536

Xinyang Wang

189001002

1 HW3

1.1

Before answering the questions, let's first take a look at the structure of the code. Compared to structure of HW2, new parts of GenereteTree-Depth(), GenereteTree-SampleSize(), GenereteTree-Independence() are added.

```
class DescisonTree():
    def __init__(self,m,k):
        self.m = m
        self.k = k
        self.tree = [None]*(2**self.k-1)
        self.label = [None]*(2**(self.k+1)-1)

    def GenerateDataPoint(self):
        '''generate data according to the given pattern '''

    def Partition(self,matrix,x):
        '''partition the data matrix based on x, \
        return two matrixs according to value of x '''

    def InformationGain(self,matrix):
        '''compute the IGs and return max IG and its corresponding x '''

    def GenerateTree(self,matrix,id = 0):
        '''recursively building tree and label'''

    def GenerateTree_Depth(self,matrix,d,id=0):
        '''recursively building tree and label and ceasing at depth of d'''

    def GenerateTree_SampleSize(self,matrix,size,id=0):
        '''recursively building tree and label and ceasing at the node \
        of controled sample size'''

    def GenerateTree_Independence(self,matrix,im,id=0):
        '''recursively building tree and label and ceasing at the node \
        where the data are independent'''

    def GetError(self, matrix):
        '''get error of a built tree  '''

    def GetNumberOfIrrelevant(self):
        '''get the number of irrevelant variables that included in \
        decision tree '''

    if __name__ == '__main__':
```

Then we go back to question 1. For a given value m, the following part of code generates a training data set based on the given pattern

```
    def GenerateDataPoint(self):
        '''generate data according to the given pattern '''
        #compute data point
        matrix = np.zeros((self.m, self.k+1),'int')
        for i in range(self.m):
            count1_x_1 = 0
            count2_x_1 = 0
            matrix[i][0] = 1 if random.random()<0.5 else 0
            for j in range(1,8):
                matrix[i][j] = matrix[i][j-1] if random.random()< \
                0.75 else 1-matrix[i][j-1]
                count1_x_1 += 1 if matrix[i][j]==1 else 0
            for j in range(8,15):
                matrix[i][j] = matrix[i][j-1] if random.random()< \
                0.75 else 1-matrix[i][j-1]
                count2_x_1 += 1 if matrix[i][j]==1 else 0
            if matrix[i][0] == 0:
                matrix[i][self.k] = 1 if count1_x_1 > 3 else 0
            else:
                matrix[i][self.k] = 1 if count2_x_1 > 3 else 0

            for j in range(15,21):
                matrix[i][j] = 1 if random.random()<0.5 else 0

        return matrix
        print(f'generate {self.m} data points successful ')
```

Then the code to fit a decision tree to data points using ID3 is as follows:

```
    def GenerateTree(self,matrix,id = 0):
        '''recursively building tree and label'''
        count = 0
        for i in range(matrix.shape[0]):
            if matrix[i][-1]==0:
                count += 1

        if count/matrix.shape[0]>0.95 :
            self.label[id] = 0
            return
        if count/matrix.shape[0]<0.05 :
            self.label[id] = 1
            return

        pos, IG = self.InformationGain(matrix)
        #print('pos&IG',pos,IG)
        #input()
        self.tree[id] = pos     # in HW3, X begins at 0
        mat0, mat1 = self.Partition(matrix, pos)
        self.GenerateTree(mat0,2*id+1)
        #print('now id', id)
        self.GenerateTree(mat1,2*id+2)
```

Finally, given a decision tree, here is the estimation of error rate of that decision tree on the data points:

```
    def GetError(self, matrix):
        '''get error of a built tree  '''
        if matrix.shape[1]-1 != self.k:
            print('depth of matrix does not comply with depth of the \
            built tree')
            return 0

        right = 0
        for i in range(matrix.shape[0]):
            pos = 0
            id = self.tree[pos]
            while id!=None:
                if matrix[i][id] == 0:
                    pos = 2*pos+1
                else:
                    pos = 2*pos+2
                if pos > len(self.tree):
                    id = None
                    break
                id = self.tree[pos]
                #print('pos',pos,'id',id)
                #input()
            predict = self.label[pos]
            if predict == matrix[i][-1]:
                right += 1

        return 1-right/matrix.shape[0]
```

When we do this repeatedly, we plot the error rate as a function of m.(for each m, we repeat for 50 times and test the decision tree on data points of 5000) And this curve complies with our tuition: the more data(not too many) on which we get the decision tree, the more accurate the decision tree is.

