\frac{2}{5})(0) + (\frac{3}{5})(0.918296)) \approx 0.1709505$

Information Gain for IsSunny: 0.1709505

Information Gain for IsHot: 0.7219281

Information Gain for IsHumid: 0.1709505

Overall, I see that the attribute that has the largest Information Gain quantity is the IsHot attribute. Thus I should choose this attribute as the next root of my decision tree.

(ii) Now I will compute the entropy's of each attribute and further compute the information gains.

- IsSunny (ii) Information Gain Calculation:
$values(IsSunny) = 1, 0$
$S(Outcome) = [1+, 2-]$ from IsComfortable
$S(1) = [0+, 0-]$
$S(0) = [1+, 2-]$
$Entropy(S(Outcome)) = -(\frac{1}{3}log_2(\frac{1}{3}) + \frac{2}{3}log_2(\frac{2}{3})) = 0.9182958$
$Entropy(S(1)) = 0$

$Entropy(S(0)) = -(\frac{1}{3}log_2(\frac{1}{3}) + \frac{2}{3}log_2(\frac{2}{3})) = 0.9182958$

$Gain(S, A) = 0.9182958 - 0.9182958 = 0$

- IsHot (ii) Information Gain Calculation:

$values(IsHot) = 1, 0$

$S(Outcome) = [1+, 2-]$ from IsComfortable

$S(1) = [1+, 0-]$

$S(0) = [0+, 2-]$

$Entropy(S(Outcome)) = -(\frac{1}{3}log_2(\frac{1}{3}) + \frac{2}{3}log_2(\frac{2}{3})) = 0.9182958$

$Entropy(S(1)) = -(\frac{1}{1}log_2(\frac{1}{1}) + \frac{0}{1}log_2(\frac{0}{1})) = 0$

$Entropy(S(0)) = -(\frac{0}{2}log_2(\frac{0}{2}) + \frac{2}{2}log_2(\frac{2}{2})) = 0$

$Gain(S, A) = Entropy(S) - ((\frac{1}{3})(0) + (\frac{2}{3})(0))$

$= 0.9182958 - 0 \approx 0.9182958$

- IsHumid (ii) Information Gain Calculation:

$values(IsHumid) = 1, 0$

$S(Outcome) = [1+, 2-]$ from IsComfortable

$S(1) = [1+, 1-]$

$S(0) = [0+, 1-]$

$Entropy(S(Outcome)) = -(\frac{1}{3}log_2(\frac{1}{3}) + \frac{2}{3}log_2(\frac{2}{3})) = 0.9182958$

$Entropy(S(1)) = -(\frac{1}{2}log_2(\frac{1}{2}) + \frac{1}{2}log_2(\frac{1}{2})) = 1$

$Entropy(S(0)) = -(\frac{0}{1}log_2(\frac{0}{1}) + \frac{1}{1}log_2(\frac{1}{1})) = 0$

$Gain(S, A) = Entropy(S) - ((\frac{2}{3})(1) + (\frac{1}{3})(0))$

$= 0.9182958 - (\frac{2}{3}) \approx 0.25162913$

Information Gain for IsSunny: 0

Information Gain for IsHot: 0.9182958

Information Gain for IsHumid: 0.25162913

Overall, I see that the attribute that has the largest Information Gain quantity is the IsHot attribute. Thus I should choose this attribute as the next root of my decision tree.

Based on the data and the calculations:

Not hot, not windy, not comfortable.

Not hot, not windy, not comfortable.

Hot, windy, not comfortable.

Not hot, windy, comfortable.

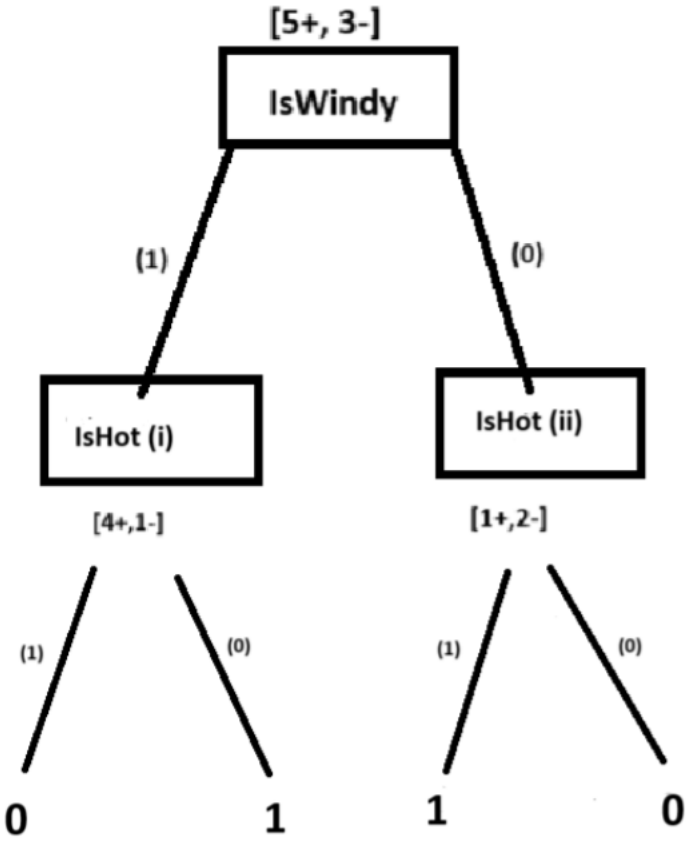
Hot, not windy, comfortable.

Not hot, windy, comfortable.

Not hot, windy, comfortable.

Not hot, windy, comfortable.

Thus, the final tree looks like the following diagram:



5. Classify days X, Y and Z using this decision tree as comfortable or not.

   Using the constructed ID3 algorithm, I can now make predictions to complete the data table. Thus, I will predict X, Y and Z.

   X : Hot, Windy $\implies$ Not Comfortable.

   Y: Hot, Windy $\implies$ Not Comfortable.

   Z: Hot, Not Windy $\implies$ Comfortable.

Example	IsSunny	IsHot	IsHumid	IsWindy	IsComfortable
A	0	0	0	0	0
B	0	0	1	0	0
C	1	1	0	1	0
D	1	0	0	1	1
E	0	1	1	0	1
F	0	0	1	1	1
G	0	0	0	1	1
H	1	0	1	1	1
X	1	1	1	1	0
Y	0	1	0	1	0
Z	1	1	0	0	1

Problem 2 (Entropy and Information)

.

The entropy of a Bernoulli (Boolean 0/1) random variable X with $p(X = 1) = q$ is given by

$$B(q) = -q \log q - (1 - q) \log(1 - q).$$

Suppose that a set S contains p positive examples and n negative examples. The entropy of S is defined as $H(S) = B\left(\frac{p}{p+n}\right)$.

Based on an attribute X_j, we split the set S into k disjoint subsets S_k, with p_k positive and n_k negative examples in each. If the ratio $\frac{p_k}{p_k+n_k}$ is the same for all k, show that the information gain of this attribute is 0.

I am given that $H(S) = B(\frac{p}{p+n})$, where $q = \frac{p}{p+n}$. This is the entropy of the entire set S. I am also given that $\frac{p_k}{p_k+n_k}$ is the same for all k, which meant hat the entropy of each subset S_k is the same and equal to $H(S_k) = B(\frac{p_k}{p_k+n_k}) = B(q)$. Which implies that every subset S_k has the same entropy as the original set S.

Recall the definition of the Information Gain: Recall that entropy is the measure of impurity in a collection of data-set, now I can measure the effectiveness of an attribute in classifying the training set. The measure is called information gain, which is the expected reduction in entropy cause by partitioning the data set according to this attribute. Generally, $Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$, where $Values(A)$ is the all possible values for attribute A, and S_v is the subset of S for which attribute A has value v.

