Andrew Garcia
3/1/2022

ECE 5984 SP22 – Prof. Jones – HW3

Part 1:

Handwritten Calculations and Tree:

ECE 5984 - Homework 3 Andrew Garcia

Part 1: $IG(d, D) = H(t, D) - rem(d, O)$

Entropy: $H(t,D) = -\sum_{l \in levels(t)} [p(t=l) \log_2 (p(t=l))]$

$$= -[p(t=\text{"Edible"}) * \log_2 (p(t=\text{"Edible"})) + p(t=\text{"Not Edible"}) * \log_2 (p(t=\text{"Not Edible"}))]$$

$$= -[(16/24) \log_2 (16/24) + (8/24) \log_2 (8/24)]$$

$$= -[-0.3899 + -0.5283]$$

$$= 0.9183 \text{ bits}$$

Remaining Entropy: $rem(d, D) = \sum_{l \in levels(d)} \frac{|D_{d=l}|}{|D|} H(t, D_{d=l})$

$$rem(\text{White}, D) = \frac{|D_{White} = T|}{D} H(t, D_{White} = T) + \frac{|D_{White} = F|}{D} H(t, D_{White} = F)$$

$$= \left(\frac{10}{24}\right) * -\sum_{l \in [\text{Edible, Not Edible}]} [p(t=l) \log_2 (p(t=l))] - \left(\frac{14}{24}\right) \sum_{l \in [\text{Edible, Not Edible}]} [p(t=l) \log_2 (p(t=l))]$$

$$= -\frac{10}{24} \left[\left(\frac{7}{10}\right) \log_2 \left(\frac{7}{10}\right) + \left(\frac{3}{10}\right) \log_2 \left(\frac{3}{10}\right)\right] - \frac{14}{24} \left[\left(\frac{9}{14}\right) \log_2 \left(\frac{9}{14}\right) + \left(\frac{5}{14}\right) \log_2 \left(\frac{5}{14}\right)\right]$$

$$= \boxed{0.9157 \text{ Bits}}$$

Andrew Garcia
3/1/2022

$$\text{rem}(\text{Tall}, D) = \frac{|D_{\text{Tall}=T}|}{|D|} H(t, D_{\text{Tall}} = T) + \frac{|D_{\text{Tall}=F}|}{|D|} H(t, D_{\text{Tall}} = F)$$

$$= \frac{14}{24}\left[-1 * \left(\frac{10}{14}\log_2\left(\frac{10}{14}\right)\right) - \left(\frac{4}{14}\log_2\left(\frac{4}{14}\right)\right)\right]$$

$$+ \frac{10}{24}\left[-\frac{6}{10}\log_2\left(\frac{6}{10}\right) - \frac{4}{10}\log_2\left(\frac{4}{10}\right)\right]$$

$$= \boxed{0.9080 \text{ bits}}$$

$$\text{rem}(\text{Frilly}, D) = -\frac{8}{24}\left[\frac{3}{8}\log_2\left(\frac{3}{8}\right) + \frac{5}{8}\log_2\left(\frac{5}{8}\right)\right]$$

$$- \frac{16}{24}\left[\frac{13}{16}\log_2\left(\frac{13}{16}\right) + \frac{3}{16}\log_2\left(\frac{3}{16}\right)\right]$$

$$= \boxed{0.7823 \text{ bits}}$$

$$IG(\text{White}, D) = 0.9183 - 0.9157 = \boxed{0.0026 \text{ bits}}$$

$$IG(\text{Tall}, D) = 0.9183 - 0.9080 = \boxed{0.0103 \text{ bits}}$$

$$IG(\text{Frilly}, D) = 0.9183 - 0.7823 = \boxed{0.1360 \text{ bits}}$$

Andrew Garcia
3/1/2022

Entropy for when Frilly is True:

$$H(t,D) = -[p(t=\text{"Edible"})\log_2(p(t=\text{"Edible"})) + p(t=\text{"Not Edible"})\log_2(p(t=\text{"Not Edible"}))]$$

