University of Waterloo
CS 486
r2knowle: 2023-10-1

Assignment # 3

Q1.1) Q1.2) If we only look at day 1 we get the following observations:

	Had soup on day 1	Didn't have soup on day 1
Liked the meal	2	0
Didn't like the meal	1	2

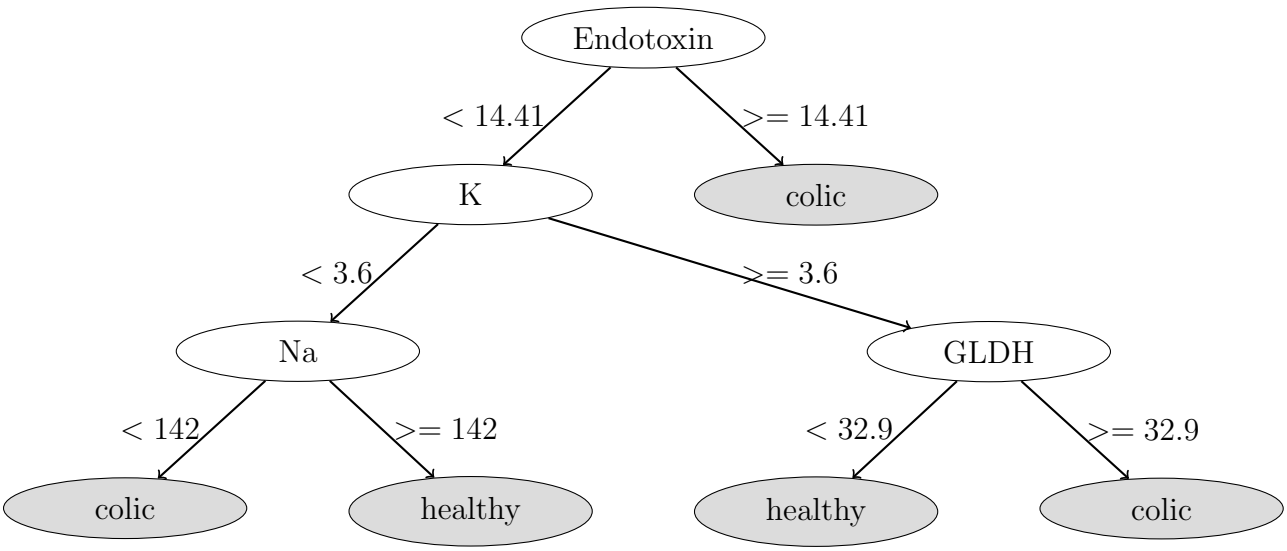
Q4.1) Below is the code used to produce the decision tree, as well as get the number of values predicted correctly vs incorrectly:

```
import numpy as np

# ============= Loss Implementation =============
def entropyLoss(vals_y):

    count = 0.0
    for val in vals_y:
        if val == "healthy.\n":
            count += 1.0
    p = 0
    if count != 0:
        p = count / float(len(vals_y))

    if p == 1 or p == 0:
        return 0

    return -p * np.log2(p) - (1 - p) * np.log2(1 - p)

# ============= Decision Tree Implementation =============

class DecisionTree:
    threshold = 0
    thresholdFeature = 0
    values = []
    leftTree = -1
    rightTree = -1
    classification = 0
    numLeft = 0
    numRight = 0
    counter = 0

    # This method determines if all values are homogenous
    def allTheSame(self, vals):
        if len(vals) == 0:
            return True
        originalValue = vals[0][-1]
        for val in vals:
            if (val[-1] != originalValue):
                return False
        return True


    # Given a dataset, this function will determine where the split is to minimize entropy loss
        (maximize info gain)
    def findBestThreshold(self, vals, loss):
        bestLoss = 9999999
        bestFeature = 0
        bestThreshold = 0

        width = len(vals[0]) - 1
        for test_entry in vals:
            for idx in range(0, width):
                left = []
                right = []

                for entry in vals:
```

```
                # Note that picking a threshold that is <= two consecutive elements is the same
                    as >= the top element.
                #   as no elements by definition can be between the threshold and the proposed
                    threshold
                if (float(entry[idx]) >= float(test_entry[idx])):
                    right.append(entry[-1])
                else:
                    left.append(entry[-1])

            leftWeight = len(left) / (len(left) + len(right))
            rightWeight = len(right) / (len(left) + len(right))

            totalLoss = loss(left) * leftWeight + loss(right) * rightWeight
            if totalLoss < bestLoss:
                bestLoss = totalLoss
                bestFeature = idx
                bestThreshold = test_entry[idx]

    return [bestFeature, bestThreshold]

def __init__(self, vals, loss):
    if self.allTheSame(vals):
        self.values = vals
    else:
        x = self.findBestThreshold(vals, loss)
        self.thresholdFeature = x[0]
        self.threshold = x[1]

        leftTreeVals = []
        rightTreeVals = []
        for val in vals:
            if float(val[self.thresholdFeature]) >= float(self.threshold):
                rightTreeVals.append(val)
            else:
                leftTreeVals.append(val)

        self.leftTree = DecisionTree(leftTreeVals, loss)
        self.rightTree = DecisionTree(rightTreeVals, loss)

def classify(self, item):
    if self.leftTree == -1:
        if item[-1] == self.values[0][-1]:
            return 1
        print(item[-1])
        return 0
    else:
        if float(item[self.thresholdFeature]) >= float(self.threshold):
            return self.rightTree.classify(item)
        return self.leftTree.classify(item)

def printTree(self, indx, cols):
    if len(self.values) == 0:
        print("LV ", indx, ": ", cols[self.thresholdFeature], " ", self.threshold)
    else:
        print("LV", indx, ": ", self.values[0][-1].split("\n")[0])
    if self.leftTree != -1:
        self.leftTree.printTree(indx+1, cols)
    if self.rightTree != -1:
        self.rightTree.printTree(indx+1, cols)
```

```
    def predict(self, items):
        count = 0
        for val in items:
            count += self.classify(val)
        return count


# ============= Testing of Values =============

trainingFile = open("horseTrain.txt", "r")
lines = trainingFile.readlines()
training = []
for line in lines:
    training.append(line.split(","))

columns = ["K", "Na", "CL", "HCO3", "Endotoxin", "Aniongap", "PLA2", "SDH", "GLDH", "TPP", "Breath
    rate",
          "PCV", "Pulse rate", "Fibrinogen", "Dimer", "FibPerDim"]

dt = DecisionTree(training, entropyLoss)
dt.printTree(1, columns)
print("Correct Predictions:", dt.predict(training), "out of", len(training))

testFile = open("horseTest.txt", "r")
lines = testFile.readlines()
testing = []
for line in lines:
    testing.append(line.split(","))
print("Correct Predictions:", dt.predict(testing), "out of", len(testing))
```

Q4.2) This gives us the following decision tree:



Q4.3) For the training set we get that we correctly classifiy:

132 out of 132 training data points

Q4.4) For the test set we get that we correctly classify:

13 our of the 13 testing data points