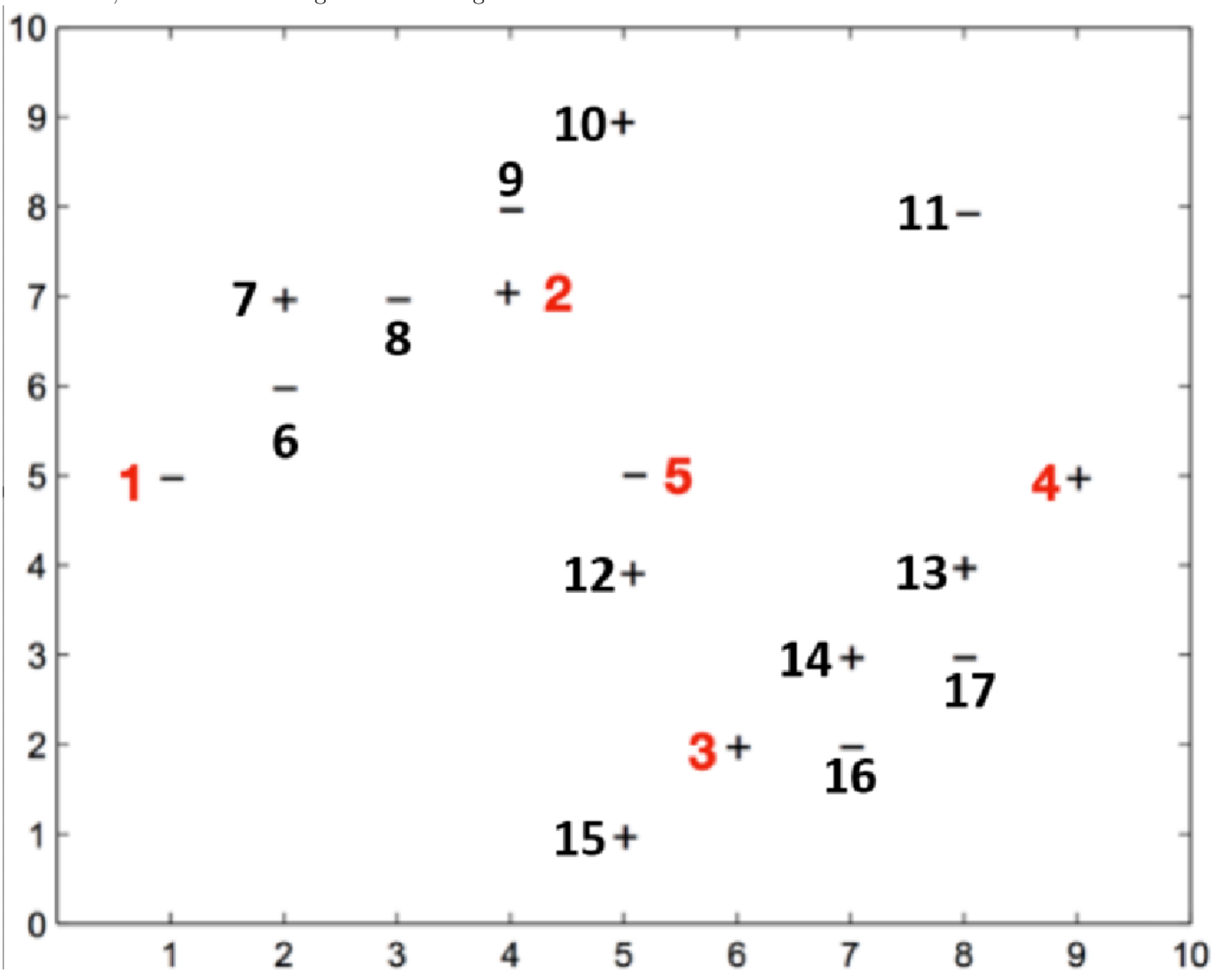
In this case I can write $Gain = H(S) - \sum_k (\frac{|S_k|}{|S|} H(S_k)) = H(S) - \sum_k (\frac{|S_k|}{|S|} B(q))$

Given that the sum of the sizes of all the subsets S_k is equal to the size of S (Since S was split into these subsets), I can simplify the summation in the following way: $\sum_k |S_k| = |S|$. I also know that $\sum_k (\frac{|S_k|}{|S|}) = 1$ Thus, the equation becomes:

$Gain = H(S) - B(q) \sum_k (\frac{|S_k|}{|S|} = H(S) - B(q)$. However, because I know that $H(S) = B(q)$, then $Gain = 0$. Therefore the information gain of of this attribute is 0.

Problem 3 (k-Nearest Neighbors)

In the following questions, you will consider a k-nearest neighbor classifier using Euclidean distance metric on a binary classification task. We assign the class of the test point to be the class of the majority of the k nearest neighbors. Below is the figure for dataset. Note that the annotated data points (from 1 to 5) belong to the test set, and the rest belong to the training set.



Recall the definition of the Euclidean distance metric: According to the Euclidean distance formula, the distance between two points in the plane with coordinates (x, y) and (a, b), is given by:

$dist(d) = \sqrt{(x-a)^2 + (y-b)^2}$

Test Set $= \{1, 2, 3, 4, 5\}$

Training Set $= \{6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17\}$

(a) What value of k minimizes the training set error? What is the resulting training error? Assume that for the purpose of computing nearest neighbors for the training set, a point can be its own neighbor.

I will now compute the distances between every point for each of the five test points. Then I will sort these distances to obtain the best k value. Note that after sorting the distances, I assume that the test points can be their own neighbor, thus in this case, the distance will be zero.

(i) Calculating the distances between every point for each of the five cases.

1(-)				2(+)				3(+)				4(+)				5(-)			
	1	5			4	7			6	2			9	4			5	5	
Test 1	X	Y	Distance	Test 2	X	Y	Distance	Test 3	X	Y	Distance	Test 4	X	Y	Distance	Test 5	X	Y	Distance
2(+)	4	7	3.605551	1(-)	1	5	3.605551	1(-)	1	5	5.830952	1(-)	1	5	8.062258	1(-)	1	5	4
3(+)	6	2	5.830952	3(+)	6	2	5.385165	2(+)	4	7	5.385165	2(+)	4	7	5.830952	2(+)	4	7	2.236068
4(+)	9	5	8	4(+)	9	5	5.385165	4(+)	9	5	4.242641	3(+)	6	2	3.605551	3(+)	6	2	3.162278
5(-)	5	5	4	5(-)	5	5	2.236068	5(-)	5	5	3.162278	5(-)	5	5	4.123106	4(+)	9	5	4
6(-)	2	6	1.414214	6(-)	2	6	2.236068	6(-)	2	6	5.656854	6(-)	2	6	7.28011	6(-)	2	6	3.162278
7(+)	2	7	2.236068	7(+)	2	7	2	7(+)	2	7	6.403124	7(+)	2	7	7.615773	7(+)	2	7	3.605551
8(-)	3	7	2.828427	8(-)	3	7	1	8(-)	3	7	5.830952	8(-)	3	7	6.708204	8(-)	3	7	2.828427
9(-)	4	8	4.242641	9(-)	4	8	1	9(-)	4	8	6.324555	9(-)	4	8	6.403124	9(-)	4	8	3.162278
10(+)	5	9	5.656854	10(+)	5	9	2.236068	10(+)	5	9	7.071068	10(+)	5	9	6.403124	10(+)	5	9	4
11(-)	8	8	7.615773	11(-)	8	8	4.123106	11(-)	8	8	6.324555	11(-)	8	8	4.123106	11(-)	8	8	4.242641
12(+)	5	4	4.123106	12(+)	5	4	3.162278	12(+)	5	4	2.236068	12(+)	5	4	4	12(+)	5	4	1
13(+)	8	4	7.071068	13(+)	8	4	5	13(+)	8	4	2.828427	13(+)	8	4	1	13(+)	8	4	3.162278
14(+)	7	3	6.324555	14(+)	7	3	5	14(+)	7	3	1.414214	14(+)	7	3	2.236068	14(+)	7	3	2.828427
15(+)	5	1	5.656854	15(+)	5	1	6.082763	15(+)	5	1	1.414214	15(+)	5	1	5	15(+)	5	1	4
16(-)	7	2	6.708204	16(-)	7	2	5.830952	16(-)	7	2	1	16(-)	7	2	2.828427	16(-)	7	2	3.605551
17(-)	8	3	7.28011	17(-)	8	3	5.656854	17(-)	8	3	2.236068	17(-)	8	3	1.414214	17(-)	8	3	3.605551

