Project Three:
Decision Tree

Allison Bolen
CIS 678
Wolffe
Winter 2018

Goals:

Decision trees are a form of supervised machine learning. Given a testing set a decision tree can classify the class of a data instance based on a simple series of classified attributes.In this specific experiment we built a decision tree generator using Python 3, and Jupyter Notebooks. Decision trees function almost like a game of 20 questions, they narrow down results based on certain questions. For example a decision tree could classify whether a given day is a good day to fish based on the tree shown in figure one.

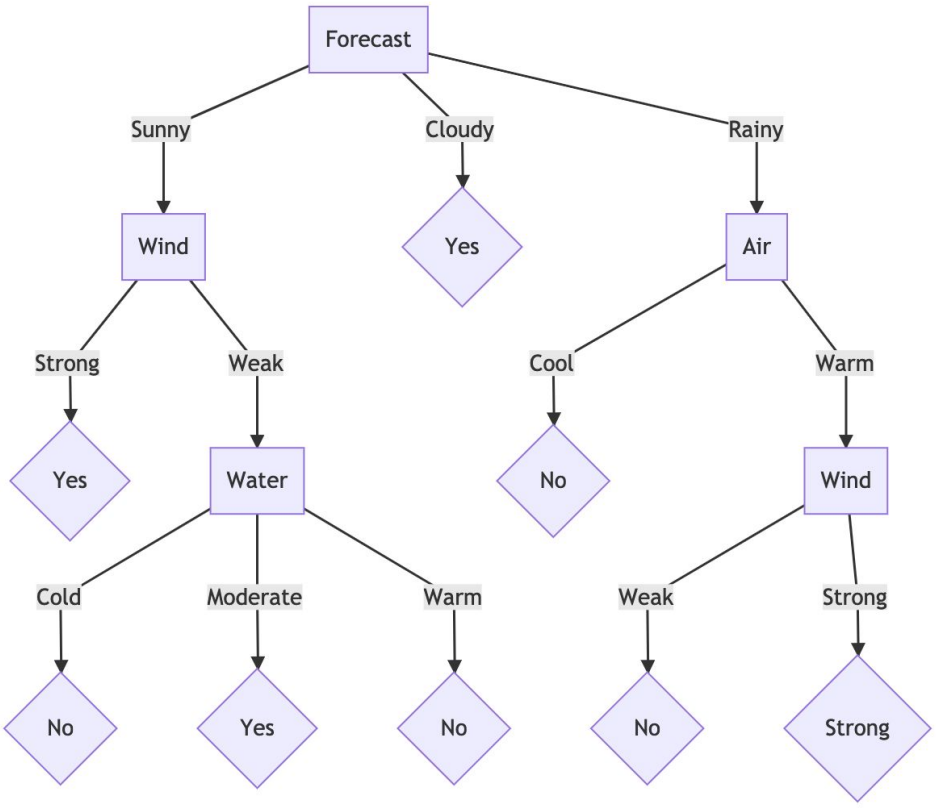

Figure One: decision tree representation of a training data set for fishing.

Given an example data instance like [Wind: Weak, Water: Cold, Air: Cool, Forecast: Rainy] and traveling down the decision tree visiting the relevant nodes the result would be "No". The path would be: node:Forecast>edge:Rainy>node:Air>edge:Cool>>No. One could say it's almost like having a conversation: Jerry asks "What's the forecast for today?" Nithin answers with "It is going to be Sunny." Jerry then asks "Whats the wind like?" and Nithin says "It is strong." Jerry can conclude that today he should go fishing. Using a testing and training dataset we will explain how a decision tree can be made and provide some examples of classification and visuals.

Preprocessing

For the preprocessing step before generating the trees we decided to reshape the data files. We originally had a description section prepending the data section, we just decided to extract the description to a new file and add headers to the remaining data file so it could be loaded as a pandas frame in one call. We then read in the description file and stored the data as a python dictionary. The data we wanted to keep track of is what the class is. In the above case the data we tracked was:

```
{'classInfo': {'num': 2, 'values': ['Yes', 'No']},
 'attributeInfo': {'num': 4,
  'Wind': {'num': '2', 'values': ['Strong', 'Weak']},
  'Water': {'num': '3', 'values': ['Warm', 'Moderate', 'Cold']},
  'Air': {'num': '2', 'values': ['Warm', 'Cool']},
  'Forecast': {'num': '3', 'values': ['Sunny', 'Cloudy', 'Rainy']}},
 'total': 14}
```

We tracked the number of class values, attributes, total instances, and number of values of attributes.

Algorithm

The steps to create a decision tree are computationally repetitive, but overall not extremely intensive (although the more attributes you have in the training set, the more levels of repetition are needed). Decision trees are made by selecting nodes to branch off of by calculating values for purity. You can do this by calculating entropy and gain, or gini index and gain for the set attribute values given the class values.

The equation for entropy is shown in figure two.

$$\text{Entropy}(S) \equiv -\sum_{i=1}^{k} p_i \log_2 p_i$$

Figure Two: Entropy equation.

