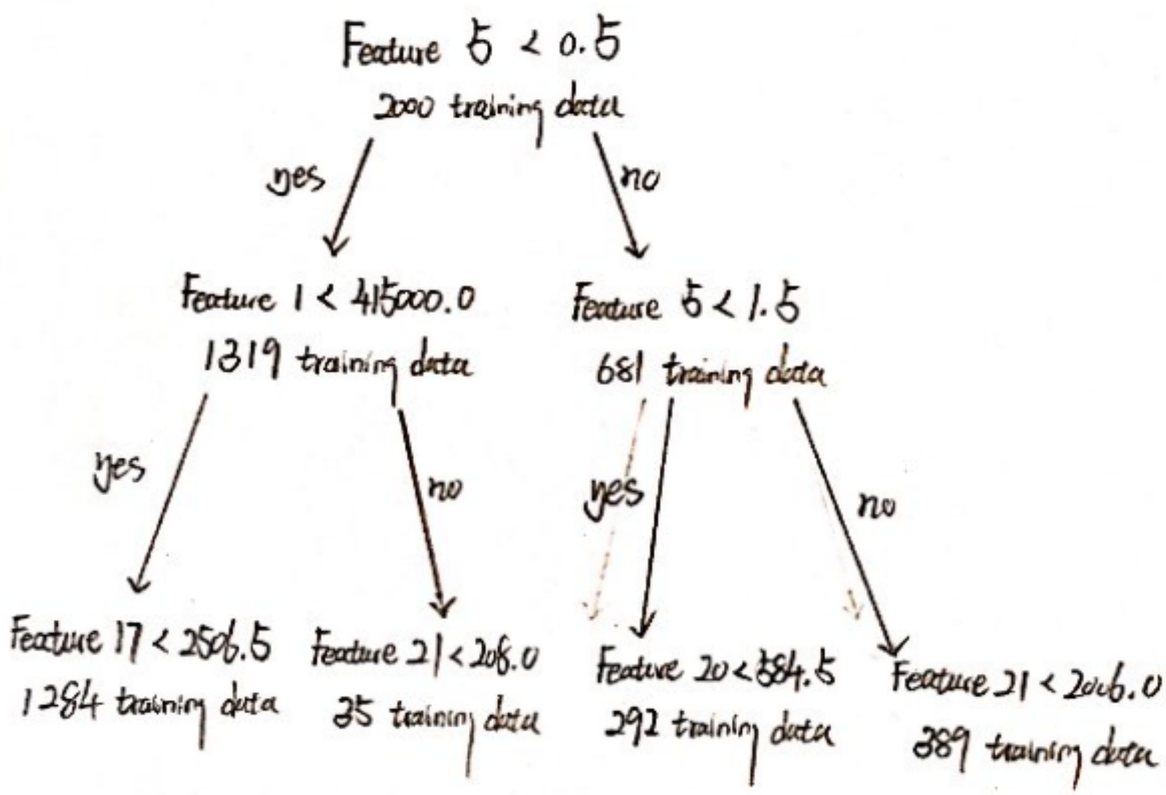
1. First, build an ID3 Decision Tree classifier based on the data in pa2train.txt. Do not use pruning. Draw the first three levels decision tree that you obtain. For each node that you draw, if it is a leaf node, write down the label that will be predicted for this node, as well as how many of the training data points lie in this node. If it is an internal node, write down the splitting rule for the node, as well as how many of the training data points lie in this node. (Hint: If your code is correct, the root node will involve the rule Feature 5 < 0.5.)



2. What is the training and test error of your classifier in part (1), where test error is measured on the data in pa2test.txt?

The training error of the classifier in part (1) is 0 and the test error of the classifier in part (1) is 0.173.

The reason why the training error is 0 is that the decision tree is built by all the training data.

3. Now, prune the decision tree developed in part (1) using the data in pa2validation.txt. While selecting nodes to prune, select them in Breadth-First order, going from left to right (aka, from the Yes branches to the No branches). Write down the validation and test error after 1 and 2 rounds of pruning (that is, after you have pruned 1 and 2 nodes from the tree.)

After the first round of pruning, the validation error is 0.122 and the test error is 0.117.

After the second round of pruning, the validation error is 0.107 and the the test error is 0.103.

Compared with the result in part (2), we can see that after pruning, both the test error and the validation error decrease. Thus, we are able to reduce overfitting through pruning the decision tree.

4. Download the file pa2features.txt from the class website. This file provides a description in order of each of the features – that is, it tells you what each coordinate means. Based on the feature descriptions, what do you think is the most salient or prominent feature that predicts credit card default? (Hint: More salient features should occur higher up in the ID3 Decision tree.)

Since the root node is in the form of "Feature 5 < 0.5", therefore the most prominent feature will be "Payment Delay September".

