Amir Hossein Eyvazkhani

1747696

Ex 9

Task 1)

a) $N(\mu, \sigma^2) \Rightarrow p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

$$H(x) = -\int_{-\infty}^{\infty} p(x) \ln p(x)\, dx$$

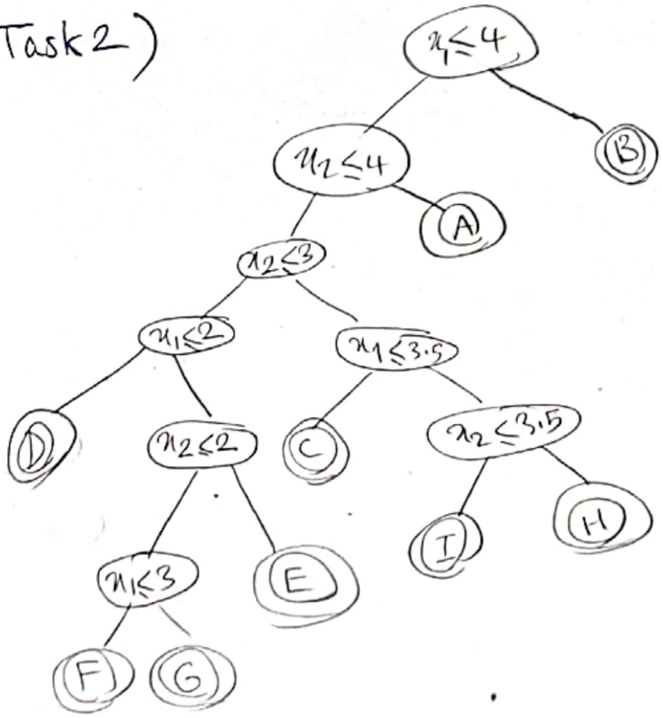
$$= \frac{1}{\sigma\sqrt{2\pi}}\left[\int \ln(\sigma\sqrt{2\pi})\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx + \int \frac{(x-\mu)^2}{2\sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx\right]$$

$$= \ln(\sigma\sqrt{2\pi}) \underbrace{\int \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx}_{=1} + \frac{1}{2\sigma^2} \underbrace{\int \frac{(x-\mu)^2}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx}_{=\sigma^2}$$

$$= \frac{1}{2} \ln(2\pi e \sigma^2)$$

b) $H(x) < 0 \Rightarrow 2\pi e \sigma^2 < 1 \Rightarrow 0 < \sigma < \frac{1}{\sqrt{2\pi e}}$

Task 2)

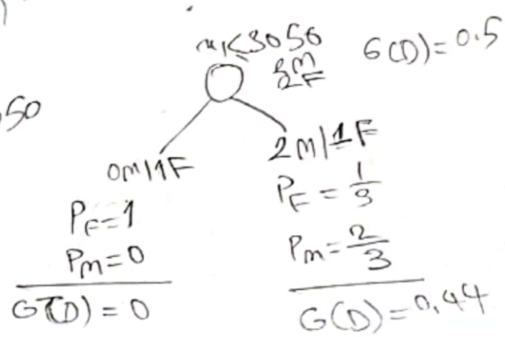


Task 3)

$N_{min} = 1$; $G(D) = 1 - \sum_{t=1}^{T} p_t^2$, $GI(D, D_1, D_2) = G(D) - \sum_{r=1}^{2} \frac{|D_r|}{|D|} G(D_r)$

(root) $x_1 \in \{2700, 3400, 3500, 5500\} \Rightarrow$ splits $\in \{3050, 3450, 4500\}$