Where *S* is the collection of examples, *i* is the number of categories, and *p* is the ratio of the cardinality of category *i* to the cardinality of *S*, as in $p_i = N_i / N$. Entropy is the calculation of purity at a node in the set. Though using entropy seems to create a tree with a bias towards attributes with more features.

The equation for the gini index is shown in figure three.

$$Gini\ Index(S) = 1 - \sum_{i=1}^{k} p_i^2$$

Figure Three: Gini index equation.

Where *S* is the collection of examples, *i* is the number of categories and *p* is the ratio of the cardinality of category *i* to the cardinality of *S.* The gini index is the expected error if a randomly selected instance is randomly labeled given the distribution of labels in the subset. Gini makes the tree insensitive to the distribution of the data.

While entropy shows the level of purity at that instance gain shows the expected reduction in impurity given a certain choice of attribute. We always want to maximize gain, since we want nodes with the most purity. The equation for gain is shown in figure four.

$$\text{Gain}(S, a) = \text{Entropy}(S) - \sum_{v=\text{values}(a)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

Figure Four: Gain equation.

Where values(*a*) is the set of all possible values for attribute *a*, and S_V is the subset of set *S* for which attribute *a* has value *v*. You can replace the entropy calculation in gain with the gini calculation.

We based our coded solution on the ID3 algorithm. The ID3 algorithm is a decision tree classifier. Here is the basics of how it works:

- if all examples in *S* are of the same class return a leaf with that class label
- else if there are no more attributes to test return a leaf with the most common class label
- else choose the attribute a that maximizes the Information Gain of S let attribute a be the decision for the current node add a branch from the current node for each possible value v of attribute a
    - for each branch
        - “sort” examples down the branches based on their value v of attribute a recursively call ID3(Sv) on the set of examples in each branch

Overall decision trees are execution efficient with generation at O(N logN) and classification at O(logN). They are easily interpretable because they flow much like if-then questions and are easy to visualize. They are the favored machine learning solution in the medical field because you can directly see how a result was reached (when it comes to medicine you don't want any mystery).

Visual

We generated trees for the fishing (Figure One), contacts, and cars dataset, however the cars dataset visual tree can not be properly shown in its visual form so a json representation

has been included at the end of the text. The contacts tree is shown in figure five with a root node of “tear-rate”. We used a JavaScript tool called MermaidJs to build the visual representations.

