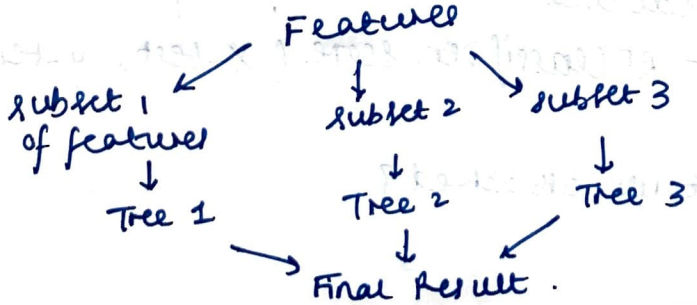
ANSWER 1

(A) • Random Forest is a type of Supervised Machine Learning Algorithm which can be used for tasks like classification & regression.

- It is an 'ensemble' learning algorithm as it aims to use multiple algorithm or multiple instances of similar algorithms in order to perform better.
- As the name suggests, Random Forest uses multiple decision trees to train on the data. It classifies the class reported by the majority of the decision trees or while regressing, it gives the mean of the predicted values by all trees.
- Decision trees on their own are susceptible to overfitting - using multiple trees which are not correlated and use different subsets of data to train on can help alleviate this problem. Low correlation between trees is done by 'feature randomness / bagging' i.e. a different subset of features is used by each tree.
- Main parameters for Random Forests include no. of trees, no. of features and node sizes.
- Since we are training using many trees, this is time consuming, complex and requires more compute power.



(B) RFClassifier = RandomForestClassifier()
RFClassifier.fit(X_train, y_train)
y_test_predicted = RFClassifier.predict(X_test)

{ Code & output attached }.

(C) FEATURE IMPORTANCE

Feature importance tells us which of the features is more relevant to the result i.e. the higher the feature importance the more relevent is that feature/variable to the result of the model.

- Formally, feature importance is computed as the 'mean and standard deviation of accumulation of the impurity decrease within each tree'. (default in sci-kit-learn)
  Another way to calculate importance is permutation importance.
- Feature importances predict which feature had the biggest impact on each prediction and hence, we can build better models by employing better data for higher impact features.

