PA2

August 22, 2019

```
In [ ]: Question 1:
            Level 1: Root
            split_rule:  (4, 0.5)
            Is it pure?  False
            Number of data points:  2000

            Level 2: Left
            split_rule:  (0, 415000.0)
            Is it pure?  False
            Number of data points:  1319

            Level 2: Right
            split_rule:  (4, 1.5)
            Is it pure?  False
            Number of data points:  681

            Level 3: LeftLeft
            split_rule:  (16, 2506.5)
            Is it pure?  False
            Number of data points:  1284

            Level 3: LeftRight
            split_rule:  (20, 208.0)
            Is it pure?  False
            Number of data points:  35

            Level 3: RightLeft
            split_rule:  (19, 584.5)
            Is it pure?  False
            Number of data points:  292

            Level 3: RightRight
            split_rule:  (20, 2006.0)
            Is it pure?  False
            Number of data points:  389

        Question 2:
            Training Error :  0.0
```

```
    Test Error :  0.173

Question 3:
    Prune time:  1
    validate_error:  0.12
    test_error:  0.119
    Prune time:  2
    validate_error:  0.107
    test_error:  0.103

Question 4:
    Since more salient features should be used in decisions higher up inthe ID3 Decisio
    As showed in Question 1, we use index 4 which is feature 5 as the split_rule for th
    According to pa2features.txt, feature 5 is PAYMENT_DELAY_SEPTEMBER. It is the most
```

```
In [1]: import numpy as np

        train_file = open("pa2train.txt","r")
        train = [line.strip() for line in train_file]
        train = [[float(i) for i in line.split()] for line in train]
```

```
In [2]: def count_total(target,data):
            count = 0
            for i in range(len(data)):
                if(data[i][-1] == target):
                    count += 1
            return count
```

```
In [3]: # count number of points on the left side of boundary
        def count_left_total(boundary,index,features):
            count = 0
            for i in range(len(features[index])):
                if(features[index][i][0] < boundary):
                    count +=1
                else:
                    return count
            return count
```

```
In [4]: def count_left_label(boundary,index,features,label):
            count = 0
            for i in range(len(features[index])):
                if(features[index][i][0] < boundary):
                    if(features[index][i][1] == label):
                        count += 1
                else:
                    return count
            return count
```

```
In [5]: def count_right_label(boundary,index,features,label):
            count = 0
            for i in range(len(features[index])):
                if(features[index][i][0] > boundary):
                    if(features[index][i][1] == label):
                        count += 1
            return count
```

```
In [6]: def entropy_selection(data):
            # correspond each feature with label
            features = []
            for i in range(22):
                features.append([(x[i],x[-1]) for x in data])
            # sort each feature in features
            for i in range(len(features)):
                features[i].sort()
            # find all possible split for each feature
            split = []
            #print(features[21])
            for i in range(len(features)):
                temp = []
                for j in range(len(features[0])-1):
                    if(features[i][j][0] != features[i][j+1][0]):
                        boundary = (features[i][j][0] + features[i][j+1][0])/2
                        temp.append(boundary)
                split.append(temp)

            #calculate total number of label 0 and 1 in the data
            count_0 = count_total(0,data)
            count_1 = count_total(1,data)
            total_num = count_0 + count_1

            entropy_mat = []
            min_h = 100000000
            min_i = 0
            min_j = 0
            #calculate entropy for each boudary and find the min one
            for i in range(len(split)):
                 for j in range(len(split[i])):
                        entropy_temp = 0
                        cur_boundary = split[i][j]
                        tmp = []
                        # number of points on the left and right side of boundary
                        left = count_left_total(cur_boundary,i,features)
                        right = total_num - left

