CSC 535 Data Mining

Assignment 1 Report Collection

Submitted to:

Dr. Jamil Saquer

Author(s):

*Chase Dickerson*

*Jacob Schaum*

**Report**

**Introduction**

For our real-life data set, we decided to go with the “Buy Computer” data set. This data set can be found at <https://www.kaggle.com/usman27/buy-computer/version/1>. We chose this data set because we felt we could easily work with it with our implementation of the ID3 algorithm. We developed our ID3 algorithm to easily work with dictionaries (in Python) and the data set could easily be transferred to a dictionary. The data set was also interesting to us because it dealt with buying computers which is something both of us have done in the past. It was interesting to see a data set we could both relate too.

**Background**

The algorithm we used for this assignment was ID3. We used ID3 to make a Decision Tree for a specific data set. Within our ID3 algorithm, we used entropy to help with our information gain portion of the algorithm. The entropy within the information gain portion helped us determine a test attribute to use as a root node in our decision tree.

The ID3 algorithm we implemented is a simple decision tree forming algorithm. It worked fast and efficiently with our real-life data set and created an accurate decision tree. The data set had good data which made it easier for making the decision tree, but we also implemented a solution that could handle missing or bad data.

**Implementation**

min\_entropy **=** math**.**inf

test\_attribute **=** ''

**for** attribute **in** attributes**:**

curr\_entropy **=** entropy**(**attribute**,** dataset**)**

**if** **(**entropy\_d **-** curr\_entropy**)** **<** min\_entropy**:**

min\_entropy **=** **(**entropy\_d **-** curr\_entropy**)**

test\_attribute **=** attribute

#This function finds entropy

**def** entropy**(**attribute**,** dataset**):**

entropy\_sum **=** 0

data\_dict **=** **{}**

**for** data **in** dataset**:**

**if** **not** data**[**0**][**attribute**]** **in** data\_dict**:**

data\_dict**[**data**[**0**][**attribute**]]** **=** 1

**else:**

data\_dict**[**data**[**0**][**attribute**]]** **+=** 1

data\_max **=** len**(**dataset**)**

**for** value **in** data\_dict**.**values**():**

entropy\_sum **=** **-**1**\*(**value**/**data\_max**\***math**.**log**(**value**/**data\_max**,** 2**))**

**return** entropy\_sum

Figure 2: Finding entropy for attributes

Figure 1: Basic checks at beginning of ID3 algorithm

# if data set class are all same, return class

**if** len**(**dataset**)** **>** 0**:**

first\_element **=** dataset**[**0**][**1**]**

all\_same **=** **False**

**for** data **in** dataset**:**

**if** data**[**1**]** **!=** first\_element**:**

all\_same **=** **False**

**break**

**else:**

all\_same **=** **True**

**if** all\_same**:**

**return** data**[**1**]**

**if** attributes**:**

Our implementation of the algorithm was very similar to the one taught in lecture. We recursively called ID3 to build our decision tree. We made sure to put in the necessary checks at the beginning of the algorithm in order create leaf nodes when needed. We learned very quickly while writing our implementation that the checks were essential. We needed the checks in the correct spot to ensure accurate results. Figure 1 is how we implemented checking for all samples of the same class and if the attribute list was empty.

We used the entropy of the attributes to select our test attributes. The test attributes were used to be nodes of the tree and to have branches off of them. Figure 2 represents our creative way of finding the test attribute based off of entropy and information gain. Once the test attribute was found we simply continued with the algorithm and created branches and leaf nodes when necessary.

