CSC <535/635> Data Mining

## Assignment 2 Report

### Submitted to:

### Dr. Jamil Saquer

### Author(s):

### David Gray

### Aileen Bui

**Application of ID3 algorithm on classification datasets**

1. **Introduction**

The work in this report aims at training an ID3 algorithm in building a decision tree to determine whether a mushroom is edible or poisonous. The application of this algorithm will be useful for people who do not know the name of the mushroom that they see during their nature trips and want to make an informed decision on whether the mushroom is edible. We chose a dataset where the attributes describe the look of the mushroom, which can be given by anyone who possesses the mushroom under test. The Figure 1 gives an overview of parts of a mushroom.

Diagram

Description automatically generated

*Figure 1: Parts of a mushroom*

*Source: FAO.ORG*

The details of the dataset (UCI Machine Learning) is as following:

* **Classes**:

edible=e, poisonous=p

* **Attributes:**

**cap-shape**: bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s

**cap-surface**: fibrous=f, grooves=g, scaly=y, smooth=s

**cap-color**: brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y

**bruises**: bruises=t, no=f

**odor**: almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s

**gill-attachment**: attached=a, descending=d, free=f, notched=n

**gill-spacing**: close=c, crowded=w, distant=d

**gill-size**: broad=b, narrow=n

**gill-color**: black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y

**stalk-shape**: enlarging=e, tapering=t

**stalk-root**: bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?

**stalk-surface-above-ring**: fibrous=f, scaly=y, silky=k, smooth=s

**stalk-surface-below-ring**: fibrous=f, scaly=y, silky=k, smooth=s

**stalk-color-above-ring**: brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y

**stalk-color-below-ring:** brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y

**veil-type**: partial=p, universal=u

**veil-color**: brown=n, orange=o, white=w, yellow=y

**ring-number**: none=n, one=o, two=t

**ring-type:** cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z

**spore-print-color**: black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y

**population**: abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y

**habitat**: grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

1. **Background**

The ID3 algorithm was used to recursively split the dataset and build branches. At each iteration, the best attribute was selected to be used for splitting the data. In order to find the best attribute, entropy and information gain of all attributes are calculated to select the top one. The algorithm is also designed to work with missing values at certain attributes by creating a default branch for these cases based on the majority value in that attribute. The recursive function compares the root node of the decision tree to the value of the concurrent attribute in the test data returning the class if it has reached a leaf node or recursively calling the function with the selected branch as the remaining decision tree.

1. **Implementation**

In order to build the decision tree, several functions were built to calculate the necessary metrics used during the training process, including “entropy” (getEntropy function) and “information gain” (getInfoGain function). These metrics are used in the function “chooseBestAttribute” in order to select the attribute with the highest information gain as the node for splitting the dataset.

The main function of the algorithm is “generateDT” function that generate the decision tree. In this function, special cases were handled in order to terminate the recursive call. There are two special cases:

* When there are no attributes exist, it will return the mode of the classes within the selected tuples.
* When only a single class exists, it will return the class.

Outside of those special cases, the function will split the dataset based on the selected attribute. In order to handle the missing values, the majority value of the attribute was found as node\_majority so that in case of missing data, the node\_majority will prevail.

Graphical user interface, text, application

Description automatically generated

The last function is “classify” function which is used to apply the decision tree on the test data. This function handles the exception case when there are missing values or values that do not exists in the training data. The decision for those cases will then base on the default branch of the node\_majority built during the training process. In order to check if the decision is at its leaf node yet or not, a datatype comparison was used, which keeps the code simple and fast.

Graphical user interface, text, application

Description automatically generated

1. **Experimental Setup and Results**

The algorithm was built in a python file that works with Python 3 version. The required libraries include math, pandas and numpy. Input data for training was read from a csv file with over seven-thousand entrees of mushrooms. Below is the decision tree output from training process.

