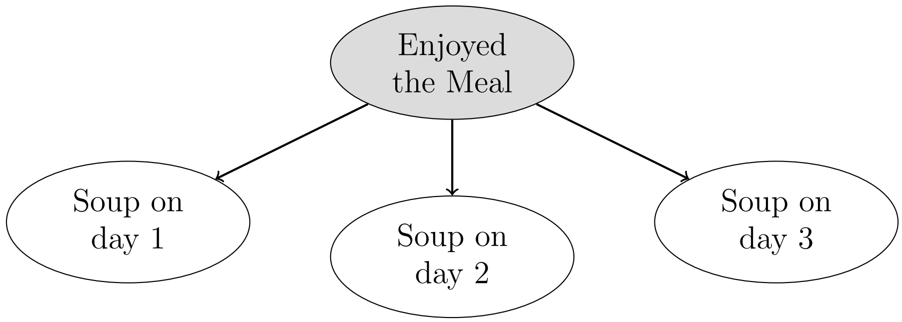
University of Waterloo
CS 486
r2knowle: 2023-11-11

Assignment # 3

Q1.1) To begin, we get the following diagram for the Naive Bayes Model based on the attributes of eating soup on days 1,2 and 3:



From this we can get the derive the following thetas:

$$\begin{aligned}
\theta &= \text{P(C=True)} \\
\theta_{1,1} &= \text{P(Soup on day 1 = True | C = True)} \\
\theta_{1,2} &= \text{P(Soup on day 1 = True | C = False)} \\
\theta_{2,1} &= \text{P(Soup on day 2 = True | C = True)} \\
\theta_{2,2} &= \text{P(Soup on day 2 = True | C = False)} \\
\theta_{3,1} &= \text{P(Soup on day 3 = True | C = True)} \\
\theta_{3,2} &= \text{P(Soup on day 3 = True | C = False)}
\end{aligned}$$

After applying Lapace smoothing we get the following values for theta:

$$\theta = \frac{2+1}{5+2} = \frac{3}{7}$$

$$\theta_{1,1} = \tfrac{2+1}{2+2} = \tfrac{3}{4} \quad \theta_{1,2} = \tfrac{1+1}{3+2} = \tfrac{2}{5}$$

$$\theta_{2,1} = \tfrac{2+1}{2+2} = \tfrac{3}{4} \quad \theta_{2,2} = \tfrac{0+1}{3+2} = \tfrac{1}{5}$$

$$\theta_{3,1} = \tfrac{1+1}{2+2} = \tfrac{2}{4} \quad \theta_{3,2} = \tfrac{2+1}{3+2} = \tfrac{3}{5}$$

Q1.2) To solve we have 2 sets of 2 probabilities. In the case of having soup on day 1 we have (note Dx represents soup on day x):

$$\begin{aligned}
P(\text{Enjoy Meal}|D_1, \neg D_2, \neg D)3) &= \alpha \times \theta \times \theta_{1,1} \times (1-\theta_{2,1}) \times (1-\theta_{3,1}) \\
&= \frac{3}{7} \times \frac{3}{4} \times \frac{1}{4} \times \frac{1}{2} \\
&= \frac{9}{224}
\end{aligned}$$

$$
\begin{aligned}
P(\text{Didnt Enjoy Meal}|D_1, \neg D_2, \neg D)3) &= \alpha \times (1-\theta) \times \theta_{1,2} \times (1-\theta_{2,2}) \times (1-\theta_{3,2}) \\
&= \frac{4}{7} \times \frac{2}{5} \times \frac{4}{5} \times \frac{2}{5} \\
&= \frac{64}{875}
\end{aligned}
$$

In comparison, if we have soup on both days 1 and 2 we get the following probabilities:

$$
\begin{aligned}
P(\text{Enjoy Meal}|D_1, D_2, \neg D)3) &= \alpha \times \theta \times \theta_{1,1} \times \theta_{2,1} \times (1-\theta_{3,1}) \\
&= \frac{3}{7} \times \frac{3}{4} \times \frac{3}{4} \times \frac{1}{2} \\
&= \frac{27}{224}
\end{aligned}
$$

$$
\begin{aligned}
P(\text{Didnt Enjoy Meal}|D_1, D_2, \neg D)3) &= \alpha \times (1-\theta) \times \theta_{1,2} \times \theta_{2,2} \times (1-\theta_{3,2}) \\
&= \frac{4}{7} \times \frac{2}{5} \times \frac{1}{5} \times \frac{2}{5} \\
&= \frac{16}{875}
\end{aligned}
$$

Therefore we can see that the probability Alex enjoys his meal goes up if he had soups on day 1 and 2, as well as his probability for not liking the meal also goes down. Therefore we can conclude he is more likely to like his meal is he has soups on both days 1 and 2.