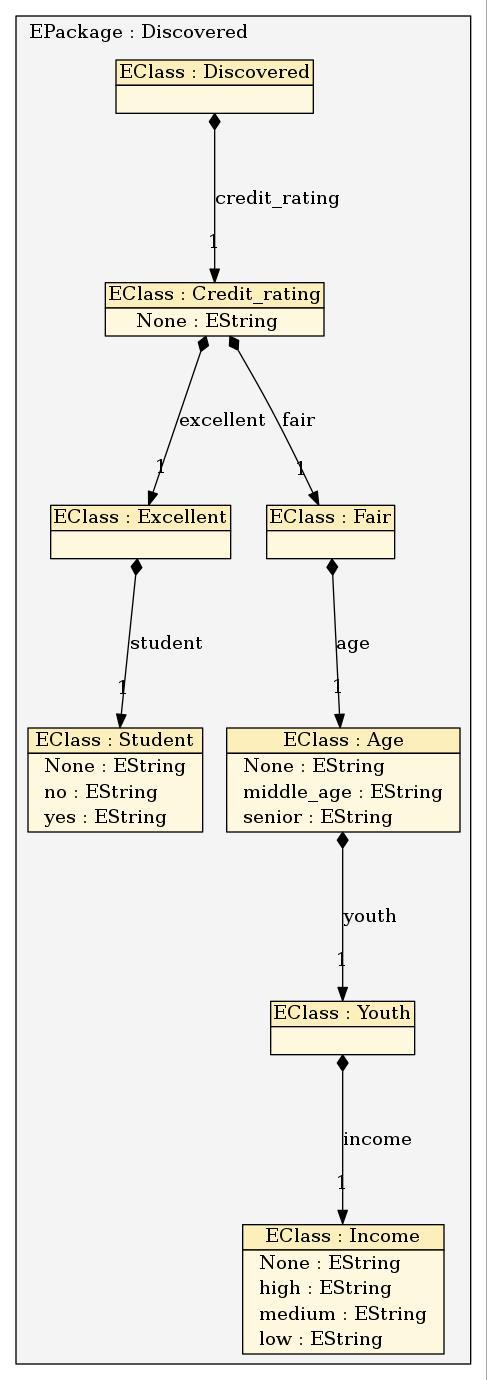
**Experimental Setup and Results**

We first set up by finding a data set we wanted to work with. We then needed to turn the data set into a data structure that our algorithm could work with. We tested more than one data set and finally settled on one of choice. We then tested our algorithm with the data set and looked at the results. We got a good decision tree and were happy with the results.

Console output:



Visual representation of console list to tree representation:



**Conclusion**

In conclusion, we learned a great deal on how the ID3 algorithm is used to generate decision trees that can be used in the field of Data Mining. The ID3 algorithm is simple but very useful. A variety of data sets can be used with the algorithm and decision trees are very helpful.

**Code**

"""

Slate Folder:

Program: assn1.py

Programmers: Chase Dickerson and Jacob Schaum

Description: This program creates a decision tree based off the data set Buy\_computer.csv

"""

#Imports

**import** math

**import** itertools

#Global Variables

training\_data **=** **[]**

#Functions

#This function finds all of the main attributes

**def** getAttributes**():**

attributes **=** **[]**

input\_data **=** training\_data**[**0**][**0**]**

**for** key **in** input\_data**:**

attributes**.**append**(**key**)**

**return** attributes

#This function finds entropy

**def** entropy**(**attribute**,** dataset**):**

entropy\_sum **=** 0

data\_dict **=** **{}**

**for** data **in** dataset**:**

**if** **not** data**[**0**][**attribute**]** **in** data\_dict**:**

data\_dict**[**data**[**0**][**attribute**]]** **=** 1

**else:**

data\_dict**[**data**[**0**][**attribute**]]** **+=** 1

data\_max **=** len**(**dataset**)**

**for** value **in** data\_dict**.**values**():**

entropy\_sum **=** **-**1**\*(**value**/**data\_max**\***math**.**log**(**value**/**data\_max**,** 2**))**

