Part I: Pen and paper

1. **Draw the training confusion matrix.**

   After consulting the decision tree described in the question's statement, we can assert that its confusion matrix is the following:
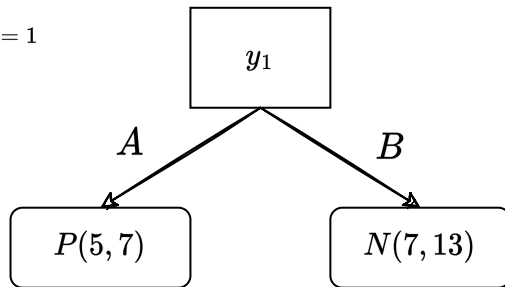
   ```
                      Real

                   P      N

               P   8      4
   Projected

               N   3      5
   ```

   Figure 1: Confusion Matrix

2. **Identify the training F1 after a post-pruning of the given tree under a maximum depth of 1.**

   Figure 2: Post-pruning confusion matrix

   We know that:

   $$d = 1: \quad F_1 = \frac{2 * \text{Precision} * \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}}$$

   Therefore, we must calculate the associated precision and sensitivity as follows:

   $$\text{Sensitivity} = \frac{TP}{TP + FN} = \frac{5}{5 + 6} = \frac{5}{11}$$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{5}{5 + 2} = \frac{5}{7}$$

We can, now, calculate the F1 score:

$$d = 1 : \quad F_1 = \frac{2 * \frac{5}{7} * \frac{5}{11}}{\frac{5}{7} + \frac{5}{11}} = \frac{5}{9}$$

3. **Identify two different reasons as to why the left tree path was not further decomposed.**

   We can find multiple reasons for the left-most tree path not being further decomposed.

   The first, perhaps most obvious reason is that we want to **avoid overfitting** the tree to the training data: given a threshold of $\frac{5}{7}$, it's already possible to make a very fair, general assessment about all instances that fall under the $y_1 = A$ condition (asserting they fall onto the P label). It's obviously not guaranteed that not further decomposing the tree will lead to a better labeling come testing time, but generally speaking it ends up being a good practice to avoid overfitting.

   Related to the overfitting idea, there also aren't many instances (only 7) in the left path, which makes it so that further decomposing that path doesn't change the classification error by much (if at all), while also adding unnecessary complexity to the tree - therefore, it's **not likely** that there will be a significant **entropy reduction** by further decomposing the left-most tree path.

4. **Compute the information gain of variable y1.**

   We know that:

   $$\text{IG}(y_{out} \mid y_1) = \text{H}(y_{out}) - \text{H}(y_{out} \mid y_1) \tag{1}$$

   $$\text{H}(Y) = - \sum_{y \in Y} p(y) \log_2 p(y) \tag{2}$$

   Considering y_{out} as our *output variable*, with possible values P or N, we'll be able to see that:

   $$\text{H}(y_{out}) = \left(-\frac{11}{20} \log_2 \frac{11}{20} - \frac{9}{20} \log_2 \frac{9}{20}\right) \tag{3}$$

$$H(y_{out} \mid y_1) = \left[\frac{7}{20} \left(-\frac{5}{7} \log_2 \frac{5}{7} - \frac{2}{7} \log_2 \frac{2}{7} \right) + \frac{13}{20} \left(-\frac{6}{13} \log_2 \frac{6}{13} - \frac{7}{13} \log_2 \frac{7}{13} \right) \right] \quad (4)$$

We have, therefore:

$$IG(y_{out} \mid y_1) = 0.04345941113$$

Part II: Programming

5. **Using `sklearn`, apply a stratified 70-30 training-testing split with a fixed seed (`random_state=1`), and assess in a single plot the training and testing accuracies of a decision tree with no depth limits (and remaining default behavior) for a varying number of selected features in {5,10,40,100,250,700}. Feature selection should be performed before decision tree learning considering the discriminative power of the input variables according to mutual information criterion (`mutual_info_classif`).**
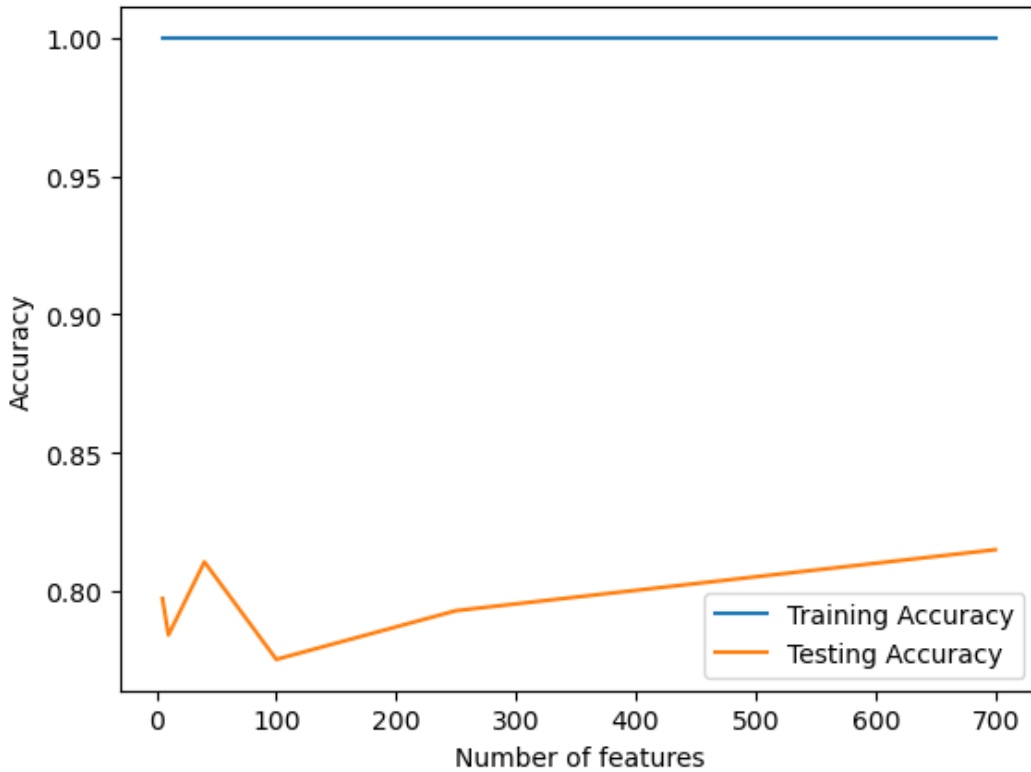

Figure 3: Training and Testing Accuracy vs. Number of Features

The code utilized to generate the plot is present in the report's appendix.

6. **Why is training accuracy persistently 1? Critically analyze the gathered results.**

Since the decision tree has no depth-limit associated, the worst-case scenario sees the tree holding exactly one leaf per training sample - leading to there always being a "correct path" in the tree for a given observation - and thus the training accuracy is always 1, for any train-test split of the dataset.

The testing accuracy isn't necessarily 1, of course, just like it can be seen in the plot shown above: in the vast majority of cases, the testing set will hold samples which differ from all seen in the training set, effectively evicting the "guarantee" of a correct guess from the decision tree. It can, of course, still take a correct guess (with that being our goal), but it's not guaranteed.

Appendix

Code utilized in the first question of the programming section shown in the pages below:

```
1  import pandas as pd
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import seaborn as sns
5  from scipy.io.arff import loadarff
6  from sklearn.model_selection import train_test_split
7  from sklearn.feature_selection import mutual_info_classif
8  from sklearn.tree import DecisionTreeClassifier
9  from sklearn.metrics import accuracy_score
10
11 FEATURE_AMOUNT = [5, 10, 40, 100, 250, 700]
12 SEED = 1
13
14 # Loading data
15
16 df = loadarff('data/pd_speech.arff')
17 df = pd.DataFrame(df[0])
18 df['class'] = df['class'].str.decode('utf-8')
19
20 # The first step should be splitting the dataset into stratified training and
21 # testing sets (70-30 split) with a fixed seed (random_state=1).
22
23 train, test = train_test_split(df, test_size=0.3, random_state=SEED, stratify=df['
      class'])
24 X = train.drop('class', axis=1)
25 y = train['class']
26
27 # Afterward, we should try to check the discriminative power of each feature
      considering the mutual_info_classif criterion.
28 # If we can only use a subset of features, we should select the most discriminative
      ones,
29 # (i.e, if we can use 5 features, we should select the 5 most discriminative ones,
      and so on).
30
31 mimportance = mutual_info_classif(X, y, random_state=SEED)
```

```
32 SORTED_FEATURE_INDICES = np.argsort(mimportance)[::-1]
33
34 # Now, time to finally assess (in a single plot) both the training and testing
     accuracies
35 # of a decision tree with no depth limits (and remaining default behavior) for a
     varying number of selected features.
36
37 def train_and_test(X_train, X_test, y_train, y_test):
38   clf = DecisionTreeClassifier(random_state=SEED)
39   clf.fit(X_train, y_train)
40   y_pred = clf.predict(X_test)
41   return accuracy_score(y_test, y_pred)
42
43 train_accuracies = []
44 test_accuracies = []
45
46 for n in FEATURE_AMOUNT:
47     chosen_features = X.columns[SORTED_FEATURE_INDICES[:n]]
48     X_train = train[chosen_features]
49     X_test = test[chosen_features]
50     train_accuracies.append(train_and_test(X_train, X_train, y, y))
51     test_accuracies.append(train_and_test(X_train, X_test, y, test['class']))
52
53 plt.plot(FEATURE_AMOUNT, train_accuracies, label='Training Accuracy')
54 plt.plot(FEATURE_AMOUNT, test_accuracies, label='Testing Accuracy')
55 plt.xlabel('Number of features')
56 plt.ylabel('Accuracy')
57 plt.legend()
58 plt.show()
```