feature_importances = RFClassifier.feature_importances_

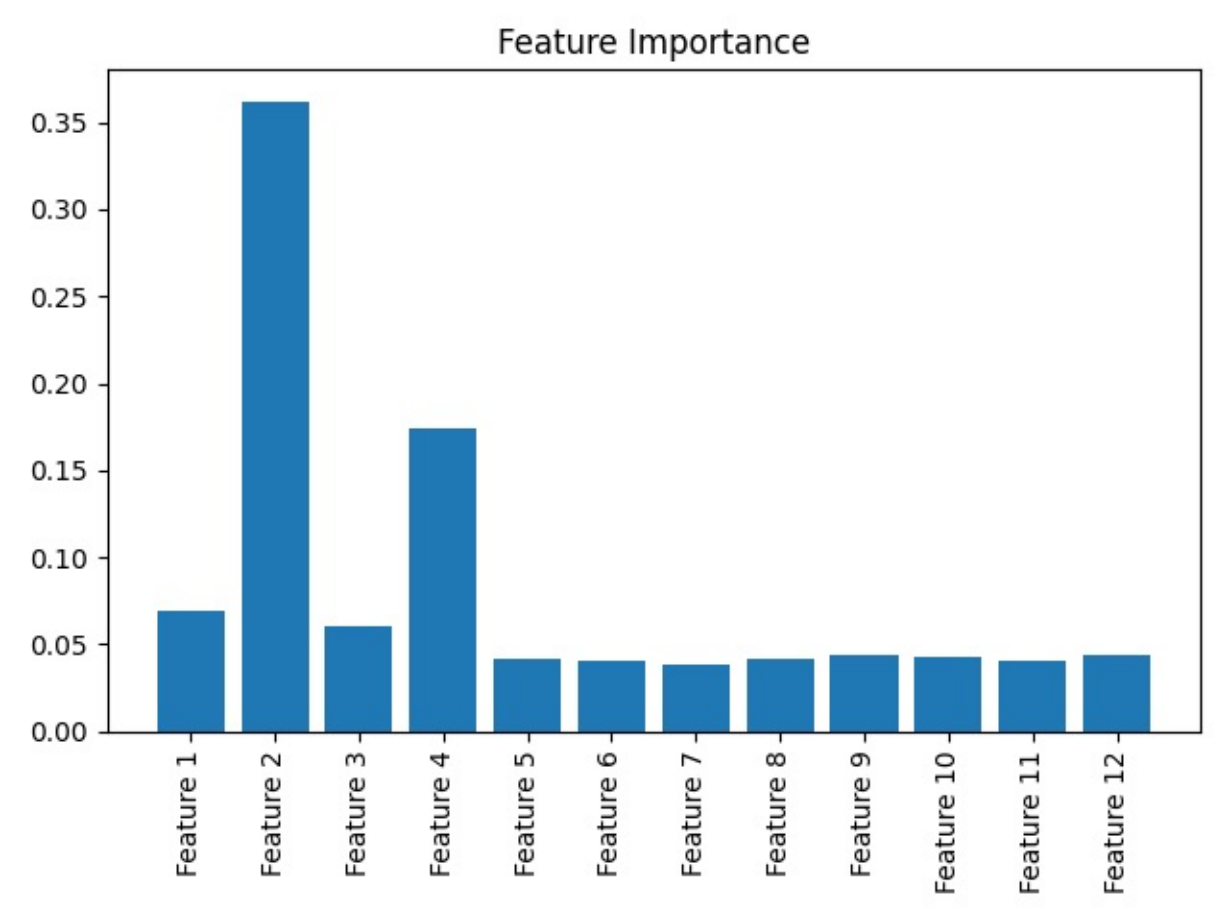
plt.bar(range(X_train.shape[1]), feature_importances, align='center')
plt.show()

{ Code & output Attached }

(D) y_test_predicted = RFClassifier.predict(X_test)
accuracy = RFClassifier.score(X_test, y_test)

{ Code & output attached }

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import matplotlib.pyplot as plt

# Do not change anything in this code
X, y = make_classification(
    n_samples=1000,
    n_features=12,
    n_informative=4,
    n_redundant=0,
    n_repeated=0,
    n_classes=2,
    random_state=5,
    shuffle=False)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=43)

RFClassifier = RandomForestClassifier()
RFClassifier.fit(X_train, y_train)
y_test_predicted = RFClassifier.predict(X_test)

feature_importances = RFClassifier.feature_importances_
print('Feature Importances: \n')
for i in range(0, 12):
    print('Feature '+str(i+1), feature_importances[i])

plt.title('Feature Importance')
plt.bar(range(X_train.shape[1]), feature_importances, align='center')
plt.xticks(range(X_train.shape[1]), ['Feature ' + str(i) for i in
range(1,X_train.shape[1]+1)], rotation=90)
plt.tight_layout()
plt.show()

accuracy = RFClassifier.score(X_test, y_test)
print('Accuracy using score function: ', accuracy)
accuracy2 = metrics.accuracy_score(y_test, y_test_predicted)
print('Accuracy using accuracy_score function: ', accuracy2)
```


Feature Importance
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Feature 1
Feature 2
Feature 3
Feature 4
Feature 5
Feature 6
Feature 7
Feature 8
Feature 9
Feature 10
Feature 11
Feature 12

q2.py 1, M | q1.py M

q1.py > ...

