CSC 635 Data Mining

## Assignment 2 Report

### Submitted to:

### Dr. Jamil Saquer

### Author(s):

### Rafail Islam

**HW2-Decision Tree**

**Introduction**

In this assignment, I have worked with a give small data set of hiring an employee. Based on the features, decision will be ‘True’ or ‘False’. To solve this problem, I implemented the ID3 algorithm to build a decision tree. The desired decision tree is the one that can predict the test data set successfully. Keeping this in mind, I looked forward to building the tree with most information gain and so on. After building the decision tree, I tested the tree with the provided test sample.

In part two, I was looking at a data set for binary classification as this is our first implementation of the decision tree. Therefore, this implementation might not work very well for complex data set with multiple classes. I have selected a data set of diabetic patients [1] for binary classification. I will have to say if a patient has diabetes or not based on some important features provided by this data set. The data sets consist of several medical predictor (independent) variables and one target (dependent) variable, Outcome. Independent variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on. This data set contains 786 instances and 9 columns. I took a portion of data as training data and the rest used as testing data. The decision tree model that I implemented in part 1 used to solve this problem.

**Background**

In real life, there are so many problems where we need to made decisions based on some variables. Most of the everyday decision-making problems are solved by human intuitions. However, most of the cases in business, banking, research, etc, decision making is not easy. To some extent, it beyond human’s normal calculations. To solve this kind of problem, researchers have introduced some useful algorithms. The decision tree is one of them which is used widely for classification problems. In this work, I implemented the ID3 algorithm among different types of decision trees algorithms. Whiling building the decision tree, one important factor was implemented so that this decision tree algorithm becomes robust against missing or inappropriate values.

Along with the most common libraries (numpy, pandas), I also used scikit-learn for splitting data set for training and testing. The pprint was used to print the decision tree.

**Implementation**

To implement the ID3 algorithm, I followed the steps mentioned in the lecture slide. The algorithms build a decision tree recursively. Firstly, a node is created to store information- feature as an inner node, label as a leaf node. I used a dictionary for creating a node. Secondly, I check if all the classes in the data are the same. The node is a label with the class if all classes are the same. The node is labeled with the majority class of the data set if the attribute list is empty. Thirdly, the attribute with the most information gain is selected and made that node as current root. Then grow a branch for all possible values of the selected attribute. Here, I used a trick to handle missing or incorrect features. Added a node with ‘None’ in the growing branch. The newly created branch is added to the previous node which was created in the beginning. After creating the branch, we generate a sub-data set for every possible value of the selected attribute and marked the branch node with the majority class if the subclass is empty. Otherwise, grow the branch by calling the same function recursively until all branches go to a leaf node.

Once the decision tree was ready, I tested the decision tree to predict the outcome for tested data. To measure the accuracy, the label of the actual data was compared with the predicted outcome from the decision tree. In the second part, the percentage of the accuracy was measured by dividing the total corrected matched label by the total number of instances in the test data set and then by multiplying by 100.

**Experimental Setup and Results**

I used jupyter notebook and python 3.8 for this assignment. I prepossessed the data, both in part I and part II, by using pandas libraries. To make this algorithm work, one must input datafram to build the decision tree. The performance of this algorithm was as expected in part I- it can predict correctly with missing and incorrect data.

However, this algorithm did not show good accuracy for the real-life data set in part II. It gave around 56-58% accuracy with a test sample of 20% of total instances. With that said, it is clear that this algorithm needs to be improved and optimized to work better in real-life problems.

**Conclusion**

In this assignment, I implemented the ID3 algorithm from the scratch. In most cases, we used libraries such as scikit-learn to build a decision tree. This assignment gives a clear idea of how to build a decision tree from scratch. However, this ID3 is not very optimized to give better accuracy. Further improvement is necessary for this algorithm.

**References**

[1] Learning, UCI Machine. “Pima Indians Diabetes Database.” *Kaggle*, 6 Oct. 2016, www.kaggle.com/uciml/pima-indians-diabetes-database.