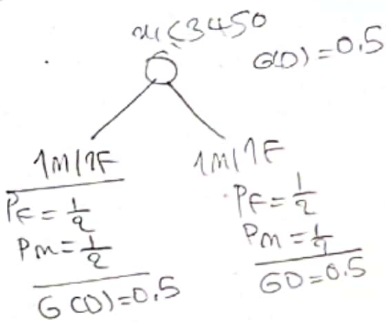
split for $x_1 \leq 3050$



$GI = 0.5 - \left(\frac{1}{4}(0) + \frac{3}{4}(0.44)\right)$

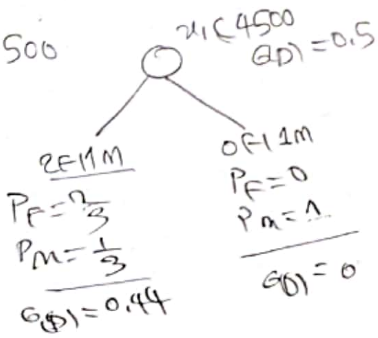
$= 0.17$

split for $x_1 \leq 3450$



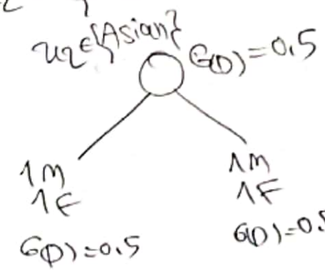
$GI = 0.5 - \left(\frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{2}\right) = 0$

split for $x_1 \leq 4500$



$GI = 0.5 - \left(\frac{1}{4}(0) + \frac{3}{4}(0.44)\right) = 0.17$

$x_2 \in \{\text{Asian}, \text{African}\}$, splits $\in \{\{\text{African}\} \text{ vs } \{\text{Asian}\}\}$



$GI = 0$

Hence here the best GI is for $x_1 \leq 3050$ or $x_1 \leq 4500$

we choose the $x_1 \leq 3050$ for our first split.

$x_1 \leq 3$[illegible]

0M/1F — 2m|1F

F

hence we only have one class here (F), we note this leaf F.

$x_1 \in \{3400, 3500, 5500\} \Rightarrow splits \in \{3450, 4500\}$

for the split $x_1 \leq 3450$

2m
1F
$G(D) = 0.44$
$x_1 \leq 3450$

1M, 0F	1M, 1F
$P_F = 0$	$P_F = 0.5$
$P_M = 1$	$P_M = 0.5$
$G(D) = 0$	$G(D) = 0.5$

$GI = 0.44 - \left(\frac{1}{3} \times 0 + \frac{2}{3} \times \frac{1}{2}\right)$

$= 0.11$

for the split $x_1 \leq 4500$

2M
1F
$G(D) = 0.44$
$x_1 \leq 4500$

1M, 1F	1m, 0F
$G(D) = 0.5$	$G(D) = 0$

$GI = 0.11$

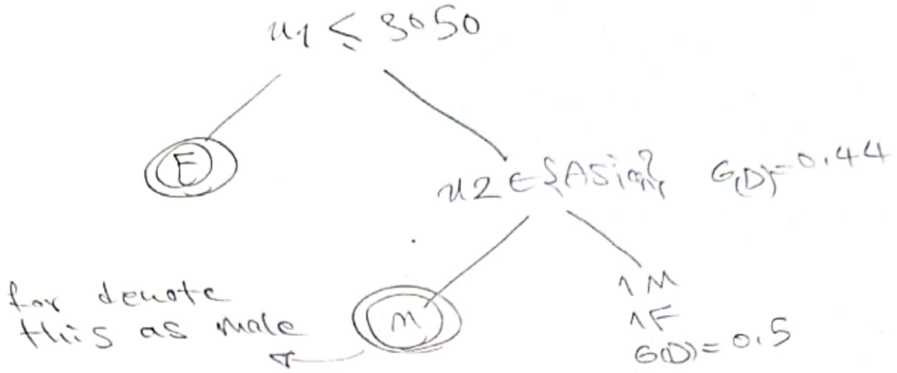
for the split $x_2 \in \{Asian\}$

2M
1F
$G(D) = 0.44$
$x_2 \in \{Asian\}$

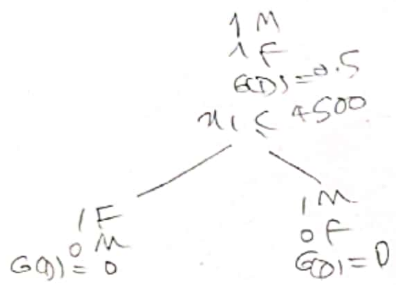
1M, 0F	1M, 1F
$G(D) = 0$	$G(D) = 0.5$

$GI = 0.11$

we here have the same GI with all splits; So we select $x_2 \in \{Asian\}$ this time.