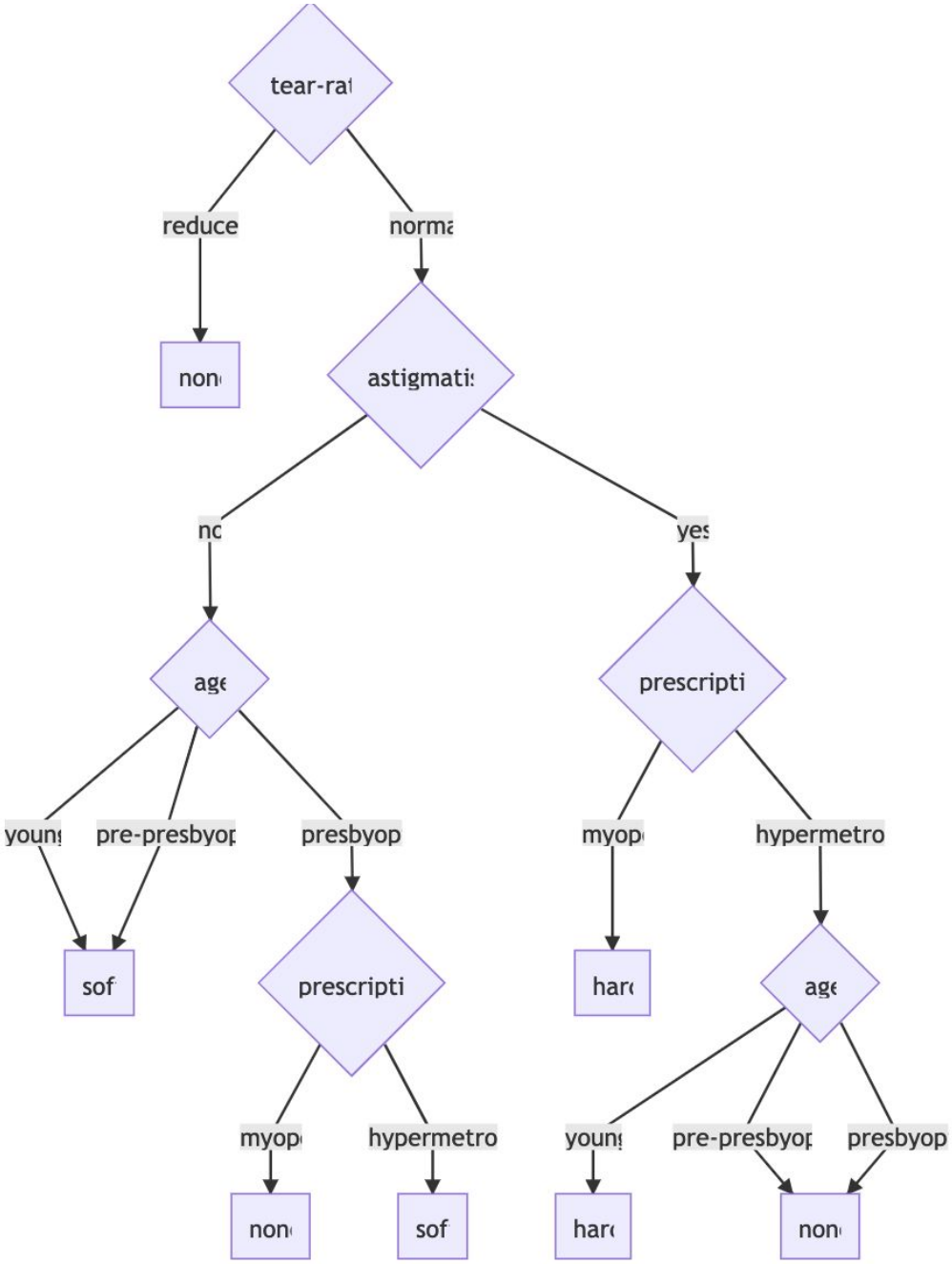

Figure Five: The Contacts data set tree.

Analysis

When we read in a car data set testing file we generated a tree using the calculation combination of entropy and gain. Using this tree we were able to classify the cars testing data

set with 89% correctness where our predictions matched the actual class value of the data instance. Using our implementation of the ID3 algorithm it is possible to get fairly decent results when classifying with the generated trees.

Initially I experienced some problems with instances in the test set that were not seen before. If the training set didn't have any data for an instance then the terminal node for that testing instances would have been blank. It turns out that originally we were not accounting for the second if statement in the ID3 algorithm. This was causing a result of 87% correctness. We solved this and we were able to get to 89% by adding in that step.

Improvements

Some improvements to our algorithm could be using the gain calculation with gini instead. We could have also pruned some of the nodes as well. Pruning is the process of shortening the branches on the tree. For example our tree calculates branches until it hits absolute purity on a node. However, you could stop at a branch above and take the result majority and set that as the terminal node. In the contacts tree in figure five we see on the no branch of the astigmatism node that 2 of the three terminal points are soft with trees that have lots of options and depth you can simplify by dropping the deeper branches and setting, in this example the, node at the end of the edge “no” to soft. Granted you are risking misclassifying some instances at that point, but that can be managed if you account for the total number of instances at each terminal node that matched that flow. If a lot of instances are mapped to the flow of astigmatism>no>age>presbyopic>myope>hard then you wouldn't want to prune that branch out.

Another way to improve this would be to implement a random forest algorithm. This is a process of creating trees from randomly chosen instances in the training set you would then classify against each of these trees and select the mode of the classification these trees return. The mode result is the classification for the testing instance. This approach is relatively time consuming but you can get slightly better results based on a consensus from trees trained on different subsets.

Program Design

You can check my Github repository for direct examples and to test the code. There is also an image file for the cars tree it is just not very readable.

Preprocessing:

```
# Allison Bolen
# win 2019
# cis678
# Wolffe

import sys
```

```
import pandas as pd
import pickle, os


# In[71]:

def main():
    '''
        This is the main that converts teh data files into an info
dictinary and a data frame
        it saves tehse for future use
    '''
    # # fish
    # dataFrame = pd.read_csv("../DataFiles/fishData/fishing.csv",
header=0, sep=',')
    # dataFrame.head()
    # saveFrame(dataFrame, "../DataFiles/fishData/processedFishData")
    # info = readFile("../DataFiles/fishData/fishInfo.txt")
    # save = "../DataFiles/fishData/fishCounts.pkl"

    # # contacts
    # dataFrame =
pd.read_csv("../DataFiles/contactData/contact-lenses.csv", header=0,
sep=',')
    # dataFrame.head()
    # saveFrame(dataFrame, "../DataFiles/contactData/contactData")
    # info = readFile("../DataFiles/contactData/contactInfo.txt")
    # save = "../DataFiles/carData/contactCounts.pkl"

    # cars
    dataFrame = pd.read_csv("../DataFiles/carData/car_training.csv",
header=0, sep=',')
    dataFrame.head()
    saveFrame(dataFrame, "../DataFiles/carData/carData")
    info = readFile("../DataFiles/carData/cartraininginfo.txt")
    save = "../DataFiles/carData/carCounts.pkl"

    print("Saving " + save)
    save_it_all(prep(info), save)

# defaults:
def save_it_all(obj, filename):
```

```
    os.makedirs(os.path.dirname(filename), exist_ok=True)
    with open(filename, 'wb') as output:  # Overwrites any existing file.
        pickle.dump(obj, output, protocol=2)

def load_objects(file):
    with open(file, 'rb') as input:
        return pickle.load(input)

def saveFrame(df, name):
    df.to_csv(name+".csv", index=False, sep=",", header=True)
    save_it_all(df, name+".pkl")

# read info file:
def readFile(fname):
    with open(fname) as f:
        content = f.readlines()
    # you may also want to remove whitespace characters like `\n` at the
end of each line
    content = [x.strip() for x in content]
    return content

def RepresentsInt(s):
    try:
        int(s)
        return True
    except ValueError:
        return False

def prep(info):
    """
        This gets the info for the data set organized and in a dictionray
    """
    infoDict = {}
    for index in range(0, len(info)):
        if index == 0:
            # class values
            infoDict["classInfo"] = {"num": int(info[index]), "values":
info[1].split(",")}

        if index == 2:
            infoDict["attributeInfo"] = {"num":int(info[index])}
            for attributeIndex in range(index+1, index+1+int(info[index])):
```

```
                attribute = info[attributeIndex].split(",")[0]
                numValues = info[attributeIndex].split(",")[1]
                values = info[attributeIndex].split(",")[2:]
                if RepresentsInt(values[0]):
                    values = list(map(int, values))
                infoDict["attributeInfo"][attribute] = {"num":numValues,
"values":values}
        if index == len(info)-1:
            infoDict["total"] = int(info[index])

    return infoDict

if __name__ == "__main__": main()
```

