Diagnosis for the eyes of patients

1. Description:

Implement ID3 algorithm on a dataset that holds a diagnosis for the eyes of patients.

- The diagnosis is based on the following features:
  1. Age: (0) young, (1) adult.
  2. Prescription: (0) myope, (1) hypermetrope.
  3. Astigmatic: (0) no, (1) yes.
  4. Tear production rate: (0) normal, (1) reduced.
  5. Diabetic: (0) not diabetic patient, (1) is a diabetic patient.
- The output classes are:
  1. Need contact lenses (1): the patient should be fitted with a special type of contact lenses.
  2. No contact lenses (0): the patient should not be fitted with a
  3. Special type of contact lenses.

- Task:
  1. Classification using ID3 algorithm.
  2. Each feature has only two attributes 0 or 1.
  3. Output classes are only two values => 0 (no special contactlenses) and 1 (need special contact lenses).

2. Introduction:

ID3 algorithm:

It is algorithm that is used in classification and prediction by making a decision tree by training on a dataset by portioning inputs into set of regions and use a simple predictor for each region.

It's applications:

- ❖ Prediction of unknown result.
- ❖ Classification of large data.
- ❖ Could use in a lot of fields by training in large data to get a specific decision.

2. The algorithm:

ID3:

1. The main steps of the algorithm:

- Step 1: Calculate the gain of each feature on my data.

  Gain = Entropy - $\sum_{i=0}^{All\ children\ of\ attribute} \frac{Pi+Ni}{Total}(child)$

  Entropy = - $\frac{P}{Total}\log_2\frac{P}{Total}$ - - $\frac{N}{Total}\log_2\frac{N}{Total}$

- Step 2: The feature with the maximum gain will be the root node for the best splitting.
- Step 3: For each value of the attributes of this node create a new child node.
- Step 4: Split training to child nodes.
- Step 5: If pure classified Stop else iterate over the new child nodes and split again.