```python
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
import matplotlib.pyplot as plt

# Do not change anything in this code
X, y = make_classification(
    n_samples=1000,
    n_features=12,
    n_informative=4,
    n_redundant=0,
    n_repeated=0,
    n_classes=2,
    random_state=5,
    shuffle=False)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=43)

RFClassifier = RandomForestClassifier()
RFClassifier.fit(X_train, y_train)
y_test_predicted = RFClassifier.predict(X_test)

feature_importances = RFClassifier.feature_importances_
print('Feature Importances: \n')
for i in range(0, 12):
    print('Feature '+str(i+1), feature_importances[i])

plt.title('Feature Importance')
plt.bar(range(X_train.shape[1]), feature_importances, align='center')
plt.xticks(range(X_train.shape[1]), ['Feature ' + str(i) for i in range(1,X_train.shape[1]+1)],
plt.tight_layout()
plt.show()

accuracy = RFClassifier.score(X_test, y_test)
print('Accuracy using score function: ', accuracy)
accuracy2 = metrics.accuracy_score(y_test, y_test_predicted)
print('Accuracy using accuracy_score function: ', accuracy2)
```

PROBLEMS 3 | OUTPUT | TERMINAL

DEBUG CONSOLE

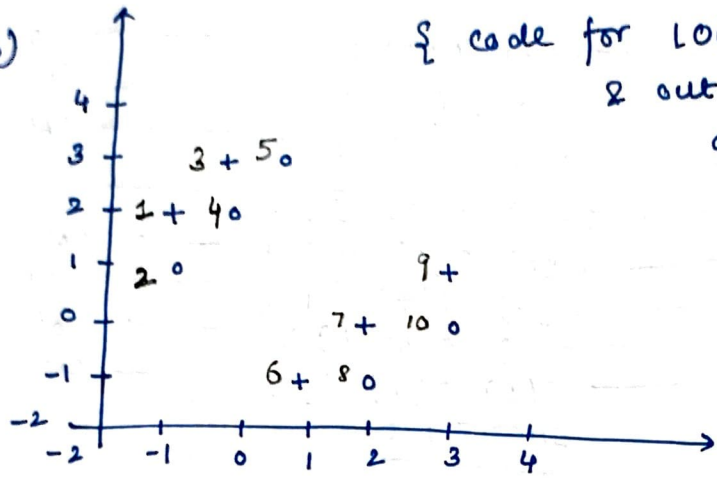
TERMINAL — ZSH

```
→  Assignment3 git:(main) ✗ python3 q1.py
Feature Importances:

Feature 1 0.06978644460046514
Feature 2 0.36209875366977057
Feature 3 0.060738332739039454
Feature 4 0.17362540851119743
Feature 5 0.04138894579182502
Feature 6 0.040114503263485354
Feature 7 0.038722709628786205
Feature 8 0.04186723981214087
Feature 9 0.04436293747237699
Feature 10 0.042533356734031795
Feature 11 0.04040247839726527
Feature 12 0.04435888937961594
Accuracy using score function:  0.844
Accuracy using accuracy_score function:  0.844
→  Assignment3 git:(main) ✗
```

ANSWER 2

(A) {code & output Attached}

(B)



{ code for LOOCV Error & output also attached }

K = 1 / 1NN

point 1 : nearest neighbors point 2 and 4 both 'o'
1 → 'o', actual '+'

point 2 : nearest neighbor point 1 '+'
2 → '+', actual 'o'

point 3 : nearest is point 4/5, both 'o'
3 → 'o', actual '+'

point 4 : nearest points 1/2 both '+'
4 → '+', actual 'o'

point 5 : nearest point 3 '+'
5 → '+', actual 'o'

point 6 : nearest point 8 'o'
6 → 'o', actual '+'

point 7 : nearest points 8/10 both 'o'
7 → 'o', actual '+'

point 8 : nearest points 6/7 both '+'
8 → '+', actual 'o'

point 9 : nearest neighbor 10 'o'
9 → 'o', actual '+'

point 10 : nearest neighbor 7/9 both '+'
10 → '+', actual 'o'

→ 1NN predicts all points wrong i.e. **Error = 1.0**

K = 3 | 3NN

point	Nearest Points (class)	Predicted class
1	2(o), 3(+), 4(o)	'o'
2	1(+) 3(+) 4(o)	'+'
3	1(+) 4(o) 5(o)	'o'
4	1(+) 3(+) 5(o)	'+'
5	1(+) 3(+) 4(o)	'+'
6	7(+) 8(o) 10(o)	'o'
7	6(+) 8(o) 10(o)	'o'
8	6(+) 7(+) 9(o)	'+'
9	7(+) 8(o) 10(o)	'o'
10	9(+) 7(+) 8(o)	'+'

Error = 1.0

K = 5 | 5NN

point	Nearet Points (class)	Predicted class
1	2(o) 3(+) 4(o) 5(o) 6(+)	'o'
2	1(+) 3(+) 4(o) 5(o) 6(+)	'+'
3	1(+) 2(o) 4(o) 5(o) 6(+)	'o'
4	2(o) 1(+) 3(+) 5(o) 6(+)	'+'
5		'+'
6	. . . .	'o'
7	(computed	'o'
8	using code	'+'
9	for higher	'o'
10	values)	'+'

Error = 1.0

K = 7 | 7NN

point	Nearest Points (class)	Predicted class
1		'+' ✓
2		'+' ✗
3	(Computed	'+' ✓
4	using	'+' ✗
5	code)	'+' ✗
6		'o' ✗
7		'o' ✗
8		'o' ✓
9		'o' ✗
10		'o' ✓

$$\text{Error} = \frac{6}{10} = 0.6$$

K = 9 / 9NN

As total points are 10,
9NN simply predicts the majority class of the remaining 9 points (10 minus the point itself)

point	predicted class
1	o
2	+
3	o
4	+
5	+
6	o
7	o
8	+
9	o
10	+

All predicted wrong

Error = 1.0

∴ K = 7 leads to the minimum LOOCV validation error.

K = 1, 3, 5, 9 Error = 1.0

K = 7 Error = 0.6