Tree.py:

```
# Allison Bolen
# win 2019
# cis678
# Wolffe

import pandas as pd
import pickle, os, math
from pprint import pprint

def main():

    # #fish
    # baseInfoDict = load_objects("DataFiles/fishData/fishCounts.pkl")
    # dataframe = load_objects("DataFiles/fishData/processedFishData.pkl")
    # saveFile = "DataFiles/fishData/fishTree.pkl"

    # # contacts
    # baseInfoDict =
load_objects("DataFiles/contactData/contactCounts.pkl")
    # dataframe = load_objects("DataFiles/contactData/contactData.pkl")
    # saveFile = "DataFiles/contactData/contactTree.pkl"

    # cars
    baseInfoDict = load_objects("../DataFiles/carData/carCounts.pkl")
    dataframe = load_objects("../DataFiles/carData/carData.pkl")
    saveFile = "../DataFiles/carData/carTree.pkl"
```

```
    # # hw2
    # baseInfoDict = load_objects("DataFiles/hw2set/trainCounts.pkl")
    # dataframe = load_objects("DataFiles/hw2set/train.pkl")
    # saveFile = "DataFiles/hw2set/trainTree.pkl"

    # print(dataframe)

    tree = makeTree(baseInfoDict, dataframe, "")
    save_it_all(tree, saveFile)

# defaults:
def save_it_all(obj, filename):
    os.makedirs(os.path.dirname(filename), exist_ok=True)
    with open(filename, 'wb') as output:  # Overwrites any existing file.
        pickle.dump(obj, output, pickle.HIGHEST_PROTOCOL)

def load_objects(file):
    with open(file, 'rb') as input:
        return pickle.load(input)

def saveFrame(df, name):
    df.to_csv(name+".csv", index=False, sep=",", header=True)
    save_it_all(df, name+".pkl")

# tree methods:
def setInfo(dataFrame, infoDict):
    '''
    This sets up the overall data of the dataframe the summary of the
entire set
    '''
    roundXDict = {"setInfo":{}}
    for classValue in infoDict["classInfo"]["values"]:
        num =
len(dataFrame["Class"][dataFrame["Class"]==classValue].tolist())
        roundXDict["setInfo"][classValue] = num/infoDict["total"]

    purity = []
    for key, value in roundXDict["setInfo"].items():
        if value == 0 or value == 1:
            purity.append(0)
        else:
```

```
            purity.append(-(value)*math.log(value,2))


    e = sum(purity)

    roundXDict["setInfo"]["purity"] = e
    return roundXDict

def entropy(val):
    return -(val)*math.log(val,2)

def gini(val):
    return val**2

def attributeInfo(dataFrame, infoDict, roundX):
    '''
    This populates a dictionary full of the data frame info
    this caclulates the number of instaces for each branch attribute
    the purity for each branch attribute
    and the class percentage
    '''
    roundX["attributes"] = {}
    # make dictionary
    for attribute in infoDict["attributeInfo"]:
        if attribute != "num":
            roundX["attributes"][attribute] = {"attrTypes":{}}
            for attributeValue in
infoDict["attributeInfo"][attribute]["values"]:

roundX["attributes"][attribute]["attrTypes"][attributeValue] =
{"values":{"numInstance":0}}

    # do data gathering
    for classValue in infoDict["classInfo"]["values"]:
        for attribute in infoDict["attributeInfo"]:
            if attribute != "num":
                for attributeValue in
infoDict["attributeInfo"][attribute]["values"]:
                    classFrame = dataFrame[dataFrame["Class"] ==
classValue]
                    numAttribute =
len(dataFrame[dataFrame[attribute]==attributeValue])
```

```
                    if numAttribute != 0:
                        numAttributeWithClass =
len(classFrame[classFrame[attribute] == attributeValue])

roundX["attributes"][attribute]["attrTypes"][attributeValue]["values"]["num
Instance"] += numAttributeWithClass

roundX["attributes"][attribute]["attrTypes"][attributeValue]["values"][clas
sValue] = numAttributeWithClass/numAttribute

roundX["attributes"][attribute]["attrTypes"][attributeValue]["values"]["pur
ity"] = None
                    else:

roundX["attributes"][attribute]["attrTypes"][attributeValue]["values"]["num
Instance"] += 0 # possible bug

roundX["attributes"][attribute]["attrTypes"][attributeValue]["values"][clas
sValue] = 0

roundX["attributes"][attribute]["attrTypes"][attributeValue]["values"]["pur
ity"] = None

    # setup purity
    for attribute in roundX["attributes"]:
        for key, value in
roundX["attributes"][attribute]["attrTypes"].items():
                p = []
                for subkey, subvalue in value["values"].items():
                    if subkey != "purity" and subkey != "numInstance":
                        if subvalue == 0 or subvalue == 1:
                            p.append(0)
                        else:
                            # change for gini or entropy
                            p.append(entropy(subvalue))