Q2.1) As taught in class we know the expected value is equal to:

$$\text{Expected Value} = \sum_s P(s)U(s)$$

In our case we get that this is equivalent to:

$$\begin{aligned}\text{Expected Value} &= \sum_{i=1}^{\infty} \frac{1}{2^i} \times 2^i \\ &= \sum_{i=1}^{\infty} 1 \\ &= \infty\end{aligned}$$

Thus showing the expected value of this game is infinite.

Q2.2) If I could play the game regardless of how much I bid, I would always only pay 0 as that would maximize my utility of winning, otherwise I would pay a maximum of 100 :).

Q2.3) Given this new utility function we get that the expected utility is:

$$\begin{aligned}\text{Expected Uility} &= \sum_{i=1}^{\infty} \frac{1}{2^i} \times (a \log_2(2^i) + b) \\ &= a\sum_{i=1}^{\infty}(\frac{i}{2^i}) + b\sum_{i=1}^{\infty}(\frac{1}{2^i}) \\ &= a\sum_{i=1}^{\infty}(\frac{i}{2^i}) + b\sum_{i=1}^{\infty}(\frac{1}{2})^i\end{aligned}$$

Note that the right equation is equivalent to the geometric series with $x = \frac{1}{2}$ (starting at the second term) and so we get:

$$\begin{aligned}&= a\sum_{i=1}^{\infty}(\frac{i}{2^i}) + b(\frac{1}{1-\frac{1}{2}} - 1) \\ &= a\left(\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + ...\right) + b \\ &= 2a + b\end{aligned}$$

Thus we get that the expected utility from playing this game is equal to 2a + b.

Q2.4) To begin we can get that our expected utility is:

$$\text{Expected Utility} = 2a + b$$

Moreover the utility of money we get is equal to the money we earn - the price we needed to bid. Lets assume that our bid is equal to our entire wealth. Thus we get:

$$\begin{aligned}
a\log_2(k-x)+b &= 2a+b \\
\log_2(k-x) &= 2 \\
k-x &= 4 \\
-x &= 4-k \\
x &= k-4
\end{aligned}$$

Therefore we have that x = k-4.

Q3.1a) Before we determine the optimal policy give a specific risk probability, we will first determine the value function for each action. Starting with action **a** we have the following value functions for each state:

$$S_1 = \gamma(0.1S_1 + 0.9S_2)$$
$$S_2 = \gamma(p_s S_2 + (1 - p_s)S_4)$$
$$S_4 = 10 + \gamma(0.1S_4 + 0.9S_1)$$

For action **b** we have the following value functions:

$$S_1 = \gamma(0.1S_1 + 0.9S_3)$$
$$S_3 = \gamma(0.1S_3 + (0.9 - p_r)S_4 + p_r S_5)$$
$$S_4 = 10 + \gamma(0.1S_4 + 0.9S_1)$$
$$S_5 = -10 + \gamma(0.1S_5 + 0.9S_1)$$

If we look at action **a** we get the following when p_s is 0.2:

$$\begin{aligned} S_1 &= 0.95(0.1S_1 + 0.9S_2) \\ S_1 &= 0.095S_1 + 0.855S_2 \\ 0.905 \times S_1 &= 0.855S_2 \\ S_1 &= 0.9447513S_2 \end{aligned}$$

$$\begin{aligned} S_2 &= 0.95(0.2S_2 + 0.8S_4) \\ 0.81 \times S_2 &= 0.762_4 \\ S_2 &= 0.9382716S_4 \end{aligned}$$

$$\begin{aligned} S_4 &= 10 + 0.95(0.1S_4 + 0.9S_1) \\ 0.905 \times S_4 &= 10 + 0.855S_1 \\ S_4 &= 11.049723 + 0.9447513S_1 \end{aligned}$$

To solve we can plug S_1 in for the equation of S_4

$$\begin{aligned} S_4 &= 11.049723 + 0.9447513(0.9447513S_2) \\ S_4 &= 11.049723 + 0.9447513(0.9447513(0.9382716S_4)) \\ S_4 &= 11.049723 + 0.837459S_4 \end{aligned}$$

$$\begin{aligned} S_4 &= 67.98 \\ S_2 &= 63.78 \\ S_1 &= 60.26 \end{aligned}$$

Now looking at policy b, lets pick p_r to be 0.01:

$$\begin{aligned}
S_1 &= 0.95(0.1S_1 + 0.9S_3)\\
S_1 &= 0.095S_1 + 0.855S_3\\
0.905 \times S_1 &= 0.855S_3\\
S_1 &= 0.9447513S_3
\end{aligned}$$

$$\begin{aligned}
S_3 &= 0.95(0.1S_3 + 0.89S_4 0.01S_5)\\
0.905 \times S_3 &= 0.8455S_4 + 0.0095S_5\\
S_3 &= 0.93425S_4 + 0.010497S_5
\end{aligned}$$

$$\begin{aligned}
S_4 &= 10 + 0.95(0.1S_4 + 0.9S_1)\\
0.905 \times S_4 &= 10 + 0.855S_1\\
S_4 &= 11.049723 + 0.9447513S_1
\end{aligned}$$

$$\begin{aligned}
S_5 &= -10 + 0.95(0.1S_4 + 0.9S_1)\\
0.905 \times S_5 &= 10 + 0.855S_1\\
S_5 &= -11.049723 + 0.9447513S_1
\end{aligned}$$