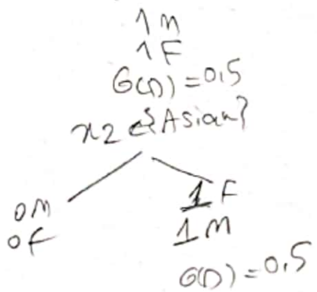


$x_1 \in \{3500, 5500\} \Rightarrow split \in \{4500\}$



$GI = 0.5$

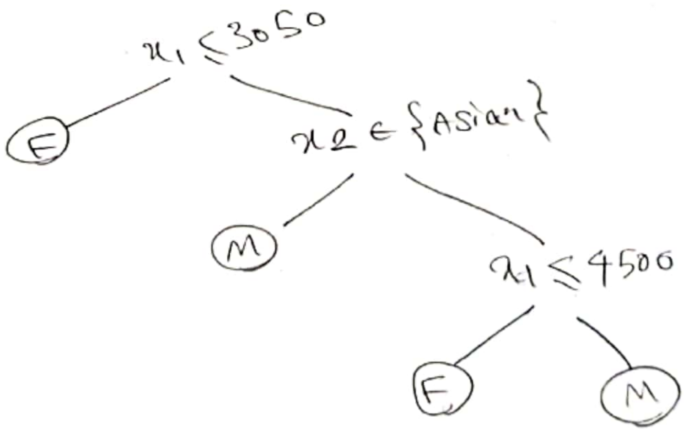
$x_2 \in \{African\}$



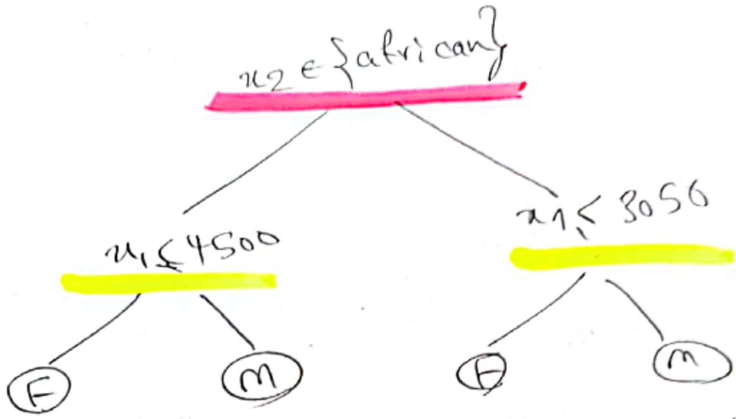
$GI = 0.5 - \left(0 + \frac{1}{2}\right) = 0$

Hence we select the better GI (0.5) and the split $x_1 \leq 4500$ is selected at the end. we stop at the leaves cause we only have a single lable.

The final tree:



The minimal Depth would be for this tree:



Because we used the Gini-index as the criteria, and the fact that the algorithm is greedy, which means it selects the best split (aiming for pure nodes) at each Step, it would miss the highlighted Pink node in the tree above that is actually not pure and the best choice when spliting from the start.

Task 4

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
from pprint import pprint
```

In [ ]:
```
%matplotlib inline
```

In [ ]:
```
def data_split(df, train_size):
    df = df.sample(frac=1)
    train_df = df.head(train_size)
    test_df = df.tail(len(df) - train_size)
    return train_df, test_df
```

In [ ]:
```
df = pd.read_csv("data_banknote_authentication.txt", names=['variance','skewness','kurtosis','entropy','class'])
# Make sure you always put the last column name as "label" as it is used by the algorithm
df = df.rename(columns={"class": "label"})
```

In [ ]:
```
train_df, test_df = data_split(df, 1000)
```

In [ ]:
```
train_df[:5]
```

Out[ ]:

	variance	skewness	kurtosis	entropy	label
1180	-2.21830	-1.2540	2.99860	0.36378	1
231	-2.33610	11.9604	3.08350	-5.44350	0
314	1.18110	8.3847	-2.05670	-0.90345	0
4	0.32924	-4.4552	4.57180	-0.98880	0
1021	-1.27920	2.1376	-0.47584	-1.39740	1

In [ ]:
```
def check_if_unique_class(data):
    # This function returns True if data contains only one unique class
```