2. The implementation of the algorithm:

```
class ID3:
    dataset = item.getDataset()
    def __init__(self, features):
        self.features = features
        self.buildtree('Root')
    def entropy(self,list):
```

```
def entropy(self,list):
    numberOFZeros = 0
    numberOFOnes = 0
    for i in range(0, len(list)):
        if list[i] == 0:
            numberOFZeros += 1
        elif list[i] == 1:
            numberOFOnes += 1
    length = numberOFOnes + numberOFZeros
    if length == 0:
        return 0
    elif numberOFZeros == 0 and numberOFOnes == 0:
        return 0
    elif numberOFZeros == 0 and numberOFOnes != 0:
        return -((numberOFOnes / length) * math.log2(numberOFOnes / length))
    elif numberOFZeros != 0 and numberOFOnes == 0:
        return -((numberOFZeros / length) * math.log2(numberOFZeros / length))
    else:
        return -(((numberOFZeros / length) * math.log2(numberOFZeros / length)) + ((numberOFOnes / length) * math.log2(numberOFOnes / length)))
def Get_Max_Gain(self):
```

```
def Get_Max_Gain(self):
    Max_Gain = -99999
    age = []
    prescription = []
    astigmatic = []
    tearRate = []
    diabetic = []
    needLenses = []
    maxColumn = 0
    for i in range(0,len(self.dataset)):
        age.append(self.dataset[i].age)
        prescription.append(self.dataset[i].prescription)
        astigmatic.append(self.dataset[i].astigmatic)
        tearRate.append(self.dataset[i].tearRate)
        diabetic.append(self.dataset[i].diabetic)
        needLenses.append(self.dataset[i].needLense)
    listOFColumns = [age,prescription,astigmatic,tearRate,diabetic,needLenses]
    for i in range(0,5):
```

```
    for i in range(0,5):
        if self.features[i].visited == -1:
            numberOfZeros = 0
            numberOfOnes = 0
            listOFZeros = []
            listOFOnes = []
            length = len(listOFColumns[i])
            for j in range(0, length):
                if listOFColumns[i][j] == 0:
                    numberOfZeros += 1
                    listOFZeros.append(listOFColumns[5][j])
                elif listOFColumns[i][j] == 1:
                    numberOfOnes += 1
                    listOFOnes.append(listOFColumns[5][j])
            gain = self.entropy(listOFColumns[i]) - (((numberOfZeros / length) * self.entropy(listOFZeros)) +
                                                     ((numberOfOnes / length) * self.entropy(listOFOnes)))
            if gain > Max_Gain:
                Max_Gain = gain
                maxColumn = i
    self.features[maxColumn].visited = 1
    return maxColumn
```

```
startNode = None
currentNode = None
def buildtree(self,postion):

    maxGainColumn = self.Get_Max_Gain()

    node = Node(maxGainColumn)
    node.id = maxGainColumn
    if postion == 'Root':
        self.startNode = node
    elif postion == 'Left':
        self.currentNode.left = node
    elif postion == 'Right':
        self.currentNode.right = node

    self.currentNode = node
    datasetZeros = []
    datasetOnes = []
    tmpResultZeros = []
    tmpResultones = []

    if maxGainColumn == 0:
```

```
    if maxGainColumn == 0:
        for i in range(0, len(self.dataset)):
            if self.dataset[i].age == 0:
                datasetZeros.append(self.dataset[i])
                tmpResultZeros.append(self.dataset[i].needLense)
            elif self.dataset[i].age == 1:
                datasetOnes.append(self.dataset[i])
                tmpResultones.append(self.dataset[i].needLense)
    elif maxGainColumn == 1:
        for i in range(0, len(self.dataset)):
            if self.dataset[i].prescription == 0:
                datasetZeros.append(self.dataset[i])
                tmpResultZeros.append(self.dataset[i].needLense)
            elif self.dataset[i].prescription == 1:
                datasetOnes.append(self.dataset[i])
                tmpResultones.append(self.dataset[i].needLense)
    elif maxGainColumn == 2:
        for i in range(0, len(self.dataset)):
            if self.dataset[i].astigmatic == 0:
                datasetZeros.append(self.dataset[i])
                tmpResultZeros.append(self.dataset[i].needLense)
            elif self.dataset[i].astigmatic == 1:
                datasetOnes.append(self.dataset[i])
                tmpResultones.append(self.dataset[i].needLense)
```

```
    elif maxGainColumn == 3:
        for i in range(0, len(self.dataset)):
            if self.dataset[i].tearRate == 0:
                datasetZeros.append(self.dataset[i])
                tmpResultZeros.append(self.dataset[i].needLense)
            elif self.dataset[i].tearRate == 1:
                datasetOnes.append(self.dataset[i])
                tmpResultones.append(self.dataset[i].needLense)
    elif maxGainColumn == 4:
        for i in range(0, len(self.dataset)):
            if self.dataset[i].diabetic == 0:
                datasetZeros.append(self.dataset[i])
                tmpResultZeros.append(self.dataset[i].needLense)
            elif self.dataset[i].diabetic == 1:
                datasetOnes.append(self.dataset[i])
                tmpResultones.append(self.dataset[i].needLense)
```

```
    self.dataset.clear()
    tmpResult = np.unique(tmpResultZeros)
    if len(tmpResult) >1:
        self.dataset = datasetZeros.copy()
        self.buildtree('Left')
    else:
        self.currentNode.left = tmpResult[0]
    tmpResult = np.unique(tmpResultones)
    if len(tmpResult) > 1:
        for i in range(0,len(datasetOnes)):
            self.dataset = datasetOnes.copy()
        self.buildtree('Right')
    else:
        self.currentNode.right = tmpResult[0]

ef classify(self, input):
```

```
def classify(self, input):
    node = self.startNode
    while True:
        if node.id == 0:
            if input[0] == 1:
                if node.right == 0 or node.right == 1:
                    return node.right
                else:
                    node = node.right
            elif input[0] == 0:
                if node.left == 0 or node.left == 1:
                    return node.left
                else:
                    node = node.left
        elif node.id == 1:
            if input[1] == 1:
                if node.right == 0 or node.right == 1:
                    return node.right
                else:
                    node = node.right
            elif input[1] == 0:
                if node.left == 0 or node.left == 1:
                    return node.left
                else:
                    node = node.left
        elif node.id == 2:
```

```
    elif node.id == 2:
        if input[2] == 1:
            if node.right == 0 or node.right == 1:
                return node.right
            else:
                node = node.right
        elif input[2] == 0:
            if node.left == 0 or node.left == 1:
                return node.left
            else:
                node = node.left
    elif node.id == 3:
        if input[3] == 1:
            if node.right == 0 or node.right == 1:
                return node.right
            else:
                node = node.right
        elif input[3] == 0:
            if node.left == 0 or node.left == 1:
                return node.left
            else:
                node = node.left
    elif node.id == 4:
        if input[4] == 1:
            if node.right == 0 or node.right == 1:
                return node.right
            else:
                node = node.right
        elif input[4] == 0:
            if node.left == 0 or node.left == 1:
                return node.left
            else:
                node = node.left
```

3. Sample run (the output)

```
testcase 1:  1
testcase 2:  0
testcase 3:  0
testcase 4:  1
```

3. Discussion

ID3 algorithm:

- In class ID3 there are five main functions:
  1. The initialization function which take the features as parameter and then call the buildtree function to build the decision tree from the dataset.
  2. The Entropy function that take a column feature and calculate the entropy by counting the number of zeros and ones then calculate and return the result of this equation.

$$\text{Entropy} = -\frac{P}{Total}\log_2\frac{P}{Total} - -\frac{N}{Total}\log_2\frac{N}{Total}$$

  3. The Get_Max_Gain Function that calculate the Gain for each column feature on the dataset by counting the number of zeros and ones and calculating the entropy by using the previous function entropy then calculate the result of this equation.

$$\text{Gain} = \text{Entropy} - \sum_{i=0}^{All\ children\ of\ attribute}\frac{Pi+Ni}{Total}(child)$$

     then return the column number of the maximum gain.

4. The buildtree function that train on the dataset and make the decision tree by taking the position of the next node in the tree and by applying number of steps:
    - Get the column number of the maximum gain by calling the function Get_Max_Gain.
    - Initialize a node and add it in the tree according to the position send in the parameter either it is a root or a node that will be placed at the left of the previous node or to its right.
    - Take a copy of the data of the feature with the maximum gain then split the data according to the result of this copy.
    - Finally, check for the zeros sets and the ones if it is pure set all are zeros or ones or a further split will be done.
5. The classify function that takes an input array of features and determine the output one or zero according to the classification by following the decision tree starting from the root node to the required output.