$$= -[(3/8)\log_2(3/8) + (5/8)\log_2(5/8)]$$

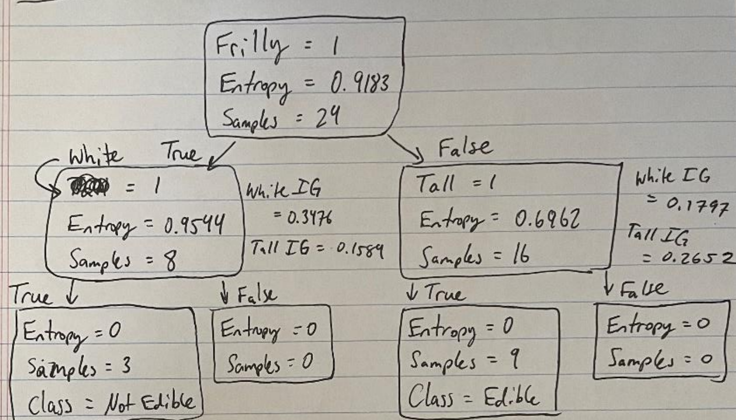
$$= \boxed{0.9544 \text{ bits}}$$

Entropy for when Frilly is False:

$$H(t,D) = -[(13/16)\log_2(13/16) + (3/16)\log_2(3/16)]$$

$$= \boxed{0.6962}\ \boxed{\text{bits}}$$

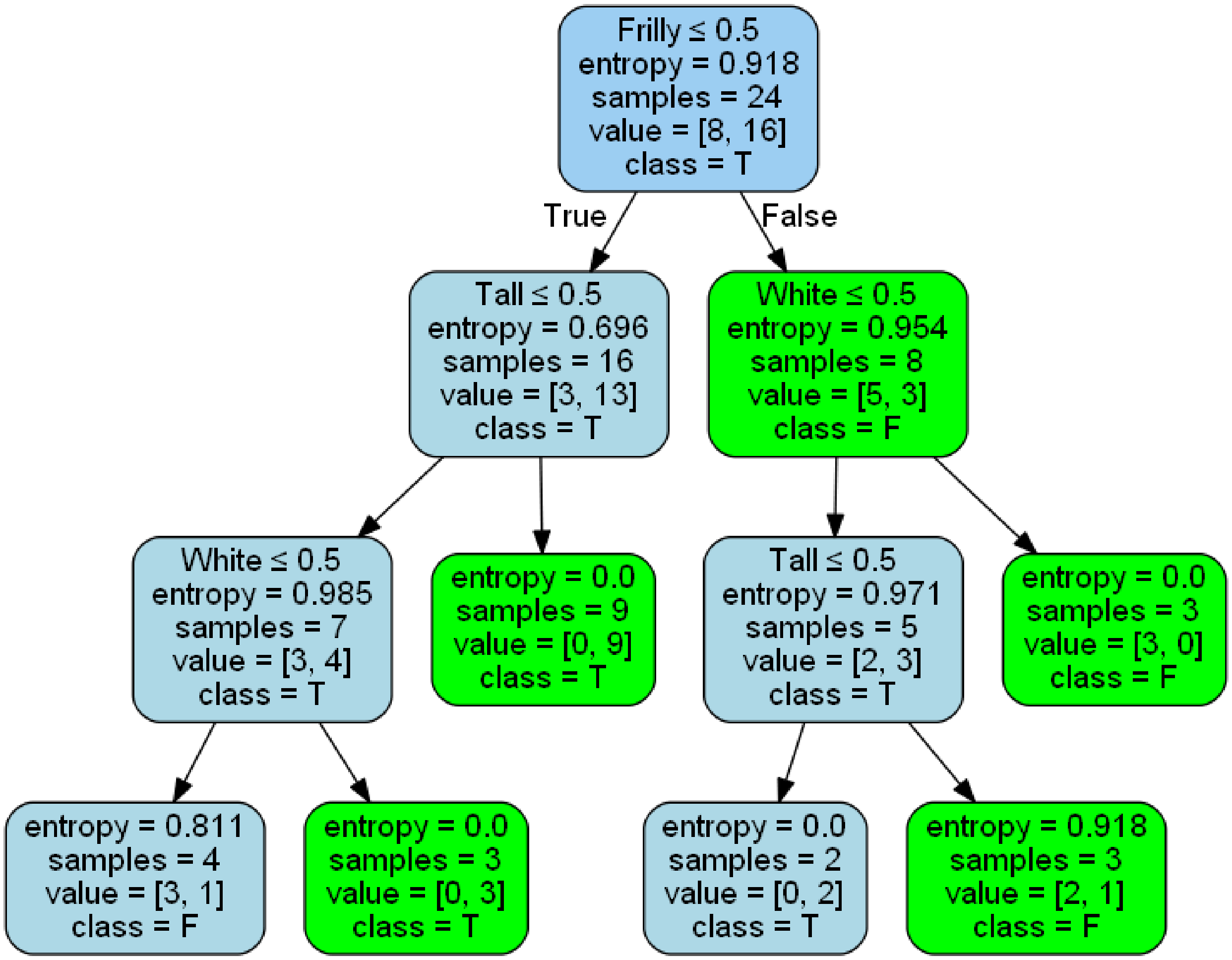
Decision Tree for Alien Mushrooms:



Andrew Garcia
3/1/2022

Part 2:

Decision Tree:



Discussion:

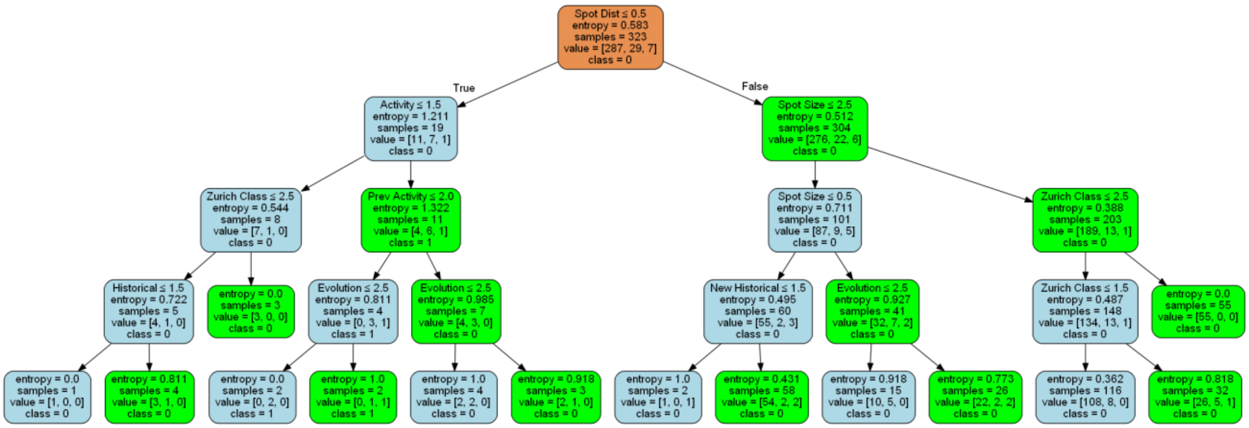
When comparing the decision tree from python to my hand drawn decision tree, I notice that my hand drawn tree calculates entropy to be zero after the second leaf. Scikit learn uses determines if something is true or false by checking if it is less than or equal to 0.5 because the values are 0 or 1. I think my hand drawn tree might be incorrect after seeing this decision tree from python because I did not continue to calculate entropy after “is White” or “is Tall”. In my calculations it was able to tell, based on those two features, whether the mushroom was edible or not.

Andrew Garcia
3/1/2022

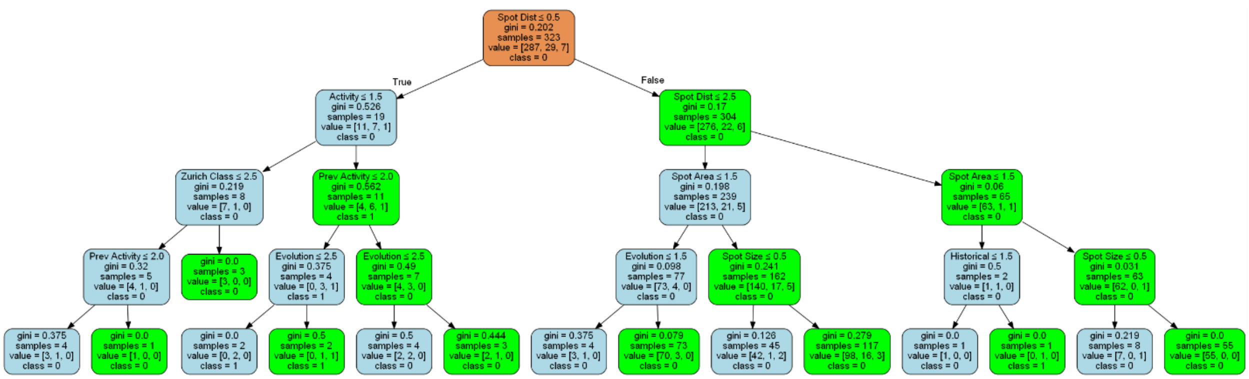
Part 3:

Part 3 Decision Tree:

Criteria = Entropy



Criteria = Gini



Discussion:

Entropy Training set score = 0.8978328173374613

Gini Training set score = 0.9009287925696594

Using the criteria of gini yields a better decision tree for this dataset because the entropy values at the end of the tree are still high where the gini values at the end of the tree are near zero. Gini also had a better test score, but not by much.

Andrew Garcia
3/1/2022

Code

Part 2 Code:

```
import pandas

import os

from sklearn import tree

import pydotplus

import collections


# for a two-class tree, call this function like this:

# writegraphtofile(clf, ('F', 'T'), dirname+graphfilename)

def writegraphtofile(clf, feature_labels, classnames, pathname):

    dot_data = tree.export_graphviz(clf, out_file=None,

                                    feature_names=feature_labels,

                                    class_names=classnames,

                                    filled=True, rounded=True,

                                    special_characters=True)

    graph = pydotplus.graph_from_dot_data(dot_data)

    colors = ('lightblue', 'green')

    edges = collections.defaultdict(list)

    for edge in graph.get_edge_list():

        edges[edge.get_source()].append(int(edge.get_destination()))

    for edge in edges:

        edges[edge].sort()

        for i in range(2):

            dest = graph.get_node(str(edges[edge][i]))[0]

            dest.set_fillcolor(colors[i])
```

```
    graph.write_png(pathname)




# Create Full Path - This is the OS agnostic way of doing so
dir_name = os.getcwd()
filename = 'AlienMushrooms.xlsx'
full_path = os.path.join(dir_name, filename)

# Create the Data Frame
df = pandas.read_excel(full_path) # read Excel spreadsheet
print('File {0} is of size {1}'.format(full_path, df.shape))
labels = df.columns
feature_labels = labels.drop("Edible")

features = df.drop(columns = ["Edible"])
target = df["Edible"]
target_unique = target.unique()

# Create the Decision Tree
clf = tree.DecisionTreeClassifier(criterion = "entropy", splitter = "best")
clf = clf.fit(features, target)
print("Training set score = ", clf.score(features, target))
#print("Test set score = ", clf.score(testX, testy))

# Generate the PNG
path_name = os.path.join(dir_name, "AlienMushroom_test.png")
```

```
writegraphtofile(clf, feature_labels, (target_unique[0], target_unique[1]), path_name)

tree.export_graphviz(clf)
```

Part 3 Code:

```
import pandas as pd

import os

from sklearn import tree

import pydotplus

import collections

import stats_report as sr


#%% Functions

# for a two-class tree, call this function like this:

# writegraphtofile(clf, ('F', 'T'), dirname+graphfilename)

def writegraphtofile(clf, feature_labels, classnames, pathname):

   dot_data = tree.export_graphviz(clf, out_file=None,

                           feature_names=feature_labels,

                           class_names=classnames,

                           filled=True, rounded=True,

                           special_characters=True)

   graph = pydotplus.graph_from_dot_data(dot_data)

   colors = ('lightblue', 'green')

   edges = collections.defaultdict(list)

   for edge in graph.get_edge_list():

      edges[edge.get_source()].append(int(edge.get_destination()))

   for edge in edges:

      edges[edge].sort()
```