(ii) Sorting the distances in descending order, such that the largest distances are on the top and the shortest distances are on the bottom of the lists.

	p=+	n=-							
	1		2		3		4		5
p	8	p	6.082763	p	7.071068	n	8.062258	n	4.242641
n	7.615773	n	5.830952	p	6.403124	p	7.615773	p	4
n	7.28011	n	5.656854	n	6.324555	n	7.28011	p	4
p	7.071068	p	5.385165	n	6.324555	n	6.708204	p	4
n	6.708204	p	5.385165	n	5.830952	n	6.403124	n	4
p	6.324555	p	5	n	5.830952	p	6.403124	n	3.605551
p	5.830952	p	5	n	5.656854	p	5.830952	n	3.605551
p	5.656854	n	4.123106	p	5.385165	p	5	p	3.605551
p	5.656854	n	3.605551	p	4.242641	n	4.123106	p	3.162278
n	4.242641	p	3.162278	n	3.162278	n	4.123106	p	3.162278
p	4.123106	n	2.236068	p	2.828427	p	4	n	3.162278
p	4	n	2.236068	p	2.236068	p	3.605551	n	3.162278
p	3.605551	p	2.236068	n	2.236068	n	2.828427	n	2.828427
n	2.828427	p	2	p	1.414214	p	2.236068	p	2.828427
p	2.236068	n	1	p	1.414214	n	1.414214	p	2.236068
n	1.414214	n	1	n	1	p	1	p	1
n	0	p	0	p	0	p	0	n	0

From (ii), I see that test point one's shortest distances are 0 (itself), which is negative and 1.414214, which is also negative. For test point two, I see that 0 is the shortest distance (itself) with positive value. For test point three, I see that 0 is the shortest distance (itself) with positive value. For test point four, I see that 0 and 1 are the shortest distances with positive value. For test point five, I see that 0 is the shortest distance (itself) with positive value.

Thus, to determine k, I will take the average frequency of positive and negative values for the corresponding shortest distance. $(k_1 = 2 + k_2 = 1 + k_3 = 1 + k_4 = 2 + k_5 = 1)/2 = 1.4 \implies k = 1$.

Thus, due to the idea that a test point can be its own neighbor, then the resulting training error will be zero, because for $k = 1$, I will always make the correct prediction of the test point being positive or negative, with k=1.

(b) In Figure 1, the points labelled as 1, 2, 3, 4, 5 are test points, whereas all other points are training points. Using the k you obtained from question (a), what is the test error for the data points marked as 1, 2, 3, 4, 5 ? Clearly mention which points are misclassified. Note that a test point cannot be its own nearest neighbor as neighbors must belong to the training set.

From part a, I assumed that for the purpose of computing nearest neighbors for the training set, a point can be its own neighbor. As a result I obtained that the optimal k value is $k = 1$. However without the assumption that a point can be its own neighbor I see that test error for the data points marked as 1,2,3,4,5 changes.

Recall the following distances from the sorted chart above. Here are the final distances, i.e. the shortest lengths between points.

	p=+	n=-							
	1		**2**		**3**		**4**		**5**
p	8	p	6.082763	p	7.071068	n	8.062258	n	4.242641
n	7.615773	n	5.830952	p	6.403124	p	7.615773	p	4
n	7.28011	n	5.656854	n	6.324555	n	7.28011	p	4
p	7.071068	p	5.385165	n	6.324555	n	6.708204	p	4
n	6.708204	p	5.385165	n	5.830952	n	6.403124	n	4
p	6.324555	p	5	n	5.830952	p	6.403124	n	3.605551
p	5.830952	p	5	n	5.656854	p	5.830952	n	3.605551
p	5.656854	n	4.123106	p	5.385165	p	5	p	3.605551
p	5.656854	n	3.605551	p	4.242641	n	4.123106	p	3.162278
n	4.242641	p	3.162278	n	3.162278	n	4.123106	p	3.162278
p	4.123106	n	2.236068	p	2.828427	p	4	n	3.162278
p	4	n	2.236068	p	2.236068	p	3.605551	n	3.162278
p	3.605551	p	2.236068	n	2.236068	n	2.828427	n	2.828427
n	2.828427	p	2	p	1.414214	p	2.236068	p	2.828427
p	2.236068	n	1	p	1.414214	n	1.414214	p	2.236068
n	1.414214	n	1	n	1	p	1	p	1
n	0	p	0	p	0	p	0	n	0

Here, using the assumption that test point cannot be their own neighbors, I can conclude the following:

Point 1: Correct prediction at k=1.

Point 2: Incorrect prediction at k=1.

Point 3: Incorrect prediction at k=1.

Point 4: Correct prediction at k=1.

Point 5: Incorrect prediction at k=1.

$\implies \frac{2}{5} = 0.4$ or %40 correct or %60 incorrect prediction using k=1.

Thus, I can conclude that for k=1, there exists a %60 test error. The points that are misclassified are points; 2, 3 and 5. The points that are correctly classified are points; 1 and 4.

In the above parts, we had considered k-nearest neighbor classifier using Euclidean distance metric. For the next part, we will choose *cosine similarity* as a distance metric. We define cosine similarity between two vectors u, v as:

$$sim(u, v) = \frac{\langle u, v \rangle}{||u||_2 ||v||_2}$$

where $< \cdot, \cdot >$ is the inner product operator between two vectors, and $||.||_2$ is the L_2 norm operator.

Note that for cosine similarity, we consider the nearest neighbors to be the points in the training set with the greatest cosine similarity (i.e. most similar points) to the test point.

[Hint: "Cosine similarity is the cosine of the angle between the vectors; that is, it is the dot product of the vectors divided by the product of their lengths. It follows that the cosine similarity does not depend on the magnitudes of the vectors, but only on their angle."]

cosine similarity $:= cos(\theta) = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} * \sqrt{\sum_{i=1}^{n} v_i^2}}$

(c) Using the k you choose from question (a), what is the test error for the data points marked as 1, 2, 3, 4, 5 in Figure 1? Clearly mention which points are mis-classified. (Assume that a point is not its own neighbor).

Here, using the assumption that test point cannot be their own neighbors, as-well as using the cosine similarity I can conclude the following:

	p=+	n=-						
	1		**2**		**3**		**4**	**5**
p	0.996729048	p	0.999927	n	0.999314834	p	0.999911	n 1
n	0.992277877	n	0.99846	n	0.999056158	p	0.998969	p 0.993883735
n	0.978549785	n	0.99348	p	0.996545758	n	0.998233	p 0.964763821
n	0.964763821	n	0.980581	p	0.992277877	p	0.99535	p 0.961523948
p	0.952424147	p	0.971136	p	0.989949494	n	0.990227	p 0.961523948
p	0.948683298	p	0.964764	p	0.982872187	p	0.975716	n 0.948683298
n	0.832050294	n	0.964764	p	0.938343117	p	0.96728	p 0.948683298
n	0.832050294	n	0.948683	n	0.894427191	n	0.933346	p 0.928476691
p	0.765704865	p	0.929807	n	0.894427191	n	0.933346	n 0.928476691
p	0.64764842	p	0.85536	p	0.745241314	p	0.806004	n 0.910366477
p	0.613940614	p	0.83205	p	0.73715414	p	0.798815	p 0.894427191
p	0.566528823	p	0.798041	n	0.707106781	n	0.77193	n 0.894427191
n	0.527934121	n	0.76941	n	0.664363839	n	0.733268	p 0.874157276
p	0.496138938	p	0.745241	n	0.6	n	0.674269	n 0.874157276
n	0.45795659	n	0.715574	p	0.564683916	n	0.641555	p 0.832050294
p	0.384615385	p	0.656781	n	0.496138938	p	0.577465	n 0.832050294
n		p		p		p		n

Point 1: Incorrect prediction at k=1.