**Code**

"""

Assignmnet: HW2

Course : CSC 635

Program: hw2.ipynb

Author: Rafail Islam

"""

#---------------------------------Imports--------------------------------------

**import** numpy **as** np

**import** pandas **as** pd

**import** pprint

**from** sklearn **import** tree

**from** matplotlib **import** pyplot **as** plt

**from** sklearn**.**model\_selection **import** train\_test\_split

#------------------------------------------------------------------------------

#---------------------------------Variables------------------------------------

#only dataset for problem 01

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

**({**'level'**:**'Senior'**,** 'lang'**:**'Java'**,** 'tweets'**:**'no'**,** 'phd'**:**'no'**},** **False),**

**({**'level'**:**'Senior'**,** 'lang'**:**'Java'**,** 'tweets'**:**'no'**,** 'phd'**:**'yes'**},** **False),**

**({**'level'**:**'Mid'**,** 'lang'**:**'Python'**,** 'tweets'**:**'no'**,** 'phd'**:**'no'**},** **True),**

**({**'level'**:**'Junior'**,** 'lang'**:**'Python'**,** 'tweets'**:**'no'**,** 'phd'**:**'no'**},** **True),**

**({**'level'**:**'Junior'**,** 'lang'**:**'R'**,** 'tweets'**:**'yes'**,** 'phd'**:**'no'**},** **True),**

**({**'level'**:**'Junior'**,** 'lang'**:**'R'**,** 'tweets'**:**'yes'**,** 'phd'**:**'yes'**},** **False),**

**({**'level'**:**'Mid'**,** 'lang'**:**'R'**,** 'tweets'**:**'yes'**,** 'phd'**:**'yes'**},** **True),**

**({**'level'**:**'Senior'**,** 'lang'**:**'Python'**,** 'tweets'**:**'no'**,** 'phd'**:**'no'**},** **False),**

**({**'level'**:**'Senior'**,** 'lang'**:**'R'**,** 'tweets'**:**'yes'**,** 'phd'**:**'no'**},** **True),**

**({**'level'**:**'Junior'**,** 'lang'**:**'Python'**,** 'tweets'**:**'yes'**,** 'phd'**:**'no'**},** **True),**

**({**'level'**:**'Senior'**,** 'lang'**:**'Python'**,** 'tweets'**:**'yes'**,** 'phd'**:**'yes'**},** **True),**

**({**'level'**:**'Mid'**,** 'lang'**:**'Python'**,** 'tweets'**:**'no'**,** 'phd'**:**'yes'**},** **True),**

**({**'level'**:**'Mid'**,** 'lang'**:**'Java'**,** 'tweets'**:**'yes'**,** 'phd'**:**'no'**},** **True),**

**({**'level'**:**'Junior'**,** 'lang'**:**'Python'**,** 'tweets'**:**'no'**,** 'phd'**:**'yes'**},** **False)**

**]**

#---------------------------------Functions----------------------------

# converting list into dataframe for convenience

**def** load\_dataset\_1**(**training\_data**):**

""" this functin load data set for problem 01. it takes trainig\_data as argumnet, and returns

a dataframe as train\_df

Parameters

-----------

training\_data : tuples

dataset as tuples

Returns

----------

train\_df : pandas dataframe

dataset as dataframe

"""

column**=** **[**'level'**,** 'lang'**,** 'tweets'**,** 'phd'**,**'class\_label'**]**

train\_df **=** pd**.**DataFrame**(**columns**=**column**)**

**for** d **in** training\_data**:**

dic **=** d**[**0**]**

dic**[**'class\_label'**]=**d**[**1**]**

df1**=**pd**.**DataFrame**([**dic**])**

train\_df **=** train\_df**.**append**(**df1**,**ignore\_index **=** **True)**

**return** train\_df

train\_df **=** load\_dataset\_1**(**training\_data**)**

# Find entropy of class

**def** entropyD**(**df **,** label **):**

""" This function calculates endropy of a dataset.

Parameters

-----------

df : dataframe

dataset as dataframe

label : str

label of the dataset

Returns

----------

entropy : float

entropy of the dataset

"""

entropy **=** 0

# calculate entropy

**for** classLabels **in** df**[**label**].**unique**():**

prob **=** df**[**label**].**value\_counts**()[**classLabels**]/**len**(**df**)**

entropy **+=** **-** prob **\*** np**.**log2**(**prob**)**

#print(entropy)

**return** entropy

# Find entropy of an attribute

**def** entropyA**(**df**,**attribute**,**label**):**

""" This function calculates endropy of an attributes over the class of a dataset.

Parameters

-----------

df : dataframe

dataset as dataframe

attribute : str

attribute name

label : str

label of the dataset

Returns

----------

entropy : float

entropy of the given attribute over class of the data set

"""

# unique values of the given attribute

attributeValues **=** df**[**attribute**].**unique**()**

# unique class of the dataset

classLabels **=** df**[**label**].**unique**()**

entropyAi **=** 0

# entropyA(D) = sum ( |Dj|/|D|\* entropy(Dj) )

**for** value **in** attributeValues**:**

entropyDj **=** 0

# entropy(Dj)

**for** cLabel **in** classLabels**:**

# |Dj|

dj **=** len**(**df**[**attribute**][**df**[**attribute**]==**value**][**df**[**label**]==**cLabel**])**

# |D|

d **=** len**(**df**[**attribute**][**df**[**attribute**]==**value**])**

prob **=** dj**/**d

entropyDj **+=** **-**prob **\*** np**.**log2**(**prob**+** np**.**finfo**(**float**).**eps**)** #RuntimeWarning: divide by zero encountered in log

# to avoid this, I added np.finfo(float).eps

prob **=** d**/**len**(**df**)**

entropyAi **+=-** prob **\*** entropyDj

**return** np**.**abs**(**entropyAi**)**

**def** infoGain**(**df**,**attribute\_list**,**label**):**

""" This function calculates information gain of all attributes provided.