**return** entropy\_sum

#This function finds the classes of the data set

**def** getClasses**():**

class\_count **=** **{}**

**for** data **in** training\_data**:**

**if** **not** data**[**1**]** **in** class\_count**:**

class\_count**[**data**[**1**]]** **=** 1

**else:**

class\_count**[**data**[**1**]]** **+=** 1

**return** class\_count

#This function finds the values of the data set

**def** getValueData**(**root\_node**):**

attribute\_values **=** **[]**

**for** data **in** training\_data**:**

# need array of all values, then build dict of counts for True/False for each value

**if** data**[**0**][**root\_node**]** **not** **in** attribute\_values**:**

attribute\_values.append(data[0][root\_node])

val\_data = {}

for att in attribute\_values:

val\_inner = {}

for data in training\_data:

if data[0][root\_node] == att:

if not data[1] in val\_inner:

val\_inner[data[1]] = 1

else:

val\_inner[data[1]] += 1

val\_data[data[0][root\_node]] = val\_inner

return val\_data

#This is our actual id3 function

# It is called recursivley to build decision tree

def id3(attributes, entropy\_d, dataset):

main\_tree = {}

# if data set class are all same, return class

if len(dataset) > 0:

first\_element = dataset[0][1]

all\_same = False

for data in dataset:

if data[1] != first\_element:

all\_same = False

break

else:

all\_same = True

if all\_same:

return data[1]

if attributes:

min\_entropy = math.inf

test\_attribute = ''

for attribute in attributes:

curr\_entropy = entropy(attribute, dataset)

if (entropy\_d - curr\_entropy) < min\_entropy:

min\_entropy = (entropy\_d - curr\_entropy)

test\_attribute = attribute

branches = getValueData(test\_attribute)

tree = {}

tree[None] = getMajorityClass(dataset)

attributes.remove(test\_attribute)

for branch in branches.items():

if len(branch[1]) == 1:

tree[branch[0]] = next(iter(branch[1]))

else:

new\_training = []

for data in dataset:

if data[0][test\_attribute] == branch[0]:

new\_training.append(data)

tree[branch[0]] = id3(attributes, entropy\_d, new\_training)

main\_tree[test\_attribute] = tree

return main\_tree

else:

return getMajorityClass(dataset)

else:

return getMajorityClass(training\_data)

#Finds the majority class

def getMajorityClass(dataset):

class\_counts = {}

for data in dataset:

if not data[1] in class\_counts:

class\_counts[data[1]] = 1

else:

class\_counts[data[1]] += 1

max\_class\_val = 0

max\_class = ""

for classes in class\_counts.items():

if classes[1] > max\_class\_val:

max\_class = classes[0]

max\_class\_val = classes[1]

return max\_class

#Used to classify new data

def classify(tree, sample, expected):

if type(tree) is str:

print(tree)

return

first, second = next(iter(tree.keys())), next(iter(tree.values()))

for key, vals in sample.items():

if key == first:

if vals in second:

classify(second[vals], sample, expected)

else:

print(second[None])

return

elif len(sample) == 1:

print(second[None])

return

#Prepares the data from the file

def preprocessData():

with open('Buy\_computer.csv', "r") as file:

for line in itertools.islice(file, 1, 1700):

buy\_computer = {}

buy\_computer["age"] = line.split(',')[1]

buy\_computer["income"] = line.split(',')[2]

buy\_computer["student"] = line.split(',')[3]

buy\_computer["credit\_rating"] = line.split(',')[4]

training\_data.append(tuple((buy\_computer, line.split(',')[5].strip())))

# Main function

def main():

preprocessData()

attributes = getAttributes()

# Get class counts/entropy

classes = getClasses()

total\_classes = 0

entropy\_d = 0

for klass in classes.values(): # Calculate total number of classes (non-unique)

total\_classes += klass

for klass in classes.values(): # Calculate entropy for each unique class

entropy\_d += -1\*(klass/total\_classes\*math.log(klass/total\_classes, 2))

tree = id3(attributes, entropy\_d, training\_data)

for sample in training\_data:

classify(tree, sample[0], sample[1])

print(tree)

if \_\_name\_\_ == "\_\_main\_\_":

main()