

PA2

February 19, 2020

1 PA2 - ID3 Decision Tree

Zhanchong Deng

A15491777

Necessary Imports

```
[1]: %load_ext autoreload
      %autoreload 2
      import numpy as np
      import pandas as pd
      import ID3 as pa2
      from scipy.stats import entropy
      from scipy import stats
      import matplotlib.pyplot as plt
```

Import Data

```
[2]: training = pa2.loadData('pa2train.txt')
      validation = pa2.loadData('pa2validation.txt')
      test = pa2.loadData('pa2test.txt')
```

Training, without Pruning

```
[3]: tree = pa2.id3()
```

Created a new ID3 tree

```
[4]: %time tree.fit(training)
```

Wall time: 15.9 s

1) Visualizing resulted Tree

Below is my representation of the tree

Number of **tabs** indicates which level the node is at.

For non leaves: (label)Section ??

For leaves: (label)Section ??

```
[5]: # If root does not count as a level
      print(tree.printTreeAt(3))
```

```
(root)[is feature at 5 < 0.5?](num_data:2000)
  (yes)[is feature at 1 < 415000.0?](num_data:1319)
    (yes)[is feature at 17 < 2506.5?](num_data:1284)
    (no)[is feature at 21 < 208.0?](num_data:35)
  (no)[is feature at 5 < 1.5?](num_data:681)
    (yes)[is feature at 20 < 584.5?](num_data:292)
    (no)[is feature at 21 < 2006.0?](num_data:389)
```

2) Validation/Test Errors

Validation Error:

```
[6]: tree.error(validation)
```

```
[6]: 0.179
```

Test Error:

```
[7]: tree.error(test)
```

```
[7]: 0.173
```

3) Pruning decision tree with Greedy approach in BFS order

Prune 1 and 2 nodes with validation/test errors:

```
[8]: tree.pruneTree(validation, test, 2)
```

Pruned 1 time(s) with error:

Validation error: 0.122

Test error: 0.117

Pruned 2 time(s) with error:

Validation error: 0.107

Test error: 0.103

Tree after two pruned nodes:

```
[9]: print(tree.printTree())
```

```
(root)[is feature at 5 < 0.5?](num_data:2000)
  (yes)[0.0](num_data:1319)
```

```
(no)[1.0](num_data:681)
```

4) Most prominent feature:

The most prominent feature must be the feature selected as threshold at root.

```
[10]: features_name = open('pa2features.txt')
      features_name.seek(0)
      columns = features_name.read().split('\n')[:-1]
```

```
[11]: columns[tree.root.feature]
```

```
[11]: 'PAYMENT_DELAY_SEPTMBER'
```

5) Code from ID3.py:

```
'''
```

```
Contains all methods for a ID3 Decision Tree.
```

```
@Author: Zhanchong Deng
```

```
@Date: 2/15/2020
```

```
'''
```

```
import numpy as np
```

```
import pandas as pd
```

```
from scipy.stats import entropy
```

```
from scipy.stats import mode
```

```
'''
```

```
Return numpy arrays representing the dataset.
```

```
@param fp is the file path.
```

```
@return 2D numpy array, (#entry, 23).
```

```
'''
```

```
def loadData(fp):
```

```
    newfile = open(fp, 'r')
```

```
    newfile.seek(0)
```

```
    raw_strings = newfile.read().split("\n")[:-1]
```

```
    return np.array([np.array(entry[0:-1].split(" "), dtype="float") for entry in raw_strings])
```

```
'''
```

```
Node for Decision Tree
```

```
'''
```

```

class idNode():
    # how many data in this node
    numPoints = 0
    data = []
    # Branches
    yes = None
    no = None
    parent = None
    # Is is a leaf?
    isLeaf = False
    # How to get here
    route = "root"
    # For non Leaves
    threshold = -1
    feature = -1
    # For leaves
    rule = -1

    def pruneNode(self):
        self.yes = None
        self.no = None
        self.isLeaf = True
        self.rule = mode(self.data[:, -1])[0][0]

    def toString(self):
        if not self.isLeaf:
            return "(" + self.route + ")" + "[is feature at " + str(self.feature + 1) + " < " + str(self.threshold) + "?" + "(num_data:" + str(self.numPoints) + ")\n"
        else:
            return "(" + self.route + ")" + "[" + str(self.rule) + "](num_data:" + str(self.numPoints) + ")\n"

'''
Decision Tree Object.
'''

class id3():
    # Default Constructor
    def __init__(self):
        self.root = idNode()
        print("Created a new ID3 tree")

    '''
    Training methods
    '''

    def fit(self, training_data):