Point 2: Correct prediction at k=1.

Point 3: Incorrect prediction at k=1.

Point 4: Correct prediction at k=1.

Point 5: Correct prediction at k=1.

Thus test error is $\frac{2}{5} = 0.4$ or %40, where points 1 and 3 were mis-classified and points 2, 4 and 5 were correctly classified.

Problem 4 (Perception and Logistic Regression)

.

Suppose we have a training set with 6 samples, each sample has feature vector in $\mathbb{R}^3$, and every feature vector $\boldsymbol{x}^{(i)}$ has been augmented with a bias term such that $\boldsymbol{x}_0^{(i)} = 1$ for all i:

i	1	2	3	4	5	6
$\boldsymbol{x}^{(i)}$	[1, 4,0]	[1, 1,1]	[1, -2,1]	[1, 1,0]	[1, 5,2]	[1, 3, -1]
$y^{(i)}$	1	-1	1	1	-1	-1

We are going to use perceptron algorithm to train a linear classifier with 3 dimensional weight vector $\boldsymbol{\theta} \in \mathbb{R}^3$ (with bias term). We start with initial weight vector as the first sample in our dataset, i.e. $\boldsymbol{\theta} = \boldsymbol{x}^{(1)}$. Note: when $\boldsymbol{\theta}^T = 0$, we will assume the algorithm predicts $+1$.

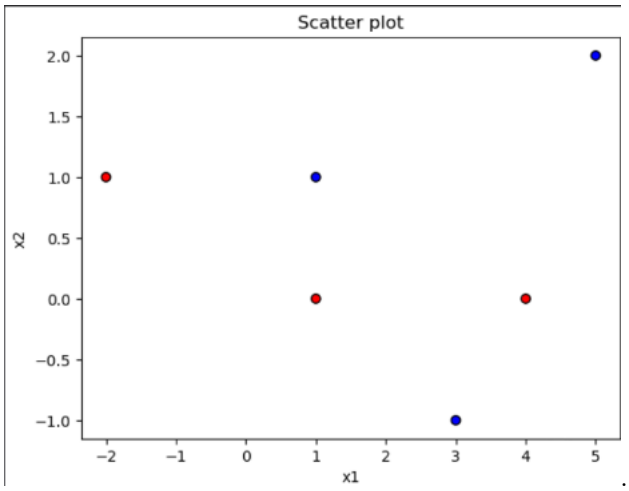
1. Plot the data in Python using matplotlib with x_1 and x_2 as the two axis (ignoring the bias term $\boldsymbol{x}_0^{(i)} = 1$). Use different colors to denote $+1/-1$ labels. Paste a screenshot of your plot in the writeup.

   Code:

```
[2]: import matplotlib as mpl
     import matplotlib.pyplot as plt
     import numpy as np

[3]: x1 = np.array([4,1,-2,1,5,3])
     x2 = np.array([0,1,1,0,2,-1])
     y = np.array([1,-1,1,1,-1,-1])

[7]: plt.scatter(x1, x2, c=y, cmap='bwr', edgecolors='k')
     plt.xlabel('x1')
     plt.ylabel('x2')
     plt.title('Scatter plot')
     plt.show()
```

   .

   Output:



   .

   Such that the blue dots represent the negative values and the red dots represent the positive values

2. Is the data linearly separable? Will our algorithm converge if we run it several times over the same sequence? Explain.

   Recall the following definitions and statments from lecture:

(i) A dataset is linearly separable if there exists some hyperplane that puts all the positive examples on one side and all the negative examples on the other side

(ii) From the Convergence theorem, if there exists a set of perception parameters such that the data is linearly separable, the perceptron algorithm will converge.

(iii) From the Cycling Theorem, if the training data is not linearly separable, then the learning algorithm will eventually repeat the same set of parameters and enter an infinite loop.

From the (i) I can conclude that the data-set is not linearly separable because the hyperplane does not put all the positive and negative examples on one side and all the negative examples on the other side. If I were to plot these points in a 3D space, the dots representing the two classes would not sit in two distinct non-overlapping regions separated by a plane. The reason the perception didn't converge is that it will keep trying to find the best position for this plane, adjusting every-time it found a dot on the wrong side, however since no perfect position exists, the perception will adjust indefinitely.

3. Regardless of whether the dataset is linearly separable or not, derive and calculate (by hand) the updates of the parameters on this dataset for one round over the entire dataset using a learning rate of $\alpha = 0.1$.

   Recall that the algorithm for perceptron loss is the following gradient descent :

   Given training data $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
   Let $\boldsymbol{\theta} \leftarrow [0, 0, \ldots, 0]$
   Repeat:
   for $i = 1 \ldots n$, do
   if $y^{(i)}\boldsymbol{\theta}^T\boldsymbol{x}^{(i)} \leq 0$ // prediction for i^{th} instance is incorrect
   $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha y^{(i)}\boldsymbol{x}^{(i)}$
   Return $\boldsymbol{\theta}$

   Initialization: $\theta \longleftarrow [0, 0, 0]$

   $i = 1 : y^{(1)}(\theta^T x^{(1)}) = 1([0, 0, 0] * [1, 4, 0]) = 0 \leq 0$

   Update: $\theta \longleftarrow [0, 0, 0] + 0.1 * 1 * [1, 4, 0] = [0.1, 0.4, 0]$

   $i = 2 : y^{(2)}(\theta^T x^{(2)}) = -1([0.1, 0.4, 0] * [1, 1, 1]) = -0.5 \leq 0$

   Update: $\theta \longleftarrow [0.1, 0.4, 0.] - 0.1 * [1, 1, 1] = [0, 0.3, -0.1]$

   $i = 3 : y^{(3)}(\theta^T x^{(3)}) = 1([0, 0.3, -0.1] * [1, -2, 1]) = -0.7$

   Update: $\theta \longleftarrow [0, 0.3, -0.1] + 0.1 * 1 * [1, -2, 1] = [0.1, 0.1, 0]$

   $i = 4 : y^{(4)}(\theta^T x^{(4)}) = 1([0.1, 0.1, 0] * [1, 1, 0]) = 0.2 > 0$. Hence, no update.

   $i = 5 : y^{(5)}(\theta^T x^{(5)}) = -1([0.1, 0.1, 0] * [1, 5, 2]) = -0.6 \leq 0$

   Update: $[0.1, 0.1, 0] - 0.1 * [1, 5, 2] = [0, -0.4, -0.2]$.

   $i = 6 : y^{(6)}(\theta^T x^{(6)}) = -1([0, -0.4, -0.2] * [1, 3, -1]) = 1.4 > 0$. Hence no update.

   Thus, after one pass, $\theta = [0, -0.4, -0.2]$.