                    purity = sum(p)

roundX["attributes"][attribute]["attrTypes"][key]["values"]["purity"] =
purity

    return roundX
```

```
def gainCheck(dataFrame, infoDict, roundXDict):
    '''
        This function calculates the gain for each branch
    '''
    for attribute, value in roundXDict["attributes"].items():
        gain = []
        for types, vals in value['attrTypes'].items():
            typeInfo = vals["values"]
            # set purity  - sum((typecount/classcount)*purityoftype + ...)
=

gain.append((typeInfo["numInstance"]/infoDict["total"]*typeInfo["purity"]))

        finalGain = roundXDict["setInfo"]["purity"] - sum(gain)
        roundXDict["attributes"][attribute]["gain"] = finalGain
    return roundXDict

def setUpMax(roundX):
    """
    Get the maximum gain value of the set and return that as the new branch
value
    """
    vals = []
    for attribute in roundX["attributes"]:
        vals.append(roundX["attributes"][attribute]["gain"])
    branch = []
    for attribute in roundX["attributes"]:
        if max(vals) == roundX["attributes"][attribute]["gain"]:
            branch.append(attribute)
    return branch

import collections
def findMajority(roundX, branch):
    """
    For attributes that dont have any existance in the test set please give
them the majority value at the current branch
    """
    majority = []
    for attributeType, value in
roundX["attributes"][branch[0]]["attrTypes"].items():
        # print("\n"+tabs+str(attributeType).upper())
        if value["values"]["purity"] == 0 and
```

```
value["values"]["numInstance"] != 0: # we are at a termianl node, this node
is pure
            # check the class values to get the class node value
            for key, val in value["values"].items():
                if key != "purity" and key != "numInstance": # for class
values only
                    if val != 0:
                        majority.append(key)
    counter = collections.Counter(majority)
    return counter.most_common()[0][0]

def getBranch(infoDict, dataframe):
    '''
        Go through teh steps of getting the branch of the current set
    '''
    roundX = setInfo(dataframe, infoDict)
    roundX = attributeInfo(dataframe, infoDict, roundX)
    roundX = gainCheck(dataframe, infoDict, roundX)
    branch = setUpMax(roundX)
    return branch , roundX

def makeTree(infoDict, dataframe, tabs):
    '''
    This makes the tree for the data set
    it is recursive
    '''
    # return the next branch level
    branch, roundX = getBranch(infoDict, dataframe)

    tree = {branch[0]:{}}
    print(tabs+"This is the next branch: "+branch[0])
    if len(branch) > 1:
        print("YOU HAVE MORE THAN ONE CHOICE FOR THIS BRANCH")

    for attributeType, value in
roundX["attributes"][branch[0]]["attrTypes"].items():
        tree[branch[0]][attributeType]= {}
        if value["values"]["purity"] == 0 and
value["values"]["numInstance"] != 0: # we are at a termianl node, this node
is pure
            # check the class values to get the class node value
            for key, val in value["values"].items():
```

```
                if key != "purity" and key != "numInstance": # for class
values only
                    if val != 0:
                        # get the terminal leaf
                        print(tabs+"Terminal for branch " + branch[0] +" at
value "+ str(attributeType) + " : "+key)
                        tree[branch[0]][attributeType] = key
        elif value["values"]["numInstance"] == 0:
            # if you dont have any occrances in the trainin set youll just
be the majority result at this node
            majority = findMajority(roundX, branch)
            tree[branch[0]][attributeType] = findMajority(roundX, branch)
            print(tabs+"Terminal for branch " + branch[0] +" at value "+
str(attributeType) + " : "+majority)
        else:
            # look for the next branch level
            # edit dataframe
            dataFrameEdit = dataframe
            dataFrameUse =
dataFrameEdit[dataFrameEdit[branch[0]]==attributeType]
            dataFrameUse = dataFrameUse.drop(columns=[branch[0]])
            # change info dictionary
            infoDictEdit = infoDict.copy()
            infoDictEdit["attributeInfo"] =
infoDict["attributeInfo"].copy()
            del infoDictEdit["attributeInfo"][branch[0]]
            infoDictEdit["attributeInfo"]["num"] -= 1

            infoDictEdit["total"] = len(dataFrameUse.index)
            print(tabs+"New round branching off of "+ branch[0] + " at " +
str(attributeType))
            # call recursive
            tree[branch[0]][attributeType] = makeTree(infoDictEdit,
dataFrameUse, tabs+"\t")

    return tree

if __name__ == "__main__": main()
```