Parameters

-----------

df : dataframe

dataset as dataframe

attribute\_list : list of str

attribute list

label : str

label of the dataset

Returns

----------

name of the columns with maximum information gain amoing the provided attributes

"""

# calculate entropy

ent\_d **=** entropyD**(**df**,**label**)**

ent\_hist**=[]**

**for** attr **in** attribute\_list**:**

# calculate attributes entropy

ent\_attr **=** entropyA**(**df**,**attr**,**label**)**

ent\_hist**.**append**(**ent\_d **-** ent\_attr **)**

#print(ent\_hist)

**return** df**.**columns**[**np**.**argmax**(**ent\_hist**)]**

infoGain**(**train\_df**,**train\_df**.**columns**[:-**1**],**'class\_label'**)**

**def** same\_class**(**df**,**label**):**

"""this function detemines the purity of a dataset.

Parameters

------------

df : dataframe

dataset

label : str

label of the dataset

Returns

----------

True/False : bool

class label : class label

"""

classes**,** counts **=** np**.**unique**(**df**[**label**],**return\_counts**=True)**

#print(classes,counts)

**if(**len**(**classes**)** **<=** 1**):**

**return** **True,**classes**[**0**]**

**else:**

**return** **False,**classes**[**np**.**argmax**(**counts**)]**

same\_class**(**train\_df**,**'lang'**)**

**def** get\_majority\_class**(**df**,**label**):**

"""this function detemines mejority class of a dataset.

Parameters

------------

df : dataframe

dataset

label : str

label of the dataset

Returns

----------

class label : dtype of class label

"""

classes**,** counts **=** np**.**unique**(**df**[**label**],**return\_counts**=True)**

**return** classes**[**np**.**argmax**(**counts**)]**

get\_majority\_class**(**train\_df**,**'class\_label'**)**

# Buid ID3 decision tree

**def** ID3**(**df**,**attribute\_list**,**label**):**

"""this function build ID3 decision tree recursively.

Parameters

------------

df : dataframe

dataset

attribute\_list : list of str

attribute list

label : str

label of the dataset

Returns

----------

dt : dictionary of dictionary

decision tree

"""

#1 create a node

dt**={}**

# 2. If samples are all of the same class C ,

# then return N as a leaf node labeled with class C

purity**,** class\_label **=** same\_class**(**df**,**label**)**

**if** purity**:**

#print("pure")

dt **=** class\_label

**return** dt

# 3. if attributes\_list is empty, then return n as a leaf node labeled with the majority class

**if** len**(**attribute\_list**)** **==** 0**:**

#print("attr empty")

dt **=** get\_majority\_class**(**df**,**label**)**

**return** dt

# 4. select test\_attribute with most information gain

test\_attribute **=** infoGain**(**df**,**attribute\_list**,**label**)**

#print("best\t",test\_attribute)

# 5. Label node N with test\_attribute

dt**={**test\_attribute**:None}**

# 6. For each known value ai of test\_attribute

#i. grow a branch from node N for the condition test\_attribute = ai

branch **={}**

branch**[None]=** get\_majority\_class**(**df**,**label**)**

**for** next\_test\_attribute **in** df**[**test\_attribute**].**unique**():**

branch**[**next\_test\_attribute**]** **=** **None**

#pprint.pprint(branch)

dt**[**test\_attribute**]=** branch

**for** value **in** df**[**test\_attribute**].**unique**():**

#print("value\t",value)

# 6 ii. let S i be the set of samples in D for which test\_attribute = ai

si **=** df**[**df**[**test\_attribute**]** **==** value**].**reset\_index**(**drop**=True)**

#print(si)

# 6 iii. if S i is empty, then attach a leaf node labeled with the majority class in D

**if** len**(**si**)==**0 **:**

**print(**"empty sub set"**)**

dt**[**test\_attribute**][**value**]=** get\_majority\_class**(**df**,**label**)**

**return** dt

**else:**

dt**[**test\_attribute**][**value**]** **=** ID3**(**si**,**si**.**columns**[:-**1**],**si**.**columns**[-**1**])**