```
import numpy as np

# preparing input
x_coords = [-1, -1, 0, 0, 1, 1, 2, 2, 3, 3]
y_coords = [2, 1, 3, 2, 3, -1, 0, -1, 1, 0]
points = [list(x) for x in zip(x_coords, y_coords)]
classification_input = ['+', 'o', '+', 'o', 'o', '+', '+', 'o', '+', 'o']

def euclidean_dist(x1, y1, x2, y2):
    # returns euclidean distance between (x1, y1) and (x2, y2)
    return np.sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));

def nearest_neighbors(x_coords, y_coords, classification, x, y):
    # returns a list of points sorted in ascending order by distance from (x,y)
    distances = []
    for i in range(0, len(x_coords)):
            distances.append((euclidean_dist(x, y, x_coords[i], y_coords[i]),
x_coords[i], y_coords[i], classification[i]))
            # append a tuple to the list of form (euclidean_dist(x, y, xi, yi), xi,
yi, class of (xi, yi))
    distances.sort()
    # sort the list by euclidean_dist i.e. first element of the tuple (default)
    return distances

def get_knn(x_coords, y_coords, classification, x, y, k):
    # returns 'k' nearest neighbors of (x,y)
    return nearest_neighbors(x_coords, y_coords, classification, x, y)[0: k];

def knn_prediction(x_coords, y_coords, classification, x, y, k):
    # checks class of k nearest neighbours and returns the majority class as
prediction for that point
    knn = get_knn(x_coords, y_coords, classification, x, y, k)
    # print(x, y, knn)
    PLUSes = 0
    Os = 0
    for j in range(0, k):
        if(knn[j][3]=='o'):
            Os += 1;
        else:
            PLUSes += 1
    if(Os>PLUSes):
        return 'o'
    elif(PLUSes>Os):
        return '+'
    else:
        return '='

def knn(x_coords, y_coords, classification, k):
```

```
    # returns array of predicted values for all the points in the input set
    prediction = []
    for i in range(0, len(x_coords)):
        prediction.append(knn_prediction(x_coords, y_coords, classification,
x_coords[i], y_coords[i], k))
    return prediction

def getLOOCVError(x_coords, y_coords, classification, k):
    wrong_prediction = 0
    for i in range(0, len(x_coords)):
        new_x_coords = x_coords[:i]+x_coords[i+1:]
        new_y_coords = y_coords[:i]+y_coords[i+1:]
        new_classification = classification[:i]+classification[i+1:]
        element_prediction = knn_prediction(new_x_coords, new_y_coords,
new_classification, x_coords[i], y_coords[i], k)
        # print(x_coords[i], y_coords[i], element_prediction, classification[i])
        if(element_prediction != classification[i]):
            wrong_prediction += 1
    return wrong_prediction/len(x_coords)