Classify:

```
# Allison Bolen
```

```
# win 2019
# cis678
# Wolffe

import pandas as pd
import pickle, os, math
from pprint import pprint

# defaults:
def save_it_all(obj, filename):
    os.makedirs(os.path.dirname(filename), exist_ok=True)
    with open(filename, 'wb') as output:  # Overwrites any existing file.
        pickle.dump(obj, output, pickle.HIGHEST_PROTOCOL)

def load_objects(file):
    with open(file, 'rb') as input:
        return pickle.load(input)

def saveFrame(df, name):
    df.to_csv(name+".csv", index=False, sep=",", header=True)
    save_it_all(df, name+".pkl")

def classify(instance, tree):
    '''
    This takes in an instance of data for classification and a tree to
classify against
    this method is recursive
    and returns the resulting instance predication
    '''
    result = None
    for k,v in tree.items():
        edge = instance[k]
        if type(tree[k][edge]) is dict:
            t = tree.copy()
            t = t[k][edge]
            result = classify(instance, t)
        else:
            return tree[k][edge]
    return result

def main():
    # #fish
```

```
    # tree = load_objects("DataFiles/fishData/fishTree.pkl")
    # dataframe = pd.read_csv("DataFiles/fishData/processedFishData.pkl")

    # # contacts
    # tree = load_objects("DataFiles/contactData/contactTree.pkl")
    # dataframe = pd.read_csv("DataFiles/contactData/contactData.pkl")

    # cars
    tree = load_objects("../DataFiles/carData/carTreeGini.pkl")
    dataframe = pd.read_csv("../DataFiles/carData/car_test.csv")

    # # hw2
    # test = load_objects("DataFiles/hw2set/trainTree.pkl")
    # dataframe = pd.read_csv("DataFiles/hw2set/test.csv")

    results = []
    count = 0
    for i in range(0,len(dataframe.index)):
        dataframeNoClass = dataframe.drop(columns=["Class"])
        prediction = classify(dataframeNoClass.iloc[i, : ], tree)
        results.append(prediction)
        try:
            if prediction == dataframe.iloc[i, : ]["Class"]:
                count = count + 1
            else:
                # print("-------WRONG--------")
                print(str(prediction)+"!")
                i = None
        except KeyError as e:
            print("Key Error")
    print()
    print("Right predictions: " + str(count) + ".\nWrong Predictions:
"+str(len(dataframe.index)-count)+".\nOut of " + str(len(dataframe.index))
+ " total intances")

if __name__ == "__main__": main()
```