                        left_0 = count_left_label(cur_boundary,i,features,0)
                        left_1 = left - left_0

                        right_0 = count_right_label(cur_boundary,i,features,0)
                        right_1 = right - right_0

                        if(left_0 == 0):
                            h_left0 = 0
                        else:
                            h_left0 = (left_0/left)*np.log(left_0/left)

                        if(left_1 == 0):
                            h_left1 = 0
                        else:
                            h_left1 =(left_1/left)*np.log(left_1/left)

                        if(right_0 == 0):
                            h_right0 = 0
                        else:
                            h_right0 =(right_0/right)*np.log(right_0/right)
                        if(right_1 == 0):
                            h_right1 = 0
                        else:
                            h_right1 =(right_1/right)*np.log(right_1/right)

                        h_left = -h_left0 - h_left1
                        h_right = -h_right0 - h_right1

                        h = (left/total_num)*h_left + (right/total_num)*h_right
                        tmp += h

                        # find min entropy
                        min_h = min(h,min_h)
                        if(min_h == h ):
                            min_i = i
                            min_j = j

                 #entropy_mat += [tmp]
            return(min_i,split[min_i][min_j])
```

```
In [7]: class Node:
            def __init__(self, data = None,split_rule = None,
                         right = None, left = None, label = None, pure = False, ):
                self.data = data
                self.split_rule = split_rule
                self.right = right
                self.left = left
                self.label = label
                self.pure = pure
```

```
In [8]: # building the tree using training data
        queue = []
```

```python
root = Node(data=train)
queue.append(root)
while(len(queue) != 0):
    curNode = queue.pop(0)
    feature_index,boundary = entropy_selection(curNode.data)
    #boundary = ret[2]
    #feature_index = ret[1]
    curNode.split_rule = (feature_index,boundary)
    cur_data = curNode.data

    # split data according to split_rule
    leftData = []
    rightData  = []
    for i in range(len(cur_data)):
        if(cur_data[i][feature_index] < boundary):
            leftData += [cur_data[i]]
        else:
            rightData += [cur_data[i]]

    #print(len(leftData))
    #print(len(rightData))

    #check if left is pure
    leftNode = Node(data = leftData)
    leftNode.label = None
    leftNode.pure = True
    if(len(leftData) > 1):
        for i in range(len(leftData)-1):
            if(leftData[i][-1] != leftData[i+1][-1]):
                leftNode.pure = False
                break

    if(leftNode.pure):
        leftNode.label = leftData[0][-1]
    else:
        queue.append(leftNode)
    curNode.left = leftNode

    #check if right is pure

    rightNode = Node(data = rightData)
    rightNode.label = None
    rightNode.pure = True
    if(len(rightData) > 1):
        for i in range(len(rightData)-1):
            if(rightData[i][-1] != rightData[i+1][-1]):
                rightNode.pure = False
                break
    if(rightNode.pure):
        rightNode.label = rightData[0][-1]
    else:
        queue.append(rightNode)
    curNode.right = rightNode
```

```
In [9]: print("Level 1: Root ")
        print("split_rule: ",root.split_rule)
        print("Is it pure? ", root.pure)
        print("Number of data points: ",len(root.data))
        print()

        print("Level 2: Left ")
        print("split_rule: ",root.left.split_rule)
        print("Is it pure? ", root.left.pure)
        print("Number of data points: ",len(root.left.data))
        print()

        print("Level 2: Right ")
        print("split_rule: ",root.right.split_rule)
        print("Is it pure? ", root.right.pure)
        print("Number of data points: ",len(root.right.data))
        print()

        print("Level 3: LeftLeft ")
        print("split_rule: ",root.left.left.split_rule)
        print("Is it pure? ", root.left.left.pure)
        print("Number of data points: ",len(root.left.left.data))
        print()

        print("Level 3: LeftRight ")
        print("split_rule: ",root.left.right.split_rule)
        print("Is it pure? ", root.left.right.pure)
        print("Number of data points: ",len(root.left.right.data))
        print()

        print("Level 3: RightLeft ")