# Running classification on the training set for an example output]
print('Actual Classes:', classification_input)
predictionk1 = knn(x_coords, y_coords, classification_input, 1)
print('Predicted Classes:')
print('k=1 ', predictionk1)
predictionk3 = knn(x_coords, y_coords, classification_input, 3)
print('k=3 ', predictionk3)
predictionk5 = knn(x_coords, y_coords, classification_input, 5)
print('k=5 ', predictionk5)
predictionk7 = knn(x_coords, y_coords, classification_input, 7)
print('k=7 ', predictionk7)
predictionk9 = knn(x_coords, y_coords, classification_input, 3)
print('k=9 ', predictionk9)

print('\nLOOCV Errors: ')
print('k=1 ', getLOOCVError(x_coords, y_coords, classification_input, 1))
print('k=3 ', getLOOCVError(x_coords, y_coords, classification_input, 3))
print('k=5 ', getLOOCVError(x_coords, y_coords, classification_input, 5))
print('k=7 ', getLOOCVError(x_coords, y_coords, classification_input, 7))
print('k=9 ', getLOOCVError(x_coords, y_coords, classification_input, 9))
```

q2.py 1, M | q1.py M

q2.py > ...

```python
import numpy as np

# preparing input
x_coords = [-1, -1, 0, 0, 1, 1, 2, 2, 3, 3]
y_coords = [2, 1, 3, 2, 3, -1, 0, -1, 1, 0]
points = [list(x) for x in zip(x_coords, y_coords)]
classification_input = ['+', 'o', '+', 'o', 'o', '+', '+', 'o', '+', 'o']

def euclidean_dist(x1, y1, x2, y2):
    # returns euclidean distance between (x1, y1) and (x2, y2)
    return np.sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));

def nearest_neighbors(x_coords, y_coords, classification, x, y):
    # returns a list of points sorted in ascending order by distance from (x,y)
    distances = []
    for i in range(0, len(x_coords)):
            distances.append((euclidean_dist(x, y, x_coords[i], y_coords[i]), x_coords
            # append a tuple to the list of form (euclidean_dist(x, y, xi, yi), xi, yi
    distances.sort()
    # sort the list by euclidean_dist i.e. first element of the tuple (default)
    return distances

def get_knn(x_coords, y_coords, classification, x, y, k):
    # returns 'k' nearest neighbors of (x,y)
    return nearest_neighbors(x_coords, y_coords, classification, x, y)[0: k];

def knn_prediction(x_coords, y_coords, classification, x, y, k):
    # checks class of k nearest neighbours and returns the majority class as predictic
    knn = get_knn(x_coords, y_coords, classification, x, y, k)
    # print(x, y, knn)
    PLUSes = 0
    Os = 0
    for j in range(0, k):
        if(knn[j][3]=='o'):
            Os += 1;
        else:
            PLUSes += 1
    if(Os>PLUSes):
        return 'o'
    elif(PLUSes>Os):
        return '+'
    else:
        return '='
```

PROBLEMS 3 | OUTPUT | TERMINAL

DEBUG CONSOLE

TERMINAL — ZSH

```
➜  Assignment3 git:(main) ✗ python3 q2.py
Actual Classes: ['+', 'o', '+', 'o', 'o', '+', '+', 'o', '+', 'o']
Predicted Classes:
k=1  ['+', 'o', '+', 'o', 'o', '+', '+', 'o', '+', 'o']
k=3  ['o', 'o', 'o', '+', 'o', '+', 'o', '+', '+', '+']
k=5  ['o', '+', 'o', 'o', 'o', 'o', '+', '+', '+', '+']
k=7  ['+', '+', '+', '+', '+', 'o', 'o', 'o', 'o', 'o']
k=9  ['o', 'o', 'o', '+', 'o', '+', 'o', '+', '+', '+']