CarsTreeJson:

```
{
   "safety":{
```

```
"low":"poor",
"medium":{
   "persons":{
      "2":"poor",
      "4":{
         "maintenance":{
            "vhigh":{
               "cost":{
                  "vhigh":"poor",
                  "high":"poor",
                  "medium":{
                     "trunk":{
                        "small":"poor",
                        "medium":{
                           "doors":{
                              "2":"poor",
                              "3":"poor",
                              "4":"acceptable",
                              "5":"acceptable"
                           }
                        },
                        "big":"acceptable"
                     }
                  },
                  "low":{
                     "trunk":{
                        "small":"poor",
                        "medium":{
                           "doors":{
                              "2":"acceptable",
                              "3":"poor",
                              "4":"acceptable",
                              "5":"acceptable"
                           }
                        },
                        "big":"acceptable"
                     }
                  }
               }
            },
            "high":{
               "cost":{
```

```
      "vhigh":"poor",
      "high":{
         "trunk":{
            "small":"poor",
            "medium":"acceptable",
            "big":"acceptable"
         }
      },
      "medium":{
         "trunk":{
            "small":"poor",
            "medium":"poor",
            "big":"acceptable"
         }
      },
      "low":"acceptable"
   }
},
"medium":{
   "cost":{
      "vhigh":{
         "trunk":{
            "small":"poor",
            "medium":"poor",
            "big":"acceptable"
         }
      },
      "high":{
         "trunk":{
            "small":"poor",
            "medium":{
               "doors":{
                  "2":"poor",
                  "3":"poor",
                  "4":"acceptable",
                  "5":"acceptable"
               }
            },
            "big":"acceptable"
         }
      },
      "medium":"acceptable",
```

```
      "low":{
         "trunk":{
            "small":"acceptable",
            "medium":"acceptable",
            "big":"good"
         }
      }
   }
},
"low":{
   "cost":{
      "vhigh":{
         "trunk":{
            "small":"poor",
            "medium":{
               "doors":{
                  "2":"poor",
                  "3":"poor",
                  "4":"poor",
                  "5":"acceptable"
               }
            },
            "big":"acceptable"
         }
      },
      "high":{
         "trunk":{
            "small":"poor",
            "medium":{
               "doors":{
                  "2":"acceptable",
                  "3":"poor",
                  "4":"acceptable",
                  "5":"acceptable"
               }
            },
            "big":"acceptable"
         }
      },
      "medium":{
         "trunk":{
            "small":"acceptable",
```

```
                  "medium":{
                     "doors":{
                        "2":"acceptable",
                        "3":"acceptable",
                        "4":"good",
                        "5":"acceptable"
                     }
                  },
                  "big":"good"
               }
            },
            "low":{
               "trunk":{
                  "small":"acceptable",
                  "medium":"acceptable",
                  "big":"good"
               }
            }
         }
      }
   },
"6":{
   "maintenance":{
      "vhigh":{
         "cost":{
            "vhigh":"poor",
            "high":"poor",
            "medium":{
               "trunk":{
                  "small":"poor",
                  "medium":"acceptable",
                  "big":"acceptable"
               }
            },
            "low":{
               "trunk":{
                  "small":"poor",
                  "medium":{
                     "doors":{
                        "2":"poor",
                        "3":"acceptable",
```

```
                  "4":"acceptable",
                  "5":"acceptable"
               }
            },
            "big":"acceptable"
         }
      }
   }
},
"high":{
   "cost":{
      "vhigh":"poor",
      "high":{
         "trunk":{
            "small":"poor",
            "medium":{
               "doors":{
                  "2":"poor",
                  "3":"acceptable",
                  "4":"acceptable",
                  "5":"acceptable"
               }
            },
            "big":"acceptable"
         }
      },
      "medium":{
         "trunk":{
            "small":"poor",
            "medium":{
               "doors":{
                  "2":"poor",
                  "3":"acceptable",
                  "4":"acceptable",
                  "5":"acceptable"
               }
            },
            "big":"acceptable"
         }
      },
      "low":{
         "doors":{
```

```
                "2":"poor",
                "3":"acceptable",
                "4":"acceptable",
                "5":"acceptable"
             }
          }
       }
    },
    "medium":{
       "trunk":{
          "small":{
             "cost":{
                "vhigh":"poor",
                "high":"poor",
                "medium":{
                   "doors":{
                      "2":"poor",
                      "3":"poor",
                      "4":"acceptable",
                      "5":"poor"
                   }
                },
                "low":"acceptable"
             }
          },
          "medium":{
             "cost":{
                "vhigh":"acceptable",
                "high":"acceptable",
                "medium":"acceptable",
                "low":"good"
             }
          },
          "big":{
             "cost":{
                "vhigh":"acceptable",
                "high":"acceptable",
                "medium":"acceptable",
                "low":"good"
             }
          }
       }
```

```
},
"low":{
   "cost":{
      "vhigh":{
         "trunk":{
            "small":"poor",
            "medium":{
               "doors":{
                  "2":"poor",
                  "3":"acceptable",
                  "4":"poor",
                  "5":"poor"
               }
            },
            "big":"acceptable"
         }
      },
      "high":{
         "trunk":{
            "small":"poor",
            "medium":{
               "doors":{
                  "2":"poor",
                  "3":"acceptable",
                  "4":"acceptable",
                  "5":"acceptable"
               }
            },
            "big":"acceptable"
         }
      },
      "medium":{
         "trunk":{
            "small":"acceptable",
            "medium":"good",
            "big":"good"
         }
      },
      "low":{
         "trunk":{
            "small":{
               "doors":{
```

```
                              "2":"poor",
                              "3":"acceptable",
                              "4":"acceptable",
                              "5":"acceptable"
                           }
                        },
                        "medium":{
                           "doors":{
                              "2":"acceptable",
                              "3":"acceptable",
                              "4":"acceptable",
                              "5":"good"