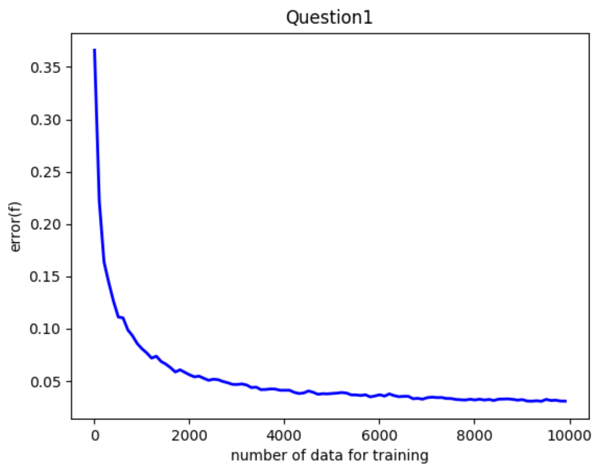

Figure 1: Q1

1.2

Below is the code part for the target of counting the irrelevant variables in our decision tree:

```
    def GetNumberOfIrrelevant(self):
        '''get the number of irrevelant variables that included in \
        decision tree '''
        count_irr = 0
        for i in range(len(self.tree)):
            if self.tree[i] != None:
                count_irr += 1 if self.tree[i] > 14 else 0
        return count_irr
```

For m in range of 10-10000, we repeatedly compute the number of irrelevant variables in our decision tree and take the average. And below, we plot the number as a function of m. As we can see, there are always irrelevant variables taken to build the decision tree. From my opinion, that is unavoidable because there is chance that irrelevant variables have nearly the same value as Y (which means the highest Info gain at that situation).
However, as the number of training data increases, the average number of irrelevant variables taken to build the decision tree seems to decrease, which implies that more data lead to more correct decision tree.

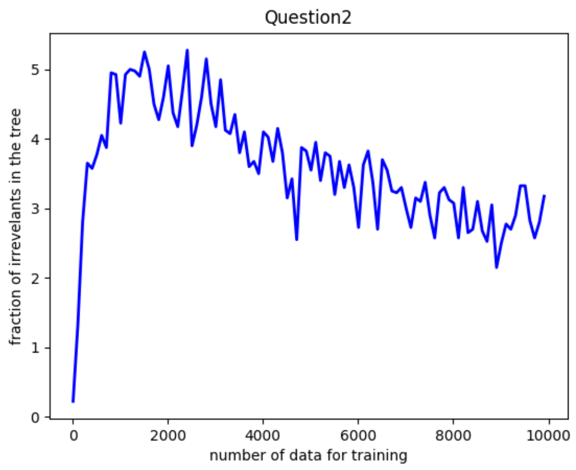

Figure 2: Q2

1.3

Generate a data set of size m= 10000, and set aside 8000 points for training, and 2000 points for testing.

1.3.1 Pruning by Depth:

For max depth in range 0-20, we generate decision tree on training data and applying it to test data to get the error. Below we plot the error as a function of max depth.(red line for error on training data and blue for error on test data).
As we can see, when max depth control reach a certain threshold (9 as to our data), the error on both training data and test data will stop decreasing quickly. Instead the lines become flat.

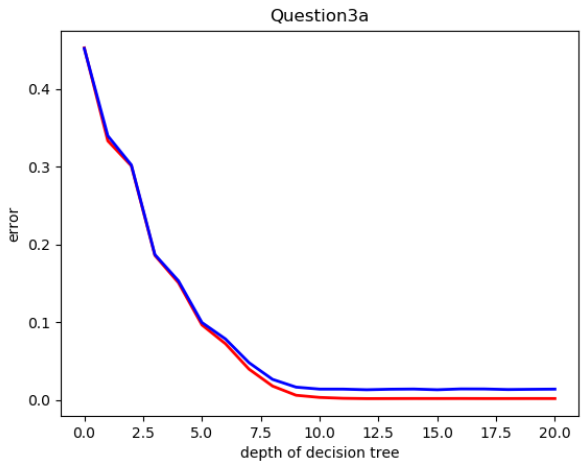

Figure 3: Q3a

1.3.2 Pruning by Sample Size:

For min sample size ranging from 1 to 2^{10}, we generate decision tree on training data and applying it to test data to get the error. Below we plot the error as a function of min sample size.(red line for error on training data and blue for error on test data).