LOOCV Errors:
k=1  1.0
k=3  1.0
k=5  1.0
k=7  0.6
k=9  1.0
➜  Assignment3 git:(main) ✗
```

```python
    else:
        return '='

def knn(x_coords, y_coords, classification, k):
    # returns array of predicted values for all the points in the input set
    prediction = []
    for i in range(0, len(x_coords)):
        prediction.append(knn_prediction(x_coords, y_coords, classification, x_coords[
    return prediction

def getLOOCVError(x_coords, y_coords, classification, k):
    wrong_prediction = 0
    for i in range(0, len(x_coords)):
        new_x_coords = x_coords[:i]+x_coords[i+1:]
        new_y_coords = y_coords[:i]+y_coords[i+1:]
        new_classification = classification[:i]+classification[i+1:]
        element_prediction = knn_prediction(new_x_coords, new_y_coords, new_classifica
        # print(x_coords[i], y_coords[i], element_prediction, classification[i])
        if(element_prediction != classification[i]):
            wrong_prediction += 1
    return wrong_prediction/len(x_coords)


# Running classification on the training set for an example output]
print('Actual Classes:', classification_input)
predictionk1 = knn(x_coords, y_coords, classification_input, 1)
print('Predicted Classes:')
print('k=1 ', predictionk1)
predictionk3 = knn(x_coords, y_coords, classification_input, 3)
print('k=3 ', predictionk3)
predictionk5 = knn(x_coords, y_coords, classification_input, 5)
print('k=5 ', predictionk5)
predictionk7 = knn(x_coords, y_coords, classification_input, 7)
print('k=7 ', predictionk7)
predictionk9 = knn(x_coords, y_coords, classification_input, 3)
print('k=9 ', predictionk9)

print('\nLOOCV Errors: ')
print('k=1 ', getLOOCVError(x_coords, y_coords, classification_input, 1))
print('k=3 ', getLOOCVError(x_coords, y_coords, classification_input, 3))
print('k=5 ', getLOOCVError(x_coords, y_coords, classification_input, 5))
print('k=7 ', getLOOCVError(x_coords, y_coords, classification_input, 7))
print('k=9 ', getLOOCVError(x_coords, y_coords, classification_input, 9))
```



```
➜  Assignment3 git:(main) ✗ python3 q2.py
Actual Classes: ['+', 'o', '+', 'o', 'o', '+', '+', 'o', '+', 'o']
Predicted Classes:
k=1  ['+', 'o', '+', 'o', 'o', '+', '+', 'o', '+', 'o']
k=3  ['o', 'o', 'o', '+', 'o', '+', 'o', '+', '+', '+']
k=5  ['o', '+', 'o', 'o', 'o', 'o', '+', '+', '+', '+']
k=7  ['+', '+', '+', '+', '+', 'o', 'o', 'o', 'o', 'o']
k=9  ['o', 'o', 'o', '+', 'o', '+', 'o', '+', '+', '+']