```

```

self.build(self.root, training_data)

def build(self, node, v):
    # Base case: Stop if it is pure
    if self.isPure(v):
        # Make it a leaf node.
        node.isLeaf = True
        node.rule = v[0][-1]
        node.numPoints = len(v)

    # Parse them according to H(entropy)
    else:
        node.numPoints = len(v)
        node.data = v
        # Pick a feature f and threshold t
        node.feature = 0
        minEntropy = float("inf")
        for thisfeature in range(0, len(v[0]) - 1):
            v_at_f = v[:, [thisfeature, -1]]
            list_of_threshold = self.generateThreshold(v_at_f)
            # This feature is not fit for splitting, pick another
            if len(list_of_threshold) == 0:
                continue
            # Set Threshold as the current best
            result = self.maxIG(list_of_threshold, v_at_f)
            if result[1] < minEntropy:
                minEntropy = result[1]
                node.feature = thisfeature
                node.threshold = result[0]

        # Split them according to the rule
        v_yes = v[v[:, node.feature] < node.threshold]
        v_no = v[v[:, node.feature] >= node.threshold]
        # Create new nodes
        node.yes = idNode()
        node.yes.route = "yes"
        node.yes.parent = node
        self.build(node.yes, v_yes)
        node.no = idNode()
        node.no.route = "no"
        node.no.parent = node
        self.build(node.no, v_no)

def isPure(self, vec):
    return len(np.unique(vec[:, -1])) == 1

def generateThreshold(self, feature):
    # Generate based solemnly on the feature vector
    distinct_val = np.sort(np.unique(feature[:, 0]))

```

```

    return (distinct_val[:-1] + distinct_val[1:]) / 2

def maxIG(self, list_of_threshold, v_at_f):
    # Initialize as the first threshold
    minthreshold = list_of_threshold[0]
    minEntropy = self.entropy_with_threshold(minthreshold, v_at_f)
    for i in range(1, len(list_of_threshold)):
        newEntropy = self.entropy_with_threshold(list_of_threshold[i], v_at_f)
        # update the threshold with the minimum entropy
        if minEntropy > newEntropy:
            minthreshold = list_of_threshold[i]
            minEntropy = newEntropy
    return (minthreshold, minEntropy)

def entropy_with_threshold(self, threshold, v_at_f):
    # Slice v according to the given threshold
    v_yes = v_at_f[v_at_f[:, 0] < threshold]
    v_no = v_at_f[v_at_f[:, 0] >= threshold]
    # Calculate H(X/Z=yes)
    margin_yes = [np.sum(v_yes[:, -1] == num_labels) / len(v_yes) for num_labels in np.unique(v_yes[:, -1])]
    h_yes = entropy(margin_yes)
    # Calculate H(X/Z=no)
    margin_no = [np.sum(v_no[:, -1] == num_labels) / len(v_no) for num_labels in np.unique(v_no[:, -1])]
    h_no = entropy(margin_no)
    # Calculate H(X/Z)
    return len(v_yes) / len(v_at_f) * h_yes + len(v_no) / len(v_at_f) * h_no

'''
    Validation methods
'''

def pruneTree(self, validation, test, maxprune):
    num_pruned = 0 # Specify how many prune can be done
    queue = [self.root] # For BFS
    while len(queue) > 0:
        # Calculate Current Error
        oldError = self.error(validation)
        # Use list as a queue
        curNode = queue.pop(0)
        # Create a deep copy
        original = idNode()
        original = self.copyNode(curNode)
        # Prune it
        curNode.pruneNode()
        # Compare errors
        newError = self.error(validation)
        # We screwed up, revert prune
        if oldError < newError:

```

```

        # Edge for root
        if curNode.parent is None:
            self.root = original
            original.yes.parent = self.root
            original.no.parent = self.root
        else:
            if curNode.route == "yes":
                curNode.parent.yes = original
            else:
                curNode.parent.no = original
        if not original.yes.isLeaf:
            queue.append(original.yes)
        if not original.no.isLeaf:
            queue.append(original.no)

        # We pruned new node
        else:
            num_pruned += 1 # Increment counter
            # Display new error and how many node pruned
            print("Pruned", num_pruned, "time(s) with error:\n\tValidation error:", newErr)
            self.error(test)
            # End pruning immediately if reached max
            if num_pruned >= maxprune:
                break

def copyNode(self, node):
    newNode = idNode()
    newNode.numPoints = node.numPoints
    newNode.data = np.copy(node.data)
    newNode.yes = node.yes
    newNode.no = node.no
    newNode.parent = node.parent
    newNode.isLeaf = node.isLeaf
    newNode.route = node.route
    newNode.threshold = node.threshold
    newNode.feature = node.feature
    newNode.rule = node.rule
    return newNode

'''
    Testing methods
'''

def predict(self, features):
    return np.apply_along_axis(self.predictOne, 1, features)

```

```

def error(self, test_data):
    return np.mean(self.predict(test_data[:, :-1]) != test_data[:, -1])

def predictOne(self, v):
    node = self.root
    while not node.isLeaf:
        # Yes branch
        if v[node.feature] < node.threshold:
            node = node.yes
        else:
            node = node.no
    # This should never happen
    if node.rule == -1:
        print("There is something wrong with your tree")
    return node.rule

'''
    Visualization methods.
'''

def printTree(self):
    tree_str = self.printTreeR(self.root, 0, float("inf"))
    return tree_str

def printTreeAt(self, maxlevel):
    return self.printTreeR(self.root, 0, maxlevel)

def printTreeR(self, curNode, level, maxlevel):
    # do yourself
    curStr = '\t' * level + curNode.toString()
    # do yes
    if not curNode.yes.isLeaf and level + 1 < maxlevel - 1:
        curStr += self.printTreeR(curNode.yes, level + 1, maxlevel)
    else:
        curStr += '\t' * (level + 1) + curNode.yes.toString()
    # do no
    if not curNode.no.isLeaf and level + 1 < maxlevel - 1:
        curStr += self.printTreeR(curNode.no, level + 1, maxlevel)
    else:
        curStr += '\t' * (level + 1) + curNode.no.toString()
    return curStr

```