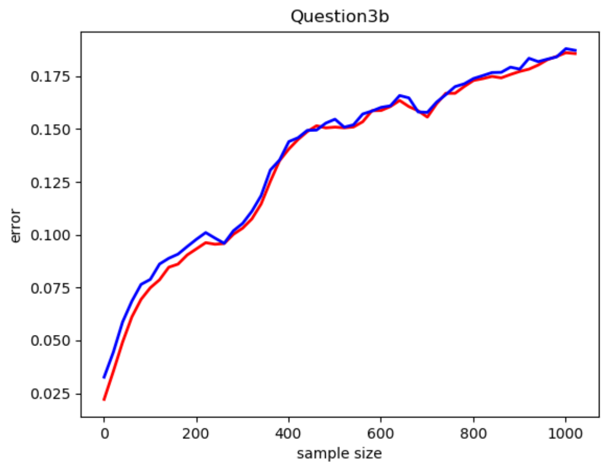

Figure 4: Q3b

It seems that error increases as the min sample size increases. While the chart above has a large step with X-coordinate, so we do not see detailed change. Below is a more detailed chart showing the tendency of error with min sample size ranging from 1 to 10. Here we can see that error reaches the smallest at around 3-5.

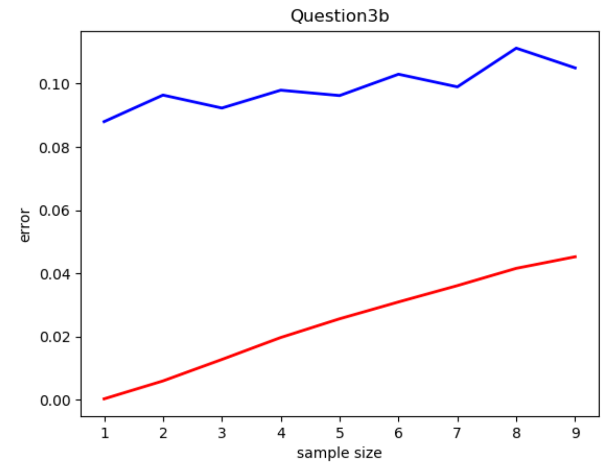

Figure 5: Q3b-more-detailed

1.3.3 Pruning by Significance:

For significance(above which the variable is deemed worth splitting) ranging from 1 to 200, we generate decision tree on training data and apply it to test data to get the error. Below we plot the error as a function of significance.(red line for error on training data and blue for error on test data).

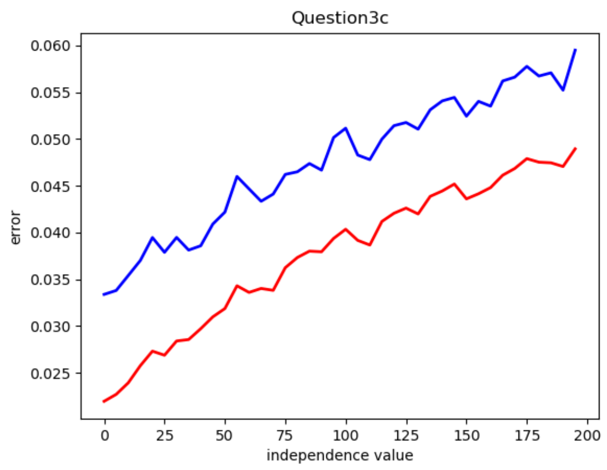

Figure 6: Q3c

The chart above has a large step which ignores some details. Below is a more detailed chart illustrating the change of error with the significance control ranging from 1-10, from which we can see the threshold is around 4.5.