4. Now, let's adopt a probabilistic perspective to tackle this binary classification problem using logistic regression. Given a logistic regression model for binary classification, the hypothesis is represented as

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $g(z) = \frac{1}{1+e^{-z}}$. Here $\boldsymbol{x}$ is the feature vector (with $x_0 = 1$ for bias) and $\boldsymbol{\theta}$ is the parameter vector. As learned in class, the loss function for logistic regression is:

$J(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)} \log h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)\right)\right]$

Suppose we are given parameter $\boldsymbol{\theta} = \begin{bmatrix}1 & 1 & 1\end{bmatrix}^T$,

(a) Compute $h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})$ for each sample in the training set.

$$\theta^T(x^{(1)}) = [1\ 1\ 1]\begin{bmatrix}1\\4\\0\end{bmatrix} = 5 \implies h_\theta(x) = g(\theta^T x) = \tfrac{1}{1+e^{-5}}$$

$$\theta^T(x^{(2)}) = [1\ 1\ 1]\begin{bmatrix}1\\1\\1\end{bmatrix} = 3 \implies h_\theta(x) = g(\theta^T x) = \tfrac{1}{1+e^{-3}}$$

$$\theta^T(x^{(3)}) = [1\ 1\ 1]\begin{bmatrix}1\\-2\\1\end{bmatrix} = 0 \implies h_\theta(x) = g(\theta^T x) = \tfrac{1}{1+e^{0}}$$

$$\theta^T(x^{(4)}) = [1\ 1\ 1]\begin{bmatrix}1\\1\\0\end{bmatrix} = 2 \implies h_\theta(x) = g(\theta^T x) = \tfrac{1}{1+e^{-2}}$$

$$\theta^T(x^{(5)}) = [1\ 1\ 1]\begin{bmatrix}1\\5\\2\end{bmatrix}) = 8 \implies h_\theta(x) = g(\theta^T x) = \tfrac{1}{1+e^{-8}}$$

$$\theta^T(x^{(6)}) = [1\ 1\ 1]\begin{bmatrix}1\\3\\-1\end{bmatrix} = 3 \implies h_\theta(x) = g(\theta^T x) = \tfrac{1}{1+e^{-3}}$$

(b) Compute $J(\boldsymbol{\theta})$ for the training set.

Note: When referring to the logarithm, we mean the natural logarithm. For the purpose of calculating the logistic regression loss, map the class label $y = -1$ to $y = 0$, while the class label $y = 1$ remains as $y = 1$.

Using the definition from (i), and the computed $h_\theta(x)$, I see that the loss calculation is the following:

$J(\theta) = \sum_{i=1}^{6}[ylog(h_\theta(x)) + (1-y)log(1-h_\theta(x))]$

$i = 1 \implies [1 * log(\frac{1}{1+e^{-5}})] = -0.006715$

$i = 2 \implies [log(1 - \frac{1}{1+e^{-3}})] = -3.0485874$

$i = 3 \implies [1 * log(\frac{1}{1+e^{0}})] = -0.6931472$

$i = 4 \implies [1 * log(\frac{1}{1+e^{-2}})] = -0.126928$

$i = 5 \implies [log(1 - \frac{1}{1+e^{-8}})] = -8.000335$

$i = 6 \implies [log(1 - \frac{1}{1+e^{-3}})] = -3.048587$

$\implies -\frac{1}{6}(-0.006715 - 3.0485874 - 0.6931472 - 0.126928 - 8.000335 - 3.048587) = 2.487$

Problem 5 (Programming Exercise: Applying Classification Models)

.

In this problem, we ask you to compare the classification models we have studied till now i.e., Decision trees, K-nearest neighbors, and Logistic regression.

This data was extracted from the 1994 Census bureau database by Ronny Kohavi and Barry Becker. For computational reasons, we have already extracted a relatively clean subset of the data for this homework. The prediction task is to determine whether a person makes over $50K a year.

In this problem, we ask you to complete the analysis of what sorts of people were likely to earn more than $50K a year. In particular, we ask you to apply the tools of machine learning to predict which individuals are more likely to have high income.

Starter Files

code and data

documentation

- Decision Tree Classifier:
  http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- K-Nearest Neighbor Classifier:
  http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- Logistic Regression:
  https://scikit-learn.org/stable/modules/generated/sklearn.linear$_model.LogisticRegression.htmlCross-Validation$:
  $https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html$
  $https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html$

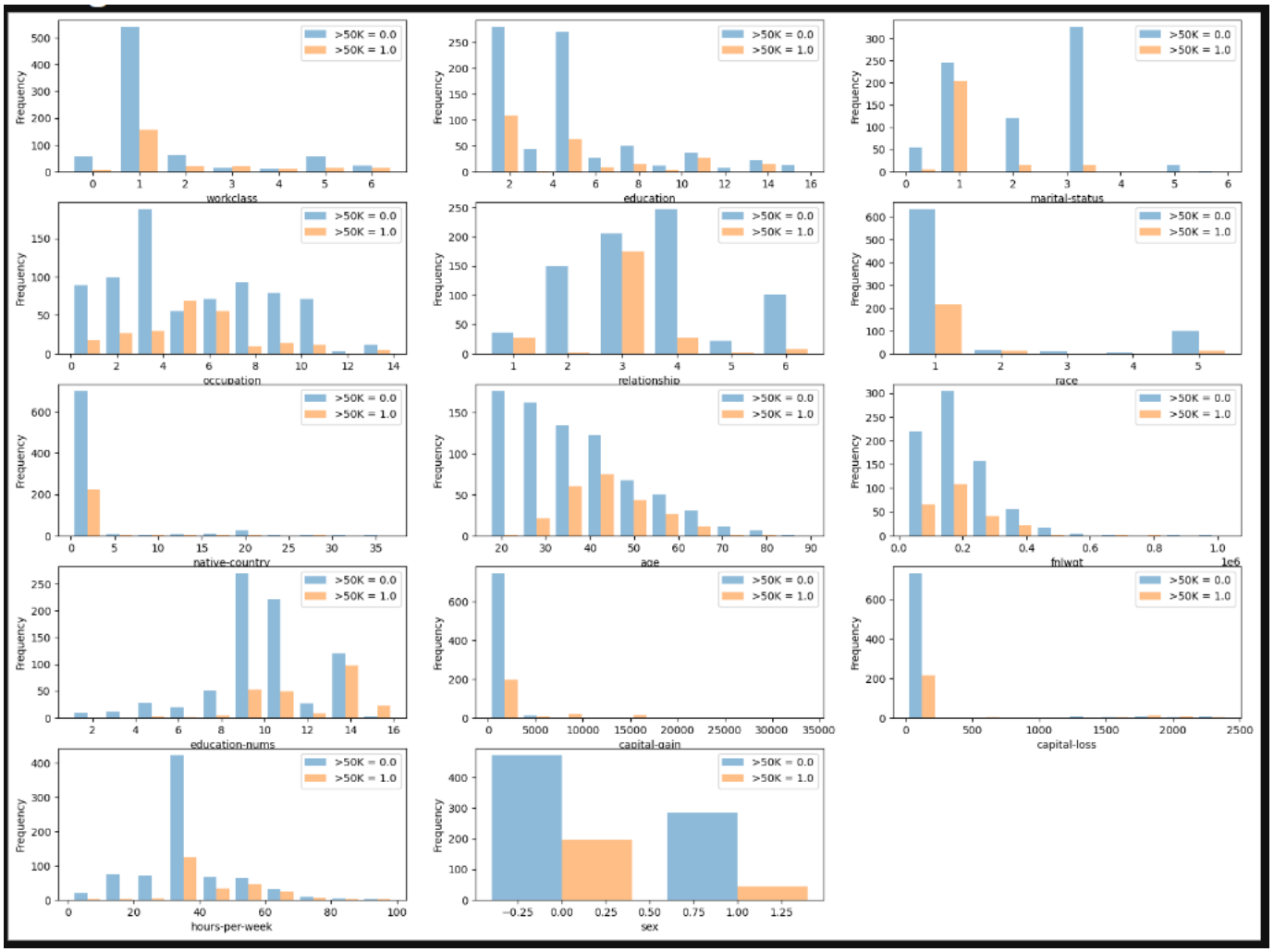
- Metrics:
  http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy$_score.html$
  $https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html$

Visualization

One of the first things to do before trying any formal machine learning technique is to dive into the data. This can include looking for funny values in the data, looking for outliers, looking at the range of feature values, what features seem important, etc.