The reason why we look at the root node is that the feature and threshold value pair at the root node can provide the maximum information gain.

```python
import math
import numpy
import operator

def loadData():
    # open the training data file
    data_file = open("pa2train.txt")
    # open the feature file
    features_file = open("pa2features.txt")
    # list that stores the training data
    data = []
    # list that stores the features
    features = []
    # read the training data file
    for line in data_file:
        d = line.split()
        for i in range(0, 23):
            d[i] = float([i])
        if d[22] == 1:
            d[22] = "yes"
        else:
            d[22] = "no"
        data.append(data)
    # read the features file
    for line in features_file:
        features.append(line)
    return data, features

def calculateEntropy(data):
    # dictionary that maps label to its count
    labels = {}
    # initialize the dictionary
    for i in data:
        labels[i[-1]] = 0
```

```
    # update the dictionary
    for i in data:
        labels[i[-1]] = labels[i[-1]] + 1
    h = 0.0
    for i in labels:
        # calculate the probability
        p = float(labels[i]) / float(len(data))
        if p != 0:
            # calculate the entropy
            h = h - p * math.log(p, 2)
    return h

def split(data, feature, value):
    splitedData = []
    for i in data:
        # split the feature vector
        if i[feature] == value:
            splitedFeatureVector = i[:feature]
            splitedFeatureVector = i[feature + 1:]
            splitedData.append(splitedFeatureVector)
    return splitedData

def getBestSplitFeature(data):
    h = calculateEntropy(data)
    bestFeature = -1
    bestInformationGain = 0.0
    # iterate through the features
    for feature in range(0, 22):
        # get all the values for each feature
        values = []
        for j in data:
            values.append(j[feature])
        valuesSet = set(values)
```

```python
        conditionalH = 0.0
        for value in valuesSet:
            # make the split
            splitedData = split(data, feature, value)
            p = float(len(splitedData)) / float(len(data))
            # calculate the conditional entropy
            conditionalH = conditionalH + p * calculateEntropy(splitedData)
            # get the information gain
            informationGain = h - conditionalH
            # get the best feature for splitting
            if (informationGain > bestInformationGain):
                bestFeature = feature
                bestInformationGain = informationGain
        return bestFeature

def getMajority(labels):
    # dictionary that maps label to its count
    labelCount = {}
    # initialize the dictionary
    for i in labels:
        labelCount[i] = 0
    # update the dictionary
    for i in labels:
        labelCount[i] = labelCount[i] + 1
    # sort the dictionary
    labelCount = sorted(labelCount.items(), key = operator.itemgetter(1))
    # get the label
    label = labelCount[len(labelCount) - 1][0]
    return label

def buildTree(data,features):
    # list that stores all the labels
    labels = []
```

```
    for i in data:
        labels.append(i[-1])
    # if there is only one label in the list
    length = len(labels)
    if labels.count(labels[0]) == length:
        return labels[0]
    # if there is no features left
    if len(data[0]) == 1:
        return getMajority(labels)
    # get the best feature
    bestFeature = getBestSplitFeature(data)
    # get the label of the best feature
    bestLabel = features[bestFeature]
    # build the tree
    tree = {bestLabel:{}}
    # get the values for the best feature
    featureValues = []
    for i in data:
        featureValues.append(i[bestFeature])
    featureValues = set(featureValues)
    new_features = []
    # remove the best feature from the features list
    for i in features:
        if i != features[bestFeature]:
            new_features.append(i)
    # recursion
    for values in featureValues:
        # make the split
        splitedData = split(data,bestFeature,values)
        tree[bestLabel][values] = buildTree(splitedData,new_features)
    return tree

def prune(tree, validation_data):
    queue = []
```

```
    queue.append(tree[0])
    # go through the tree with BFS order
    while len(queue) != 0:
        key = queue.pop(0)
        # calculate the current validation error
        old_error = getError(tree[key][0], validation_data)
        getMajority(tree[key][0])
        new_error = getError(tree[key][0], validation_data)
        if new_error <= old_error:
            return new_error
        for i in tree[key][1]:
            queue.append(i)
    # the tree can not be pruned
    return -1


if __name__ == '__main__':
    # load the data
    data, features = loadData()
    # build the decision tree
    tree = buildTree(data, features)
    # print the tree to see the top three levels
    print(tree)
    # read the validation data
    validation_file = open("pa2validation.txt")
    validation_data = []
    for line in validation_file:
        d = line.split()
        for i in range(0, 23):
            d[i] = float([i])
        if d[22] == 1:
            d[22] = "yes"
        else:
```

```
            d[22] = "no"
        validation_data.append(d)
    # first round of pruning
    prune(tree, validation_data)
    # second round of pruning
    prune(tree, validation_data)
    # print the pruned tree
    print(tree)
```