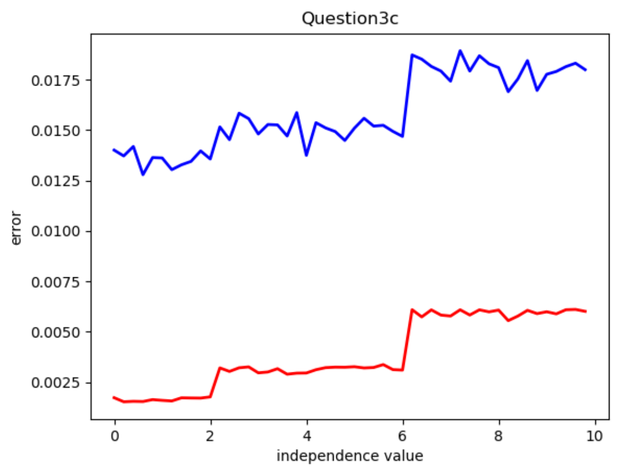

Figure 7: Q3c more detailed

1.4

1.5

From question 3 a), we get the depth threshold of 9. With this threshold, we redo question 2, and below we plot the average number of irrelevant variables in the decision as a function of m.
Compared to the chart in section 3.2, we can see that the average number of irrelevant variables decreases at a higher speed. This is because depth control rule out noises, which are irrelevant variables in this case.

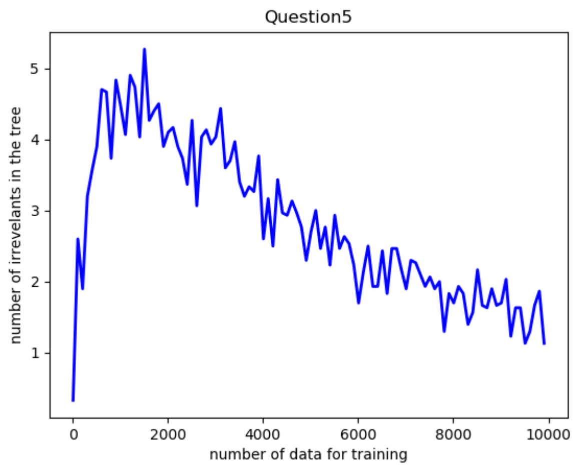

Figure 8: Q5

1.6

From question 3 b), we get the sample size threshold of 5. With this threshold, we redo question 2, and below we plot the average number of irrelevant variables in the decision as a function of m.
Compared to the chart in section 3.2, we can see that the average number of irrelevant variables decreases a little more than without sample size control.

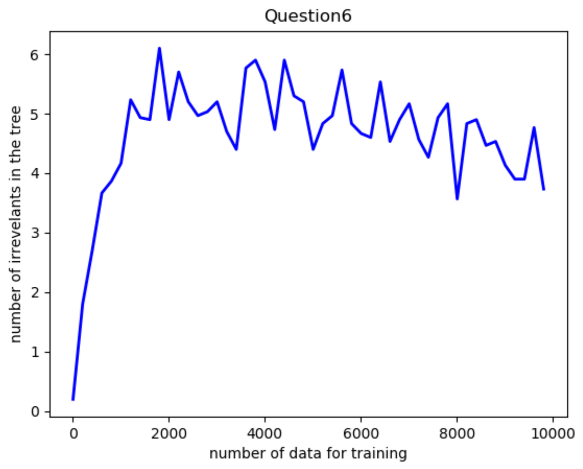

Figure 9: Q6

1.7

From question 3 a), we get the significance threshold of 4.5. With this threshold, we redo question 2, and below we plot the average number of irrelevant variables in the decision as a function of m.
The error is supposed to decrease with the significance control. However in my experiments, it instead increases a little. I tried many times but this situation still occurs. Still trying to find out why.

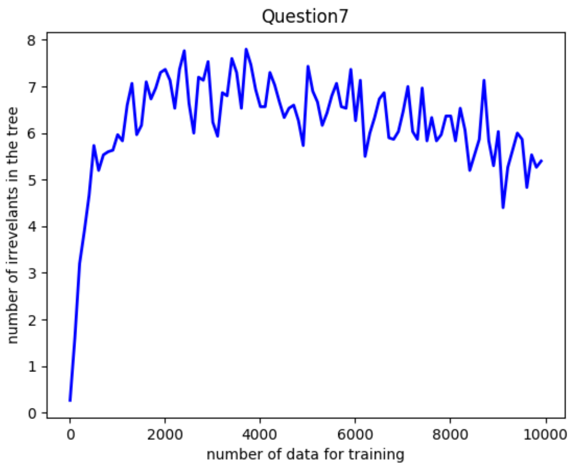
Question7
number of irrevelants in the tree
number of data for training
0
1
2
3
4
5
6
7
8
0
2000
4000
6000
8000
10000

Figure 10: Q7