Note: We have already converted all the categorical features to numerical ones. The target column is the last one: ">50k", where 1 and 0 indicate >50k or $\leq$ 50k respectively. The feature "fnlwgt" describes the number of people the census believes the entry represents. All the other feature names should be self-explanatory. If you want to learn more about this dataset, please click https://archive.ics.uci.edu/ml/datasets/adulthere.

(a) Plot and submit histograms for each feature, separating the examples by class (e.g. income greater than 50k or smaller than or equal to 50k). This should produce fourteen plots, one for each feature. Each plot should have two overlapping histograms, with the color of the histogram indicating the class.



Evaluation

Now, let us use `scikit-learn` to train a `DecisionTreeClassifier`, `KNeighborsClassifier`, and `LogisticRegression` on the data.

Using the predictive capabilities of the `scikit-learn` package is very simple. In fact, it can be carried out in three simple steps: initializing the model, fitting it to the training data, and predicting new values.[1]

[1]Note that almost all of the model techniques in `scikit-learn` share a few common named functions, once they are initialized. You can always find out more about them in the documentation for each model. These are `some-model-name.fit(...)`, `some-model-name.predict(...)`, and `some-model-name.score(...)`.

(b) Before trying out any classifier, it is often useful to establish a *baseline*. We have implemented one simple baseline classifier, `MajorityVoteClassifier`, that always predicts the majority class from the training set. Read through the `MajorityVoteClassifier` and its usage and make sure you understand how it works.

Code:

```
class Classifier(object) :
    """
    Classifier interface.
    """

    def fit(self, X, y):
        raise NotImplementedError()

    def predict(self, X):
        raise NotImplementedError()


class MajorityVoteClassifier(Classifier) :

    def __init__(self) :
        """
        A classifier that always predicts the majority class.

        Attributes
        --------------------
            prediction_ -- majority class
        """
        self.prediction_ = None

    def fit(self, X, y) :
        """
        Build a majority vote classifier from the training set (X, y).

        Parameters
        --------------------
            X    -- numpy array of shape (n,d), samples
            y    -- numpy array of shape (n,), target classes

        Returns
        --------------------
            self -- an instance of self
        """
        majority_val = Counter(y).most_common(1)[0][0]
        self.prediction_ = majority_val
        return self

    def predict(self, X) :
        """
        Predict class values.

        Parameters
        --------------------
            X    -- numpy array of shape (n,d), samples

        Returns
        --------------------
            y    -- numpy array of shape (n,), predicted classes
        """
        if self.prediction_ is None :
            raise Exception("Classifier not initialized. Perform a fit first.")

        n,d = X.shape
        y = [self.prediction_] * n
        return y
```

Ouput:

```
#=====================================
# train Majority Vote classifier on data
print('Classifying using Majority Vote...')
clf = MajorityVoteClassifier() # create MajorityVote classifier, which includes all model parameters
clf.fit(X, y)                  # fit training data using the classifier
y_pred = clf.predict(X)        # take the classifier and run it on the training data
train_error = 1 - metrics.accuracy_score(y, y_pred, normalize=True)
print('\t-- training error: %.3f' % train_error)
```

```
Classifying using Majority Vote...
        -- training error: 0.240
```

Your goal is to implement and evaluate another baseline classifier, `RandomClassifier`, that predicts a target class according to the distribution of classes in the training data set. For example, if 85% of the examples in the training set have `>50k = 0` and 15% have `>50k = 1`, then, when applied to a test set, `RandomClassifier` should randomly predict 85% of the examples as `>50k = 0` and 15% as `>50k = 1`.

Implement the missing portions of `RandomClassifier` according to the provided specifications. Then train your `RandomClassifier` on the entire training data set, and evaluate its training error. If you implemented everything correctly, you should have an error of 0.385 or 0.374, depending on an implementation detail; both error values are correct. Paste a screenshot of your code.

Code:

```python
class RandomClassifier(Classifier) :

    def __init__(self) :
        """
        A classifier that predicts according to the distribution of the classes.

        Attributes
        --------------------
            probabilities_ -- class distribution dict (key = class, val = probability of class)
        """
        self.probabilities_ = None

    def fit(self, X, y) :
        """
        Build a random classifier from the training set (X, y).

        Parameters
        --------------------
            X    -- numpy array of shape (n,d), samples
            y    -- numpy array of shape (n,), target classes

        Returns
        --------------------
            self -- an instance of self
        """

        ### ========== TODO : START ========== ###
        # part b: set self.probabilities_ according to the training set
        unique, counts = np.unique(y, return_counts=True)
        total=y.shape[0]
        self.probabilities_ = dict(zip(unique, counts / total))

        ### ========== TODO : END ========== ###

        return self

    def predict(self, X, seed=1234) :
        """
        Predict class values.

        Parameters
        --------------------
            X    -- numpy array of shape (n,d), samples
            seed -- integer, random seed

        Returns
        --------------------
            y    -- numpy array of shape (n,), predicted classes
        """
        if self.probabilities_ is None :
            raise Exception("Classifier not initialized. Perform a fit first.")
        np.random.seed(seed)

        ### ========== TODO : START ========== ###
        # part b: predict the class for each test example
        # hint: use np.random.choice (be careful of the parameters)
        classes = list(self.probabilities_.keys())
        probs = list(self.probabilities_.values())

        y=np.random.choice(classes, size=X.shape[0], p=probs)

        ### ========== TODO : END ========== ###

        return y
```

Output:

```
### ========== TODO : START ========== ###
# part b: evaluate training error of Random classifier
clf = RandomClassifier()

clf.fit(X, y)
y_pred = clf.predict(X)

incorrect_predictions = np.sum(y_pred != y)
training_error = incorrect_predictions / len(y)

print(f"Training Error of Random Classifier: {training_error:.3f}")
### ========== TODO : END ========== ###
```

```
Training Error of Random Classifier: 0.374
```

(c) Now that we have a baseline, train and evaluate a `DecisionTreeClassifier` (using the `DecisionTreeClassifier` class from `scikit-learn` and referring to the documentation as needed). Make sure you initialize your classifier with the appropriate parameters; in particular, use the 'entropy' criterion discussed in class. What is the training error of this classifier? Paste a screenshot of your code.

```
### ========== TODO : START ========== ###
# part c: evaluate training error of Decision Tree classifier

from sklearn.metrics import accuracy_score

clf = DecisionTreeClassifier(criterion="entropy")

clf.fit(X, y)

y_pred = clf.predict(X)

training_accuracy = accuracy_score(y, y_pred)
training_error = 1 - training_accuracy

print(f"Training Error of Decision Tree Classifier {training_error:.3f}")

### ========== TODO : END ========== ###
```

```
Training Error of Decision Tree Classifier 0.000
```

(d) Similar to the previous question, train and evaluate a `KNeighborsClassifier` (using the class from `scikit-learn` and referring to the documentation as needed). Use k=3, 5 and 7 as the number of neighbors and report the training error of this classifier respectively. Paste a screenshot of your code.