```python
    label_column = data[:, -1]
    unique_classes = np.unique(label_column)

    if len(unique_classes) == 1:
        return True
    else:
        return False
```

In [ ]:
```python
def unique_classes(data):
    # This function does simply return unique classes in the data
    label_column = data[:, -1]
    unique_classes, counts_unique_classes = np.unique(label_column, return_counts=True)

    index = counts_unique_classes.argmax()
    classification = unique_classes[index]

    return classification
```

In [ ]:
```python
def get_potential_splits(data):
    # Get all potential splits for all the columns of the data
    potential_splits = {}
    _, n_columns = data.shape
    for column_index in range(n_columns - 1):        # excluding the last column which is the label
        potential_splits[column_index] = []
        values = data[:, column_index]
        unique_values = np.unique(values)

        for index in range(len(unique_values)):
            if index != 0:
                current_value = unique_values[index]
                previous_value = unique_values[index - 1]
                potential_split = (current_value + previous_value) / 2

                potential_splits[column_index].append(potential_split)

    return potential_splits
```

In [ ]:
```python
def split_data(data, split_column, split_value):
    # This function is useful in the splitting of a node given a feature attribute and the split value
    split_column_values = data[:, split_column]
```

```python
    data_below = data[split_column_values <= split_value]
    data_above = data[split_column_values >  split_value]

    return data_below, data_above
```

In [ ]:
```python
def compute_entropy(data):
    # This function computes the entropy of the label column
    label_column = data[:, -1]
    _, counts = np.unique(label_column, return_counts=True)

    probabilities = counts / counts.sum()
    entropy = sum(probabilities * -np.log2(probabilities))

    return entropy
```

In [ ]:
```python
def calculate_split_entropy(data_below, data_above):
    # This function computes the split entropy given the data going into the left and the right nodes
    n = len(data_below) + len(data_above)
    p_data_below = len(data_below) / n
    p_data_above = len(data_above) / n

    overall_entropy =  (p_data_below * compute_entropy(data_below)
                      + p_data_above * compute_entropy(data_above))

    return overall_entropy
```

In [ ]:
```python
def determine_best_split(data, potential_splits):
    IG=float('inf')
    for column,splits in potential_splits.items():
        for threshold in splits:
            leftData,rightData=split_data(data,column,threshold)
            currIG=calculate_split_entropy(leftData,rightData)
            if currIG<=IG:
                bestCol=column
                bestVal=threshold
                IG=currIG
    return bestCol,bestVal
```

In [ ]:
```python
def dtree(df, counter=0):
    # This function implements a basic tree algorithm. It returns the learnt tree as a dictionary.
    # The keys of the dictionary are of the form X<=Y, where X is the index of the column used for splitting and Y is
```

```
    # data preparations
    if counter == 0:
        data = df.values
        global COLUMN_HEADERS, FEATURE_TYPES
        COLUMN_HEADERS = df.columns
    else:
        data = df


    # base cases
    if check_if_unique_class(data):
        classification = unique_classes(data)
        return classification


    # recursive part
    else:
        counter += 1

        # helper functions
        potential_splits = get_potential_splits(data)
        split_column, split_value = determine_best_split(data, potential_splits)
        data_below, data_above = split_data(data, split_column, split_value)

        # instantiate sub-tree
        feature_name = COLUMN_HEADERS[split_column]
        question = "{} <= {}".format(feature_name, split_value)
        sub_tree = {question: []}

        # find answers (recursion)
        yes_answer = dtree(data_below, counter)
        no_answer = dtree(data_above, counter)

        sub_tree[question].append(yes_answer)
        sub_tree[question].append(no_answer)

        return sub_tree
```

In [ ]:

```
treeBank = dtree(train_df)
```

In [ ]:

```
pprint(treeBank)
```

```
{'variance <= 0.7602949999999999': [{'skewness <= 6.8193': [{'variance <= -0.462635': [{'kurtosis <= 6.21865': [1.0,
                                                                                                                                   {'ske
wness <= -4.72105': [1.0,

0.0]}]},
                                                                                                                  {'kurtosis <= 0.34073': [{'ske
wness <= 5.73915': [1.0,

0.0]},
                                                                                                                                   {'ent
ropy <= 0.72843': [{'kurtosis <= 0.972205': [{'entropy <= -0.073935': [0.0,

1.0]},

0.0]},

{'kurtosis <= 6.2508': [1.0,

0.0]}]}]}]},
                                                                    {'variance <= -4.3819': [1.0,
                                                                                             0.0]}]},
                                        {'kurtosis <= -1.8648': [{'skewness <= 5.07075': [{'variance <= 3.4798': [1.0,
                                                                                                                   0.0]},
                                                                                              0.0]},
                                                                 0.0]}]}
```

In [ ]:

```python
def predict_example(example, tree):
    # This function makes predictions for a single row of a pandas dataframe
    # tree is just a root node
    if not isinstance(tree, dict):
        return tree

    question = list(tree.keys())[0]
    feature_name, comparison_operator, value = question.split(" ")

    # ask question
    if comparison_operator == "<=":
        if example[feature_name] <= float(value):
            answer = tree[question][0]
        else:
            answer = tree[question][1]
```

```python
    # feature is categorical
    else:
        if str(example[feature_name]) == value:
            answer = tree[question][0]
        else:
            answer = tree[question][1]

    # base case
    if not isinstance(answer, dict):
        return answer

    # recursive part
    else:
        residual_tree = answer
        return predict_example(example, residual_tree)
```

In [ ]:
```python
def make_predictions(df, tree):
    # This uses pandas apply function to make predictions for the complete dataframe
    if len(df) != 0:
        predictions = df.apply(predict_example, args=(tree,), axis=1)
    else:
        # "df.apply()"" with empty dataframe returns an empty dataframe,
        # but "predictions" should be a series instead
        predictions = pd.Series()

    return predictions
```

In [ ]:
```python
from sklearn import tree
model=tree.DecisionTreeClassifier()
predict_bank=make_predictions(test_df, treeBank)
ytest=test_df['label'].values
bank_accuracy=(ytest==predict_bank).mean()
print('Accuracy obtained on the bank dataset using given algorithm is: %0.2f'%bank_accuracy)
```

```
Accuracy obtained on the bank dataset using given algorithm is: 0.99
```

In [ ]:
```python
Xtrain=train_df.values[:,:-1]
Ytrain=train_df.values[:,-1]
Xtest=test_df.values[:,:-1]
Ytest=test_df.values[:,-1]
bank=model.fit(Xtrain,Ytrain)
predSK=model.predict(Xtest)
```

```
SKaccur=(predSK==Ytest).mean()
print('Accuracy obtained on the bank dataset using scitkit learn is: %0.2f'%SKaccur)
```

```
Accuracy obtained on the bank dataset using scitkit learn is: 0.99
```

Iris Dataset

In [ ]:
```
col_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'type']
iris = pd.read_csv("iris.csv", skiprows=1, header=None, names=col_names)
iris= iris.rename(columns={"type": "label"})
iris.head()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	label
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa

In [ ]:
```
train, test = data_split(iris, 105)
treeIris = dtree(train)
pprint(treeIris)
```

```
{'petal_width <= 0.8': ['Iris-setosa',
                        {'petal_width <= 1.75': [{'petal_length <= 5.05': ['Iris-versicolor',
                                                                           {'sepal_width <= 2.75': ['Iris-versicolo
r',
                                                                                                    'Iris-virginic
a']}]},
                                                 {'petal_length <= 4.85': [{'sepal_width <= 3.0': ['Iris-virginica',
                                                                                                   'Iris-versicolo
r']},
                                                                           'Iris-virginica']}]}]}
```

In [ ]:
```
predict_iris=make_predictions(test,treeIris)
ytestI=test['label'].values
```

```
iris_accuracy=(ytestI==predict_iris).mean()
print('Accuracy obtained on the iris dataset using given algorithm is: %0.2f'%iris_accuracy)
```

```
Accuracy obtained on the iris dataset using given algorithm is: 0.93
```

In [ ]:

```
model=tree.DecisionTreeClassifier()
XtrainI=train.values[:,:-1]
YtrainI=train.values[:,-1]
XtestI=test.values[:,:-1]
YtestI=test.values[:,-1]
iris=model.fit(XtrainI,YtrainI)
predISK=model.predict(XtestI)
SKaccurI=(predISK==YtestI).mean()
print('Accuracy obtained on the iris dataset using scitkit learn is: %0.2f'%SKaccurI)
```

```
Accuracy obtained on the iris dataset using scitkit learn is: 0.91
```

By default scikit learn uses the Gini Index as quality criteria, but here we are using information gain. For the bank dataset, the results were the same, but for the iris dataset, entrophy (which we used here), seems like a better option; cause it gave more accuracy.