Solving this gives us:

$$\begin{aligned}
S_1 &= 61.52\\
S_3 &= 65.12\\
S_4 &= 69.17\\
S_5 &= 47.07
\end{aligned}$$

Thus action b is better then $pr = 0.01$. If we do the same for $pr = 0.03$ we get the following values:

$$\begin{aligned}
S_1 &= 58.72\\
S_3 &= 62.16\\
S_4 &= 66.52\\
S_5 &= 44.43
\end{aligned}$$

In this case we can see that action a is better when $pr = 0.03$. To find the indifference we will set S_1 to be the same at action a and solve:

$$
\begin{aligned}
S_3 &= 0.95(0.1S_3 + (1-pr)S_4 + prS_5) \\
0.905 \times S_3 &= 0.95(1-pr)(11.049723 + 0.9447513S_1) + 0.95pr(-11.049723 + 0.9447513S_1) \\
S_3 &= 0.95(0.1S_3 + (1-pr)S_4 + prS_5) \\
57.7245 &= 64.58141(1-pr) + 0.9447513S_1) + 43.58694pr) \\
pr &= 0.01899
\end{aligned}
$$

Thus we are indifferent when pr = 0.018989.

If we do the same with ps = 0.6 and pr = 0.1 we get with policy a (I didnt list all the steps for brevity):

$$
\begin{aligned}
S_4 &= 52.31 \\
S_2 &= 46.23 \\
S_1 &= 43.67
\end{aligned}
$$

We then get the following for policy **b**:

$$
\begin{aligned}
S_1 &= 48.93 \\
S_3 &= 51.80 \\
S_4 &= 57.28 \\
S_5 &= 35.18
\end{aligned}
$$

Thus we can see that policy b is better in this case. If we change pr to be 0.2 we then get the following for **b** (policy a is unchanged):

$$
\begin{aligned}
S_1 &= 34.95 \\
S_3 &= 37.00 \\
S_4 &= 44.07 \\
S_5 &= 21.97
\end{aligned}
$$

In this case we can see that policy a is better. Lastly the we are indifferent when pr is 0.1375.

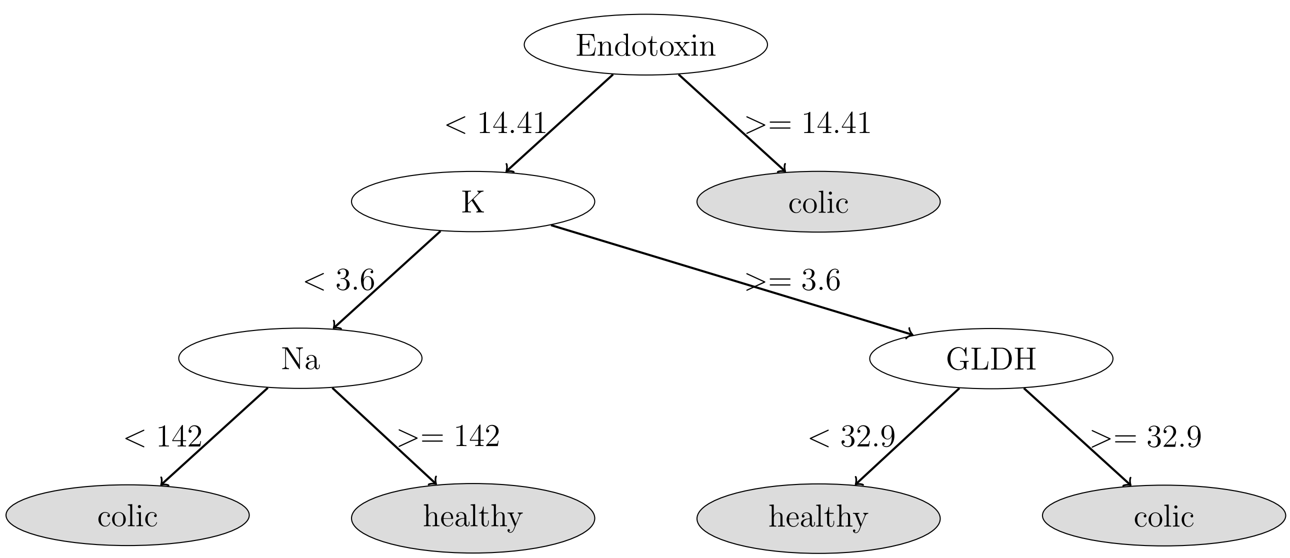
Q4.1) Below is the code used to produce the decision tree, as well as get the number of values predicted correctly vs incorrectly:

```
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:
```

```python
                # 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)
```

```python
    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