Code:

```python
### ========== TODO : START ========== ###
# part d: evaluate training error of k-Nearest Neighbors classifier
# use k = 3, 5, 7 for n_neighbors

from sklearn.metrics import accuracy_score

k_values = [3,5,7]

for i in k_values:

    clf = KNeighborsClassifier(n_neighbors = i)

    clf.fit(X,y)

    y_pred = clf.predict(X)

    training_accuracy = accuracy_score(y,y_pred)
    training_error = 1 - training_accuracy

    print(f"Training Error of KNN Classifier k=  { i } : {training_error:.3f}")



### ========== TODO : END ========== ###
```

```
Training Error of KNN Classifier k=  3 : 0.153
Training Error of KNN Classifier k=  5 : 0.195
Training Error of KNN Classifier k=  7 : 0.213
```

(e) Similar to the previous question, train and evaluate a `LogisticRegression` (using the class from `scikit-learn` and referring to the documentation as needed). Use λ=0.1, 1 and 10 as the regularization hyperparameter and report the training error of this classifier respectively. Make sure you initialize your classifier with the appropriate parameters; `random_state`=0 and `max_iter`=1000. (*Hint*: function argument `C` is the inverse of regularization strength, therefore `C` $= 1/\lambda$.) Paste a screenshot of your code.

```
### ========== TODO : START ========== ###
# part e: evaluate training error of Logistic Regression
# use lambda_ = 0.1, 1, 10 for n_neighbors
# Note: Make sure you initialize your classifier with the appropriate parameters:
# random_state=0 and max_iter=1000, using the default solver is fine
from sklearn.metrics import accuracy_score

lambdas = [0.1,1,10]

for lambda_val in lambdas:

    clf = LogisticRegression(C=1/lambda_val, random_state=0, max_iter=1000)

    clf.fit(X,y)

    y_pred = clf.predict(X)

    training_accuracy = accuracy_score(y,y_pred)
    training_error = 1 - training_accuracy

    print(f"Training Error of Logistic Regression with lambda= { lambda_val }: {training_error:.3f}")
### ========== TODO : END ========== ###
```

```
Training Error of Logistic Regression with lambda= 0.1: 0.208
Training Error of Logistic Regression with lambda= 1: 0.208
Training Error of Logistic Regression with lambda= 10: 0.213
```

(f) So far, we have looked only at training error, but as we learned in class, training error is a poor metric for evaluating classifiers. Let us use cross-validation instead.

Implement the missing portions of `error(...)` according to the provided specifications. You may find it helpful to use `StratifiedShuffleSplit(...)` from `scikit-learn`. To ensure that we always get the same splits across different runs (and thus can compare the classifier results), set the `random_state` parameter to be the same (e.g., 0).

Read up about F1 scores. Next, use your `error(...)` function to evaluate the training error and (cross-validation) validation error and validation micro averaged F1 Score of the `RandomClassifier`, `DecisionTreeClassifier`,`KNeighborsClassifier`, and `LogisticRegression` models. For the `DecisionTreeClassifi` use 'entropy' criterion, for the `KNeighborsClassifier`, use $k = 5$, for the `LogisticRegression`, use $\lambda = 1$, `random_state`=0 and `max_iter`=1000. To do this, generate a random 85/15 split of the training data, train each model on the 85% fraction, evaluate the error on both the 85% and the 15% fraction, and repeat this splitting 100 times to get an average result. What are the average training error, validation error, and validation F1 score of each of your classifiers on the adult_subsample data set? Paste a screenshot of your code.

```
#####################################################################
# Mutatble functions
#####################################################################

from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.metrics import accuracy_score, f1_score

def error(clf, X, y, ntrials=100, test_size=0.15) :
    """
    Computes the classifier error over a random split of the data,
    averaged over ntrials runs.

    Parameters
    --------------------
        clf         -- classifier
        X           -- numpy array of shape (n,d), features values
        y           -- numpy array of shape (n,), target classes
        ntrials     -- integer, number of trials

    Returns
    --------------------
        train_error -- float, training error
        val_error  -- float, validation error
        f1_score    -- float, validation "micro" averaged f1 score
    """

    ### ========== TODO : START ========== ###
    # part f:
    # compute cross-validation error using StratifiedShuffleSplit over ntrials
    # hint: use StratifiedShuffleSplit (be careful of the parameters)

    sss = StratifiedShuffleSplit(n_splits=ntrials, test_size=test_size, random_state=0)

    train_errors = []
    val_errors = []
    f1_scores = []

    for train_index, val_index in sss.split(X,y):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        clf.fit(X_train, y_train)

        y_train_pred = clf.predict(X_train)
        train_errors.append(1-accuracy_score(y_train, y_train_pred))

        y_val_pred = clf.predict(X_val)
        val_errors.append(1-accuracy_score(y_val, y_val_pred))

        f1_scores.append(f1_score(y_val, y_val_pred, average ='micro'))



    train_error = sum(train_errors) / ntrials
    val_error = sum(val_errors) / ntrials
    avg_f1_score = sum(f1_scores) / ntrials

    pass
    ### ========== TODO : END ========== ###

    return train_error, val_error, avg_f1_score
```

```python
### ========== TODO : START ========== ###
# part f: use cross-validation to compute average training and validation error of classifiers
print('Investigating various classifiers...')

clf_majority = MajorityVoteClassifier()
train_error, val_error, f1 = error(clf_majority, X, y)
print(f"MajorityVoteClassifier -> Training Error: {train_error:.3f}, Validation Error: {val_error:.3f}, F1 Score: {f1:.3f}")

clf_random = RandomClassifier()
train_error, val_error, f1 = error(clf_random, X, y)
print(f"RandomClassifier -> Training Error: {train_error:.3f}, Validation Error: {val_error:.3f}, F1 Score: {f1:.3f}")

clf_tree = DecisionTreeClassifier(criterion='entropy')
train_error, val_error, f1 = error(clf_tree, X, y)
print(f"DecisionTreeClassifier -> Training Error: {train_error:.3f}, Validation Error: {val_error:.3f}, F1 Score: {f1:.3f}")

clf_knn = KNeighborsClassifier(n_neighbors=5)
train_error, val_error, f1 = error(clf_knn, X, y)
print(f"KNeighbotsClassifier -> Training Error: {train_error:.3f}, Validation Error: {val_error:.3f}, F1 Score: {f1:.3f}")

clf_logistic = LogisticRegression(C=1/1, random_state=0, max_iter=1000)
train_error, val_error, f1 = error(clf_logistic, X, y)
print(f"LogisticRegression -> Training Error: {train_error:.3f}, Validation Error: {val_error:.3f}, F1 Score: {f1:.3f}")
### ========== TODO : END ========== ###
```

```
Investigating various classifiers...
MajorityVoteClassifier -> Training Error: 0.240, Validation Error: 0.240, F1 Score: 0.760
RandomClassifier -> Training Error: 0.372, Validation Error: 0.367, F1 Score: 0.633
DecisionTreeClassifier -> Training Error: 0.000, Validation Error: 0.202, F1 Score: 0.798
KNeighbotsClassifier -> Training Error: 0.200, Validation Error: 0.254, F1 Score: 0.746
LogisticRegression -> Training Error: 0.207, Validation Error: 0.212, F1 Score: 0.788
```

MajorityVoteClassifier : Training Error: 0.240, Validation Error: 0.240, F1 Score: 0.760

RandomClassifier : Training Error: 0.372, Validation Error: 0.367, F1 Score: 0.633

DecisionTreeClassifier : Training Error: 0.000, Validation Error: 0.202, F1 Score: 0.798

KNeighbotsClassifier : Training Error: 0.200, Validation Error: 0.254, F1 Score: 0.746