**Mushroom decision tree:**

('odor', {'p': 'p', 'a': 'e', 'l': 'e', 'n': ('spore-print-color', {'n': 'e', 'k': 'e', 'w': ('habitat', {'w': 'e', 'l': ('cap-color', {'c': 'e', 'n': 'e', 'w': 'p', 'y': 'p', 'None': 'e'}), 'd': ('gill-size', {'n': 'p', 'b': 'e', 'None': 'p'}), 'g': 'e', 'p': 'e', 'None': 'e'}), 'h': 'e', 'r': 'p', 'o': 'e', 'y': 'e', 'b': 'e', 'None': 'e'}), 'f': 'p', 'c': 'p', 'y': 'p', 's': 'p', 'm': 'p', 'None': ('spore-print-color', {'n': 'e', 'k': 'e', 'w': ('habitat', {'w': 'e', 'l': ('cap-color', {'c': 'e', 'n': 'e', 'w': 'p', 'y': 'p', 'None': 'e'}), 'd': ('gill-size', {'n': 'p', 'b': 'e', 'None': 'p'}), 'g': 'e', 'p': 'e', 'None': 'e'}), 'h': 'e', 'r': 'p', 'o': 'e', 'y': 'e', 'b': 'e', 'None': 'e'})})

Accuracy in our algorithm is very important seeing as the prediction from the software may be the deciding factor between life and death for a user. To test the performance of the algorithm, one thousand samples of mushrooms with populated attributes were used as test data. The algorithm accurately predicted 99.6% of the samples as seen in Figure 4.

A picture containing text

Description automatically generated*Figure 4: Classification accuracy*

To further test our algorithm, we decided to use two mushrooms outside of the dataset that are commonly mistaken for each other. Agarius bisporus (left) is the edible mushroom and Amanita bisporigera (right) is the poisonous mushrooms. We manually entered values for each of the attributes about these mushrooms.

 

*Figure 2: Agarius bisporus Figure 3: Amanita bisporigera*

*Source: Gluckspilze. Source: Gstatic*

The classification function accurately predicted which mushroom is edible and which one is poisonous. The result printout from the script is shown in Figure 5.

Text

Description automatically generated

*Figure 5: Classification result on 2 specific test cases*

1. **Conclusion**

Mushroom classification is a sensitive subject. Many mushrooms appear much different when they are young verses when they are mature. Although we were able to make accurate predictions as to whether Agarius bisporus and Amanita bisporigera were edible or poisonous, we did notice that if only one of the attributes was changed it could lead to the wrong classification. Mushroom classification will need more testing, or more attention than a basic decision tree can provide.

**References**

FAO ORG. Wild edible fungia global overview of their use and importance to people. (n.d.). Retrieved October 05, 2020, from <http://www.fao.org/3/y5489e/y5489e00.htm>

Gluckspilze. Retrieved October 05, 2020, from <https://gluckspilze.com/media/image/product/1062/lg/buttom-mushroom-brown-agaricus-bisporus-pure-culture-for-organic-mushroom-cultivation-according-to-regulation-ec-834-2007-and-889-2008-at-bio-701-strain-no-105003.jpg>

Gstatic. Retrieved October 05, 2020, from https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcSdhyjvHgRO9-sUi6QJOU5dp2At0Y9PYjbWrw&usqp=CAU (right)

UCI Machine Learning. Mushroom Classification dataset. Retrieved October 05, 2020, from <https://www.kaggle.com/uciml/mushroom-classification>

**Code**

'''

Program: hw2.ipynb

Author: David Gray - Aileen Bui

Description: Complete homework 2 assignment containing decision tree generation and predicting classification of mushroom samples.

'''

import pandas as pd

import numpy as np

import math

def getEntropy(D):

entropy = 0

probabilities = (D['class'].value\_counts() / D['class'].count()).tolist()

for x in probabilities:

entropy += (-x)\*math.log(x,2)

return entropy

def getInfoGain(D, attr, wholeEntropy):

testEntropy = 0

total = D[attr].count()

for value in D[attr].unique():

count = D[attr][D[attr] == value].count()

testEntropy += (count/total)\*(getEntropy(D[D[attr] == value]))

infoGain = wholeEntropy - testEntropy

return infoGain

def chooseBestAttribute(D):

wholeEntropy = getEntropy(D)

attr\_list = D.columns.tolist()

attr\_list.remove('class')

infoGain = []

for a in attr\_list:

infoGain.append([a, getInfoGain(D, a, wholeEntropy)])

gaindf = pd.DataFrame(infoGain, columns=['Attribute','Info Gain'])

return gaindf['Attribute'][gaindf['Info Gain'] == gaindf['Info Gain'].max()].tolist()[0]

def generateDT(D):

branches = {}

classes = D['class'].unique().tolist()

if len(classes) == 1: #Handle cases of only single class

return classes[0]

if len(D.columns.tolist()[:-1]) < 1: #Handle cases of no attribute

return D['class'].mode()

node = chooseBestAttribute(D)

node\_values = D[node].unique()

node\_majority = D.mode()[node][0]

node\_values = np.append(node\_values,'None')

for value in node\_values:

# If the value is missing or not within the values existing in the training data,

# we will treat it the same with the majority value of this node

if value == 'None':

branches['None'] = generateDT(D[D[node] == node\_majority])

else:

branches[str(value)] = generateDT(D[D[node] == value])

return (node, branches)

def classify(dt, test\_data):

root = dt[0]

value = test\_data[root]

try:

decision\_rule = dt[1][value]

except:

decision\_rule = dt[1]['None']

if type(decision\_rule) != tuple:

return decision\_rule

return classify(decision\_rule, test\_data)

# Training data provided in Homework requirement

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)

]

# Convert the training data into a dataframe

train\_df = pd.DataFrame(columns = ['level', 'lang', 'tweets', 'phd', 'class'])

for i in range(len(training\_data)):

row = training\_data[i]

data = row[0]

data['class'] = row[1]

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

# Call the function to generate the decision tree on training data

decision\_tree = generateDT(train\_df)

print('Decision tree:\n', decision\_tree)

# Generate testing data

test\_data = [

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

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

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

# Apply the decision tree on test data and output the result

for i in range(len(test\_data)):

test\_input = test\_data[i]

actual = test\_input[1]

predicted = classify(decision\_tree, test\_input[0])

print('Test ', i, ':', '\nActual: ', actual, '\nPredicted: ', predicted, '\n---', '\nCorrect' if actual == predicted else '\nIncorrect','\n------------')

# Get the real dataset as training data and call function to generate decision tree

real\_data = pd.read\_csv('mushrooms.csv')

decision\_tree = generateDT(real\_data)

print('Decision tree:\n', decision\_tree)

real\_data

#Genertate testing data for accuracy

accuracy\_test\_data = original\_data.iloc[-1000:]

accuracy\_test\_data

total\_correct\_count = 0

# Apply the decision tree on test data and output the result

for i in range(len(accuracy\_test\_data)):

test\_input = accuracy\_test\_data.iloc[i]

actual = test\_input.loc['class']

predicted = classify(decision\_tree, test\_input)

if actual == predicted:

total\_correct\_count += 1

print('Correctly predicted: ', total\_correct\_count, ' out of ', len(accuracy\_test\_data), 'samples', '\nAccuracy: ', (total\_correct\_count/len(accuracy\_test\_data))\*100, '\n------------')

# Generate testing data

mushroom\_test\_data = [

({'cap-shape':'x', 'cap-surface':'y', 'cap-color':'n', 'bruises':'f', 'odor':'n', 'gill-attachment':'f', 'gill-spacing':'w', 'gill-size':'n', 'gill-color':'n', 'stalk-shape':None, 'stalk-root':None, 'stalk-surface-above-ring':None, 'stalk-surface-below-ring':None, 'stalk-color-above-ring':'w', 'stalk-color-below-ring':'w', 'veil-type':None, 'veil-color':None, 'ring-number':'o', 'ring-type':'l', 'spore-print-color':'n', 'population':'n', 'habitat':'g'},'e'),

({'cap-shape':'f', 'cap-surface':'s', 'cap-color':'w', 'bruises':'f', 'odor':'f', 'gill-attachment':'f', 'gill-spacing':'w', 'gill-size':'n', 'gill-color':'w', 'stalk-shape':'t', 'stalk-root':'b', 'stalk-surface-above-ring':'f', 'stalk-surface-below-ring':'f', 'stalk-color-above-ring':'w', 'stalk-color-below-ring':'w', 'veil-type':'p', 'veil-color':'w', 'ring-number':'o', 'ring-type':None, 'spore-print-color':'w', 'population':'s', 'habitat':'w'},'p')]

# Apply the decision tree on test data and output the result

for i in range(len(mushroom\_test\_data)):

test\_input = mushroom\_test\_data[i]

actual = test\_input[1]

predicted = classify(decision\_tree, test\_input[0])

print('Test ', i, ':', '\nActual: ', actual, '\nPredicted: ', predicted, '\n---', '\nCorrect' if actual == predicted else '\nIncorrect','\n------------')