Andrew Garcia
3/1/2022

```
        for i in range(2):
            dest = graph.get_node(str(edges[edge][i]))[0]
            dest.set_fillcolor(colors[i])
    graph.write_png(pathname)


#%% Setup
# Create Full Path - This is the OS agnostic way of doing so
dir_name = os.getcwd()
filename = 'FlareData.xlsx'
full_path = os.path.join(dir_name, filename)


#
# Create the Data Frame
#
df = pd.read_excel(full_path) # read Excel spreadsheet
print('File {0} is of size {1}'.format(full_path, df.shape))
labels = df.columns
#feature_labels = labels.drop(["C class", "M class", "X class"])

#features = df[feature_labels]
#target = df["C class"]


#%% Simple Stats
#
# Getting Simple Stats based on HW1
#
report = sr.StatsReport()
```

Andrew Garcia
3/1/2022

```
# Create a simple data set summary for the console

for thisLabel in labels: # for each column, report stats

    thisCol = df[thisLabel]

    report.addCol(thisLabel, thisCol)


print(report.to_string())

#report.statsdf.to_excel("FlareData_Report.xlsx")


#%% Preprocessing

from sklearn.preprocessing import OrdinalEncoder


# Identify the Unique values of Ordinal Data

zurich_class_unique = pd.DataFrame(df["Zurich Class"].unique())

spot_size_unique = pd.DataFrame(df["Spot Size"].unique())

spot_dist_unique = pd.DataFrame(df["Spot Dist"].unique())


# Encode the Ordinal Data

encoder = OrdinalEncoder()

zur_class_encoded = pd.DataFrame(encoder.fit_transform(zurich_class_unique))

spot_size_encoded = pd.DataFrame(encoder.fit_transform(spot_size_unique))

spot_dist_encoded = pd.DataFrame(encoder.fit_transform(spot_dist_unique))


df_encoded = df.copy()


for idx in range(len(zur_class_encoded)):

    old_value = zurich_class_unique[0][idx]
```

```
    new_value = int(zur_class_encoded[0][idx])
    df_encoded["Zurich Class"] = df_encoded["Zurich Class"].replace(old_value,int(new_value))

for idx in range(len(spot_size_encoded)):
    old_value = spot_size_unique[0][idx]
    new_value = spot_size_encoded[0][idx]
    df_encoded["Spot Size"] = df_encoded["Spot Size"].replace(old_value,int(new_value))

for idx in range(len(spot_dist_encoded)):
    old_value = spot_dist_unique[0][idx]
    new_value = spot_dist_encoded[0][idx]
    df_encoded["Spot Dist"] = df_encoded["Spot Dist"].replace(old_value,int(new_value))

# Combining the new and old values into one
zur_class_encoded['Old Values'] = zurich_class_unique
spot_size_encoded['Old Values'] = spot_size_unique
spot_dist_encoded['Old Values'] = spot_dist_unique

print(f"[DATA after Ordinals are Encoded] \n{df_encoded}")
print(f"[Zurich Class encoder key] \n{zur_class_encoded}")
print(f"[Spot Size encoder key] \n{spot_size_encoded}")
print(f"[Spot Dist encoder key] \n{spot_dist_encoded}")

target = df_encoded["C class"]
target_unique = target.unique()

labels = df_encoded.columns
```

Andrew Garcia
3/1/2022

```
feature_labels = labels.drop(["C class", "M class", "X class"])

features = df_encoded[feature_labels]


#%% Decision Tree Entropy


clf_entropy = tree.DecisionTreeClassifier(criterion = "entropy", max_depth = 4)

clf_entropy = clf_entropy.fit(features, target)

print("Training set score = ", clf_entropy.score(features, target))

#print("Test set score = ", clf.score(testX, testy))


path_name = os.path.join(dir_name, "FlareData_DecisionTree_Entropy.png")

writegraphtofile(clf_entropy, feature_labels, (str(target_unique[0]), str(target_unique[1]),
str(target_unique[2])), path_name)

tree.export_graphviz(clf_entropy)


#%% Decision Tree Entropy


clf_gini = tree.DecisionTreeClassifier(criterion = "gini", max_depth = 4)

clf_gini = clf_gini.fit(features, target)

print("Training set score = ", clf_gini.score(features, target))

#print("Test set score = ", clf.score(testX, testy))


path_name = os.path.join(dir_name, "FlareData_DecisionTree_Gini.png")

writegraphtofile(clf_gini, feature_labels, (str(target_unique[0]), str(target_unique[1]),
str(target_unique[2])), path_name)

tree.export_graphviz(clf_gini)
```