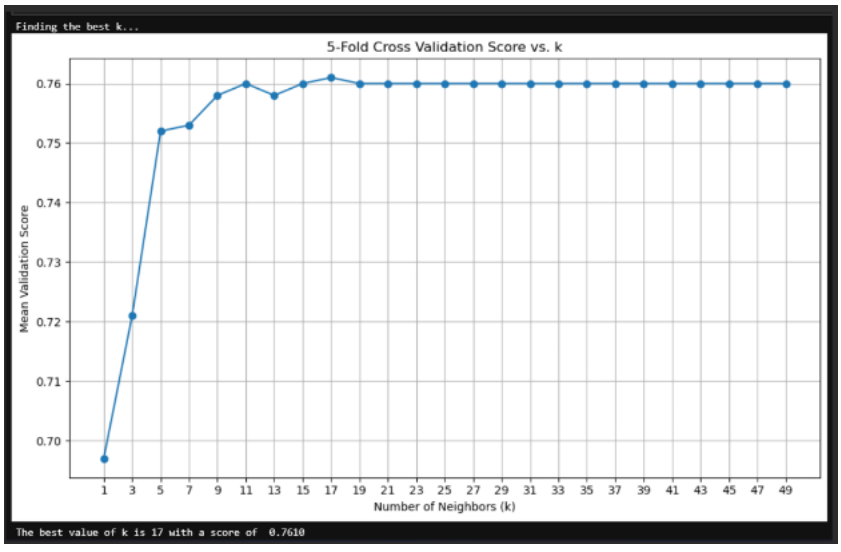
LogisticRegression : Training Error: 0.207, Validation Error: 0.212, F1 Score: 0.788

(g) One way to find out the best value of k for `KNeighborsClassifier` is n-fold cross validation. Find out the best value of k using 5-fold cross validation. You may find the `cross_val_score(...)` from `scikit-learn` helpful. Run 5-fold cross validation for all odd numbers ranging from 1 to 50 as the number of neighbors. Then plot the validation score against the number of neighbors, k. Include this plot in your writeup, and provide a 1-2 sentence description of your observations. What is the best value of k and what is the corresponding score? Paste a screenshot of your code.

Code:

```python
### ========== TODO : START ========== ###
# part g: use 5-fold cross-validation to find the best value of k for k-Nearest Neighbors classifier
print('Finding the best k...')

k_values = list(range(1,51,2))
scores = []

for k in k_values:
    knn=KNeighborsClassifier(n_neighbors=k)
    cross_val_scores = cross_val_score(knn, X, y, cv=5)
    scores.append(cross_val_scores.mean())

plt.figure(figsize=(10,6))
plt.plot(k_values, scores, marker='o', linestyle='-')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Mean Validation Score')
plt.title('5-Fold Cross Validation Score vs. k')
plt.grid(True)
plt.xticks(k_values)
plt.tight_layout()
plt.show()

best_k = k_values[np.argmax(scores)]
best_score = np.max(scores)

print(f"The best value of k is {best_k} with a score of {best_score: .4f}")


### ========== TODO : END ========== ###
```

Output:



The best value of k is 17 with a score of 0.7610

Through 5 fold cross validation over odd values of k ranging from 1 to 50, I can determine that the optimal number of neighbors for KNN Classifier to be k=17, resulting in a score of 0.76 or %76. The plot suggests that the model is sensitive to the choice of k. At k=17, there exists a balance, such that the sensitivity goes away. As the number of neighbors, k, increased from 1 to 5, there was a pronounced upward trend in the mean validation score. This suggests that using only a single neighbor might lead to an over fitted model. Between k=5 to k=17, the mean validation score experienced a more gradual rise, indicating that the models performance was stabilizing as more neighbors were considered. As

already mentioned, at k=17 the model reached equilibrium, getting rid of any overfitting.

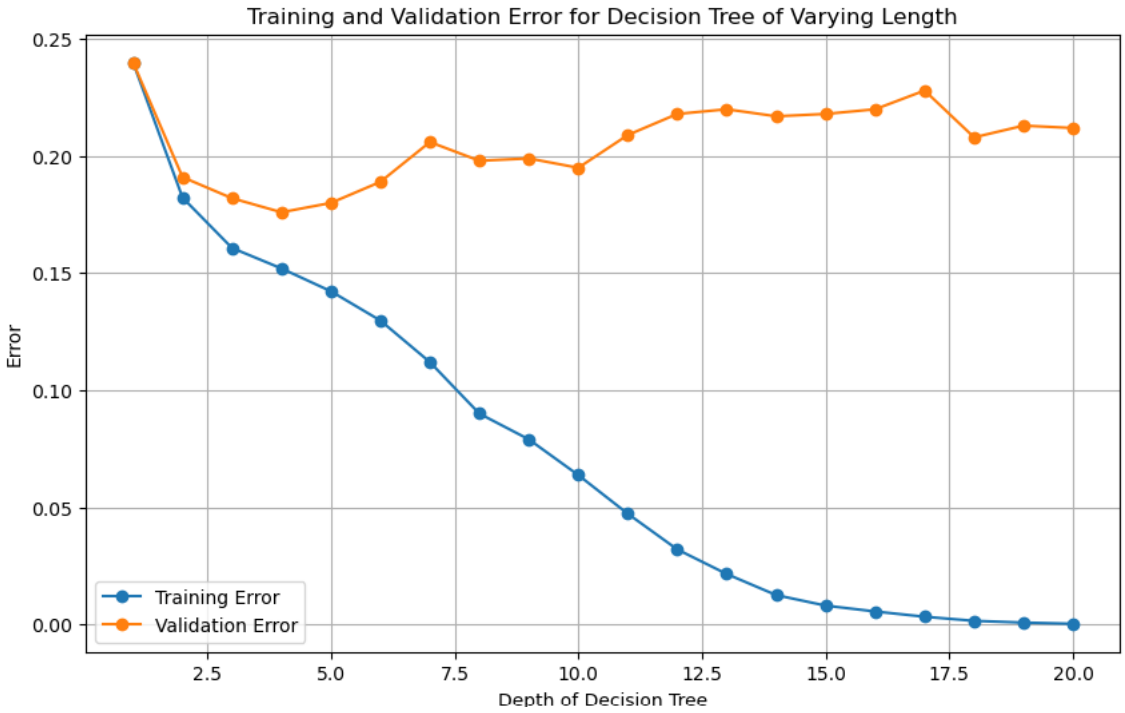
(h) One problem with decision trees is that they can *overfit* to training data, yielding complex classifiers that do not generalize well to new data. Let us see whether this is the case for the adult_subsample data.

One way to prevent decision trees from overfitting is to limit their depth. Repeat your cross-validation experiments but for increasing depth limits, specifically, $1, 2, \ldots, 20$. Then plot the average training error and validation error against the depth limit. You may find the `cross_validate(...)` from `scikit-learn` helpful; note that this function returns accuracy, while here, error $= 1 -$ accuracy. Include this plot in your writeup, making sure to label all axes and include a legend for your classifiers. What is the best depth limit to use for this data? Do you see overfitting? Justify your answers using the plot. Paste a screenshot of your code.

Code:

```
### ========== TODO : START ========== ###
# part h: investigate decision tree classifier with various depths
print('Investigating depths...')

depths = list(range(1,21))
train_errors = []
val_errors = []

for depth in depths:
    clf = DecisionTreeClassifier(max_depth=depth)
    cv_results=cross_validate(clf, X, y, cv=5, return_train_score=True)
    train_error=1-np.mean(cv_results['train_score'])
    val_error=1-np.mean(cv_results['test_score'])
    train_errors.append(train_error)
    val_errors.append(val_error)

plt.figure(figsize=(10,6))
plt.plot(depths, train_errors, label='Training Error', marker='o')
plt.plot(depths, val_errors, label='Validation Error', marker='o')
plt.xlabel('Depth of Decision Tree')
plt.ylabel('Error')
plt.title('Training and Validation Error for Decision Tree of Varying Length')
plt.legend()
plt.grid(True)
plt.show()
```

Output:



From the output, and based on the validation error, the optimal depth is approximately 3 or 4. Training error decreases with depth, but the validation error decreases then increases, indicating that the classifier is overfitting