        print("split_rule: ",root.right.left.split_rule)
        print("Is it pure? ", root.right.left.pure)
        print("Number of data points: ",len(root.right.left.data))
        print()

        print("Level 3: RightRight ")
        print("split_rule: ",root.right.right.split_rule)
        print("Is it pure? ", root.right.right.pure)
        print("Number of data points: ",len(root.right.right.data))
        print()
```

```
Level 1: Root
split_rule:  (4, 0.5)
Is it pure?  False
Number of data points:  2000

Level 2: Left
split_rule:  (0, 415000.0)
Is it pure?  False
Number of data points:  1319

Level 2: Right
split_rule:  (4, 1.5)
Is it pure?  False
Number of data points:  681

Level 3: LeftLeft
split_rule:  (16, 2506.5)
Is it pure?  False
Number of data points:  1284

Level 3: LeftRight
split_rule:  (20, 208.0)
Is it pure?  False
Number of data points:  35

Level 3: RightLeft
split_rule:  (19, 584.5)
Is it pure?  False
Number of data points:  292

Level 3: RightRight
split_rule:  (20, 2006.0)
Is it pure?  False
Number of data points:  389
```

```
In [10]: test_file = open("pa2test.txt","r")
         test = [line.strip() for line in test_file]
         test = [[float(i) for i in line.split()] for line in test]
```

```
In [11]: def check(root,feature_vec):
             cur_node = root
             while(cur_node.pure != True):
                 (index,boundary) = cur_node.split_rule
                 if(feature_vec[index] < boundary):
                     cur_node = cur_node.left
                 else:
                     cur_node = cur_node.right

             return cur_node.label

In [12]: # check training error
         errorNum = 0
         for i in range(len(train)):
             if(check(root,train[i]) != train[i][-1]):
                 errorNum += 1
         training_error = errorNum / len(train)

         # check test error
         errorNum = 0
         for i in range(len(test)):
             if(check(root,test[i]) != test[i][-1]):
                 errorNum += 1
         test_error = errorNum / len(test)

In [13]: print("Training Error : ", training_error)
         print("Test Error : ", test_error)

Training Error :  0.0
Test Error :  0.173


In [14]: # pruning the tree using validate_data
         file_validate = open('pa2validation.txt', 'r')
         validate = [line.strip() for line in file_validate]
         validate = [[float(i) for i in line.split()] for line in validate]

In [15]: queue = []
         queue.append(root)
         prune_time = 0
         while(len(queue) != 0):

             # calculate validate error on original tree
             errorNum = 0
             for i in range(len(validate)):
                 if(check(root,validate[i]) != validate[i][-1]):
                     errorNum += 1
             validate_error = errorNum / len(validate)

             # prune the tree
             cur = queue.pop(0)

             if(cur.pure == True):
                 continue
             cur.pure = True
             # go over all data points to find majority label
             label_0 = 0
             label_1 = 0

             for i in range(len(cur.data)):
                 if(cur.data[i][-1] == 0):
                     label_0 += 1
                 else:
                     label_1 += 1

             if(label_0 > label_1):
                 cur.label = 0
             else:
                 cur.label = 1

             # calculate the validate error after pruning
             errorNum = 0
             for i in range(len(validate)):
                 #print(check(root,validate[i]),"  ",validate[i][-1] )
                 if(check(root,validate[i]) != validate[i][-1]):
                     errorNum += 1
             error_after = errorNum / len(validate)

             #print(validate_error, "  ", error_after)

             if(error_after < validate_error):
                 prune_time += 1
                 # calculate test error
                 errorNum = 0
                 for i in range(len(test)):
                     if(check(root,test[i]) != test[i][-1]):
                         errorNum += 1
                 test_error = errorNum / len(test)

                 print("Prune time: ", prune_time)
                 print("validate_error: ", error_after)
                 print("test_error: ", test_error)

             else:
                 cur.pure = False
                 queue.append(cur.left)
                 queue.append(cur.right)
```

```
Prune time:  1
validate_error:  0.12
test_error:  0.119
```

```
Prune time:  2
validate_error:  0.107
test_error:  0.103
```

```
In [ ]:
```