                           }
                        },
                        "big":"good"
                     }
                  }
               }
            }
         }
      }
   },
"high":{
   "persons":{
      "2":"poor",
      "4":{
         "cost":{
            "vhigh":{
               "maintenance":{
                  "vhigh":"poor",
                  "high":"poor",
                  "medium":"acceptable",
                  "low":"acceptable"
               }
            },
            "high":{
               "maintenance":{
                  "vhigh":"poor",
                  "high":"acceptable",
                  "medium":"acceptable",
                  "low":"acceptable"
```

```
      }
   },
   "medium":{
      "maintenance":{
         "vhigh":"acceptable",
         "high":"acceptable",
         "medium":{
            "doors":{
               "2":"acceptable",
               "3":"acceptable",
               "4":"vgood",
               "5":"vgood"
            }
         },
         "low":{
            "trunk":{
               "small":"good",
               "medium":"vgood",
               "big":"vgood"
            }
         }
      }
   },
   "low":{
      "maintenance":{
         "vhigh":"acceptable",
         "high":{
            "trunk":{
               "small":"acceptable",
               "medium":{
                  "doors":{
                     "2":"acceptable",
                     "3":"acceptable",
                     "4":"acceptable",
                     "5":"vgood"
                  }
               },
               "big":"vgood"
            }
         },
         "medium":{
            "trunk":{
```

```
                  "small":"good",
                  "medium":{
                     "doors":{
                        "2":"good",
                        "3":"good",
                        "4":"good",
                        "5":"vgood"
                     }
                  },
                  "big":"vgood"
               }
            },
            "low":{
               "trunk":{
                  "small":"good",
                  "medium":{
                     "doors":{
                        "2":"good",
                        "3":"vgood",
                        "4":"vgood",
                        "5":"vgood"
                     }
                  },
                  "big":"vgood"
               }
            }
         }
      }
   },
   "6":{
      "cost":{
         "vhigh":{
            "maintenance":{
               "vhigh":"poor",
               "high":"poor",
               "medium":{
                  "doors":{
                     "2":{
                        "trunk":{
                           "small":"poor",
                           "medium":"acceptable",
```

```
                  "big":"acceptable"
               }
            },
            "3":"acceptable",
            "4":"acceptable",
            "5":"acceptable"
         }
      },
      "low":{
         "doors":{
            "2":{
               "trunk":{
                  "small":"poor",
                  "medium":"acceptable",
                  "big":"acceptable"
               }
            },
            "3":"acceptable",
            "4":"acceptable",
            "5":"acceptable"
         }
      }
   }
},
"high":{
   "maintenance":{
      "vhigh":"poor",
      "high":{
         "doors":{
            "2":{
               "trunk":{
                  "small":"poor",
                  "medium":"acceptable",
                  "big":"poor"
               }
            },
            "3":"acceptable",
            "4":"acceptable",
            "5":"acceptable"
         }
      },
      "medium":{
```

```
            "doors":{
               "2":{
                  "trunk":{
                     "small":"poor",
                     "medium":"acceptable",
                     "big":"acceptable"
                  }
               },
               "3":"acceptable",
               "4":"acceptable",
               "5":"acceptable"
            }
         },
         "low":{
            "doors":{
               "2":"poor",
               "3":"acceptable",
               "4":"acceptable",
               "5":"acceptable"
            }
         }
      }
   },
   "medium":{
      "maintenance":{
         "vhigh":{
            "doors":{
               "2":{
                  "trunk":{
                     "small":"poor",
                     "medium":"poor",
                     "big":"acceptable"
                  }
               },
               "3":"acceptable",
               "4":"acceptable",
               "5":"acceptable"
            }
         },
         "high":{
            "doors":{
               "2":{
```

```
                  "trunk":{
                     "small":"poor",
                     "medium":"acceptable",
                     "big":"acceptable"
                  }
               },
               "3":"acceptable",
               "4":"acceptable",
               "5":"acceptable"
            }
         },
         "medium":{
            "trunk":{
               "small":{
                  "doors":{
                     "2":"poor",
                     "3":"acceptable",
                     "4":"acceptable",
                     "5":"acceptable"
                  }
               },
               "medium":"vgood",
               "big":"vgood"
            }
         },
         "low":{
            "trunk":{
               "small":{
                  "doors":{
                     "2":"poor",
                     "3":"good",
                     "4":"good",
                     "5":"good"
                  }
               },
               "medium":"vgood",
               "big":"vgood"
            }
         }
      }
   },
   "low":{
```

```
"maintenance":{
   "vhigh":{
      "doors":{
         "2":{
            "trunk":{
               "small":"poor",
               "medium":"acceptable",
               "big":"poor"
            }
         },
         "3":"acceptable",
         "4":"acceptable",
         "5":"acceptable"
      }
   },
   "high":{
      "trunk":{
         "small":{
            "doors":{
               "2":"poor",
               "3":"acceptable",
               "4":"acceptable",
               "5":"acceptable"
            }
         },
         "medium":{
            "doors":{
               "2":"acceptable",
               "3":"vgood",
               "4":"vgood",
               "5":"vgood"
            }
         },
         "big":"vgood"
      }
   },
   "medium":{
      "doors":{
         "2":"good",
         "3":"vgood",
         "4":"good",
         "5":{
```

```
                        "trunk":{
                           "small":"good",
                           "medium":"vgood",
                           "big":"vgood"
                        }
                     }
                  }
               },
               "low":{
                  "doors":{
                     "2":{
                        "trunk":{
                           "small":"poor",
                           "medium":"good",
                           "big":"poor"
                        }
                     },
                     "3":"vgood",
                     "4":{
                        "trunk":{
                           "small":"good",
                           "medium":"good",
                           "big":"vgood"
                        }
                     },
                     "5":"vgood"
                  }
               }
            }
         }
      }
   }
}
```