LOOCV Errors:
k=1  1.0
k=3  1.0
k=5  1.0
k=7  0.6
k=9  1.0
➜  Assignment3 git:(main) ✗
```

ANSWER 3

	A_1	A_2	A_3	output y
x_1	1	0	0	0
x_2	1	0	1	0
x_3	0	1	0	0
x_4	1	1	1	1
x_5	1	1	0	1

We want to choose the attribute to split in such a way that remaining entropy is minimized.

i.e. we compute Remainder(Ai) for each Ai at split and select one that gives least value.

$$\Rightarrow \text{Remainder}(A_i) = \sum_{k=1}^{2} \frac{p_k + n_k}{p+n} B\left(\frac{p_k}{p_k + n_k}\right)$$

k=1 corresponds to $A_i = 0$, k=2 to $A_i = 1$

where $B(q) = -(q \log_2 q + (1-q)\log_2(1-q))$

p_k is no. of positive egs and

n_k is no. of neg egs for k^{th}

first split

$$\text{Remainder}(A_1) = \frac{4}{5}\left(-\frac{2}{4}\log_2\left(\frac{2}{4}\right) - \frac{2}{4}\log_2\left(\frac{2}{4}\right)\right) + \frac{1}{5}\left(-0 - \frac{1}{1}\log\left(\frac{1}{1}\right)\right) =$$

$$\frac{4}{5}\left(\frac{1}{2} + \frac{1}{2}\right) + 0 = 0.8.$$

$$\text{Remainder}(A_2) = \frac{3}{5}\left(-\frac{2}{3}\log_2\left(\frac{2}{3}\right) - \frac{1}{3}\log_2\left(\frac{1}{3}\right)\right) + \frac{2}{5}\left(-0 - \frac{2}{2}\log_2\left(\frac{2}{2}\right)\right)$$

$$= \frac{3}{5}\frac{1}{3}\left(2\log_2\left(\frac{3}{2}\right) + \log_2(3)\right) = 0.55.$$

$$\text{Remainder}(A_3) = \frac{2}{5}\left(-\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right)$$

$$+ \frac{3}{5}\left(-\frac{1}{3}\log_2\left(\frac{1}{3}\right) - \frac{2}{3}\log_2\left(\frac{2}{3}\right)\right)$$

$$= \frac{1}{5}(1+1) + \frac{1}{5}\left(\log_2 3 + 2\log_2\frac{3}{2}\right)$$

$$= 0.9509 = 0.95$$

Hence, A_2 minimizes remaing entropy.

Choose A_2 to split.

All with $A_2 = 0$ are output $y = 0$.

so, we are left with $\{x_3, x_4, x_5\}$.

and A_1 and A_3.

$$\text{Remainder}(A_1) = \frac{2}{3}\left(-\frac{2}{2}\log_2\left(\frac{2}{2}\right) - 0\right)$$

$$+ \frac{1}{3}\left(-0 - \frac{1}{1}\log_2\left(\frac{1}{1}\right)\right) = 0$$

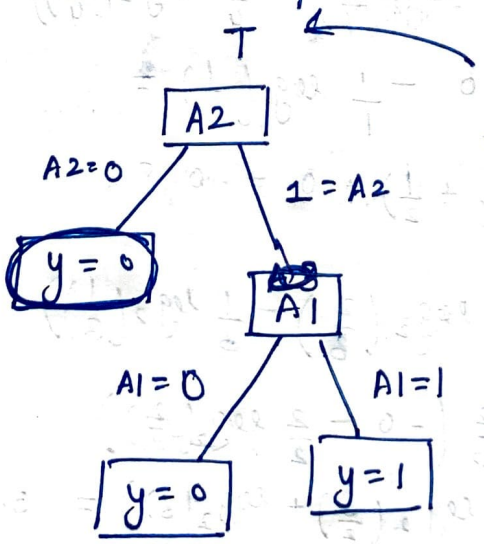
$$\text{Remainder}(A_3) = \frac{1}{3}\left(-\frac{1}{1}\log_2\left(\frac{1}{1}\right)\right) + \frac{2}{3}\left(-\frac{1}{2}\log_2\left(\frac{1}{2}\right)\right.$$

$$\left. - \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right)$$

$$= \frac{1}{3}(0) + \frac{2}{3}\left(\frac{1}{2} + \frac{1}{2}\right) = \frac{2}{3} = 0.67$$

Hence, A_1 minimizes remaining entropy to 0.

i.e. we split wrt A_3 now.



final decision tree

ANSWER 4

(a) fitted value at $x = 7$ is $\hat{y}(7)$

$\hat{y} = 18 - 0.4x$

$\hat{y}(7) = 18 - 0.4(7) = 18 - 2.8 = 15.2$

$\therefore$ fitted value at $x = 7$ is 15.2

(b) Residual at $x = 3$ and $y = 20$

is $20 - \hat{y}(3)$

$= 20 - 18 + 0.4(3)$

$= 2 + 0.4(3) = 3.2$