**return** dt

**def** classify**(**sample**,**dt**):**

"""this function determine the result of a decision tree

for a single sample

Parameters

------------

sample : dictionary

sample data to predict class

dt : dictionary of dictionary

decision tree

Returns

----------

decision : dtype of leaf node

label of the predicted class

"""

#Recursively traverse the tree

**for** node **in** dt**.**keys**():**

# for missing data returns majority class

**if** node **not** **in** sample**.**keys**():**

**return** dt**[**node**][None]**

attribute **=** sample**[**node**]**

# for incurrect data returns majority class

**if** attribute **not** **in** dt**[**node**].**keys**():**

**return** dt**[**node**][None]**

dt **=** dt**[**node**][**attribute**]**

#pprint.pprint(dt)

decision **=** **False**

#go until we reach leaf node

**if** isinstance**(**dt**,**dict**)** **:**

decision **=** classify**(**sample**,** dt**)**

**else:**

decision **=** dt

#print(tree)

**break;**

**return** decision

# Here we do all the steps in problem 01

**def** problem1**():**

# Load data

train\_df **=** load\_dataset\_1**(**training\_data**)**

# Build decision tree

dt **=** ID3**(**train\_df**,**train\_df**.**columns**[:-**1**],**train\_df**.**columns**[-**1**])**

pprint**.**pprint**(**dt**)**

# sample test data

sample1 **=** **{**"level" **:** "Senior"**,**"lang" **:** "Java"**,**"tweets" **:** "yes"**,**'phd' **:** 'no'**}** # true

sample2 **=** **{**"level" **:** "Junior"**,**"lang" **:** "Java"**,**"tweets" **:** "yes"**,**"phd" **:** "yes"**}** # false

sample3 **=** **{**"level"**:** "Intern"**}** # true

sample4 **=** **{**"level"**:** "Senior"**}** # false

# classify test data

**print(**classify**(**sample1**,** dt**))**

**print(**classify**(**sample2**,** dt**))**

**print(**classify**(**sample3**,** dt**))**

**print(**classify**(**sample4**,** dt**))**

problem1**()**

# Load dataset for problem 02

**def** load\_dataset2**():**

""" Load dataset and returns data for training and testing

Parameters

--------

null

Returns

----------

training\_data : dataframe

training dataset

X\_test : dataframe

dataset for testing

y\_test : dataframe

actual class of testing dataset

"""

#https://www.kaggle.com/uciml/pima-indians-diabetes-database

dataset **=** pd**.**read\_csv**(**'diabetes.csv'**,**delimiter**=**','**)**

#dataset.head(5)

y **=** dataset**[**'Outcome'**]**

X **=** dataset**.**drop**(**'Outcome'**,**axis**=**1**)**

y**=** y**.**to\_frame**()**

# split dataset into test and train

X\_train**,**X\_test**,**y\_train**,**y\_test **=** train\_test\_split**(**X**,**y**,**test\_size**=**0.2**,** random\_state**=**42**)**

# concatenate dataframe to make traing dataset with labels

train\_data **=** pd**.**concat**([**X\_train**,**y\_train**],**axis**=**1**)**

**return** train\_data**,** X\_test**,**y\_test

**def** classification\_accuracy**(**dt**,**X\_test**,**y\_test**):**

''' This function caculate classification accuracy

Parameters

-----------

dt : dictionary of dictionaries

decision tree

X\_test : dataframe

testing dataset

Y\_test : dataframe

actual class of testing dataset

Returns

---------

Accuracy : float

Accuracy in percentage

'''

# Convert dataframe into dictionaries

samples **=** X\_test**.**to\_dict**(**'records'**)**

count **=** 0

# make 1D np array

ytest **=**np**.**squeeze**(**y\_test**.**to\_numpy**())**

**for** sample**,** y **in** zip**(**samples**,**ytest**):**

# get class for single data (row)

y\_predict **=** classify**(**sample**,**dt**)**

**if** y **==** y\_predict**:**

count **+=** 1

**return** **(**count**/**len**(**X\_test**))\***100

#classification\_accuracy(dt,X\_test,y\_test)

# Here we do all steps for problem 02

**def** problem2**():**

# load data

train\_data**,** X\_test**,** y\_test **=** load\_dataset2**()**

# build decision tree

dt **=** ID3**(**train\_data**,** attribute\_list **=** train\_data**.**columns**[:-**1**],** label **=** train\_data**.**columns**[-**1**])**

# print accuracy

**print(**"Test Accuracy %.2f%%"**%(**classification\_accuracy**(**dt**,**X\_test**,**y\_test**)))**

problem2**()**

#-------------------------------------------------------------------------------------------------------