Efficient AI

ECE 595

Project Part A

Due Date: April 3, 2024

Project Part A – Entropy, Information Gain and Decision Tree

Introduction

This project had two sections including 1) Feature Ranking and 2) Classifier Building using Information Gain. I decided to utilize the given dataset due to the extensive work and examples covered in class. In this project report, I will dive into the code and mathematics I had to utilize to get the project output.

Data

I utilized the data provided to me by the professor. This is the table that has binary classes – Yes and No and has the attributes of Outlook, Temperature, Humidity, and Wind. Accordingly, to those conditions the classes determined the decision to either play (yes) or not (no.)

$\langle x_i, y_i \rangle$

Predictors				Response
Outlook	**Temperature**	**Humidity**	**Wind**	**Class**
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

This data was written as:

```
data = {
'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast',
'Overcast', 'Rain'],
'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal',
'High'],
'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak',
'Strong'],
```

```
'Class': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}
```

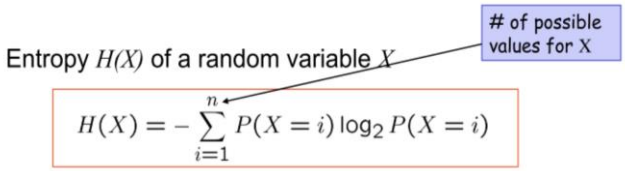
Using the Pandas library, the dataset was converted into a DataFrame which allows the easy manipulation of given data. I separated the data into two 1) Predictors – All the columns but Class column and 2) Response – Just the class column alone, just as it is mentioned in given dataset above. All manipulations were done on the Predictors and Response data.

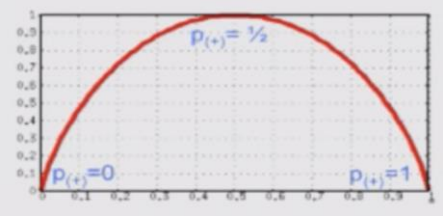
1] Feature Ranking

The two functions that were essential were:

1) calculate_entropy():

This function calculates the entropy of any input column considered the random variable at hand. First, the individual labels are perceived for example – the feature "Outlook" has 3 unique labels - "Sunny", Overcast" and "Rain". These are the possible values of X (random variable) in the following equation. Here entropy is a probability so it will have a value between 0 (minimum impurity) and 1 (maximum impurity.) Entropy for each attribute will allow the calculation of the next function I have created since it is an integral part of calculating information gain.





With respect to my code: for each unique label in the column, it calculated the proportion of each label in the attribute column and add the negation to the previously calculated value for the entropy to start at 0 and then have the final value. The log helps negate the "-" sign allowing entropy to be a positive value between 0 and 1.

2) calculate_information_gain():

This function takes two inputs, one is the parent data and the target data to calculate the information gain when the target is chosen. This is calculated by making calls and retrieving the entropy using the previous function.

```
IG = entropy(parent) - [average entropy(children)]
```

This function calculates the entropy of the parent which in this can is somewhat around 0.9402 and then goes through each attribute (feature) which each starts at 0 until the values of each unique attribute is attained. Then I create a variable called subset that stores the parent columns where the target variable matches the values at hand. To this we increase the feature entropy by multiplying the weight (subset length divided by the parent length) * feature entropy. This adds the entropy to each index of the feature entropy array. Next, the information gain is calculated using the feature entropy array along with the parent entropy. Therefore, each feature has an individual information gain if it were chosen in a potential decision tree.

According to the results, of my code:

Feature: Temperature, Information Gain: 0.02922256565895487

Feature: Wind, Information Gain: 0.04812703040826949

Feature: Humidity, Information Gain: 0.15183550136234159

Feature: Outlook, Information Gain: 0.24674981977443933

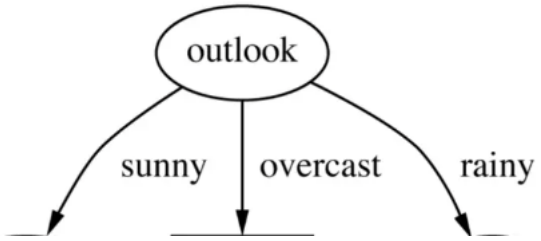
Outlook has the highest information gain and that makes sense since its values are Sunny, Overcast, and Rain that consists of the most information if that node is chosen to split on. This happens since the weight of each value multiplied by the entropy gives the highest Information Gain.

Part 2: Classifier Building based on Information Gain

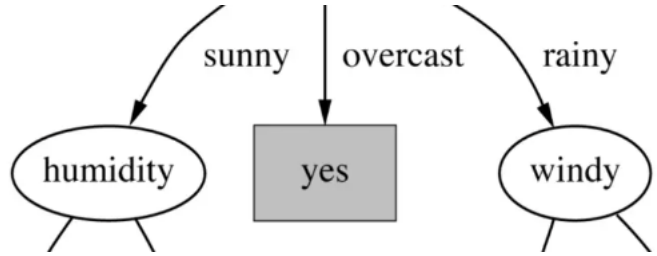
Now that I have created the functions of entropy and information gain (uses entropy) we can move forward to the main goal of this area of study – Classifying using Decision trees. This is a type of supervised machine learning used to categorize or make predictions based on how a previous set of questions were answered.

Here I have one function that is called recursively names recursive_decision_tree(). This can be done by iteration and recursion, but I felt with respect to trees, recursion prevails in the scope of implementation. Here, the functions starts with collecting all the information agains for anything under that Node. At the start it would be the parent would be the whole dataset and the target would be the attributes like Outlook, Wind, Temp, and Humidity. So part1 of this project is kind of repeated. This implementation becomes important when the children of these nodes are considered.

After retrieval of the respective information gains, the max is selected and splitting is done on that. For example, the first set of information gains, outlook is the highest, therefore thats where splitting occurs.

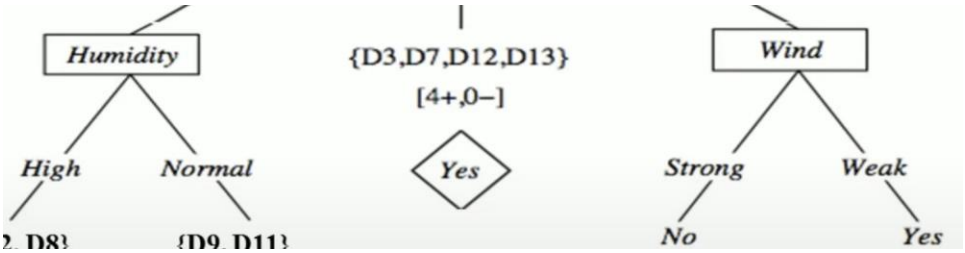


For different levels of the tree, I printed out the node according to what level the tree was at. Then after the splitting, I introduced a recursive loop that would go through each subset of the tree extract the best split node and if a clear class cannot be determined, then recursively call the function.

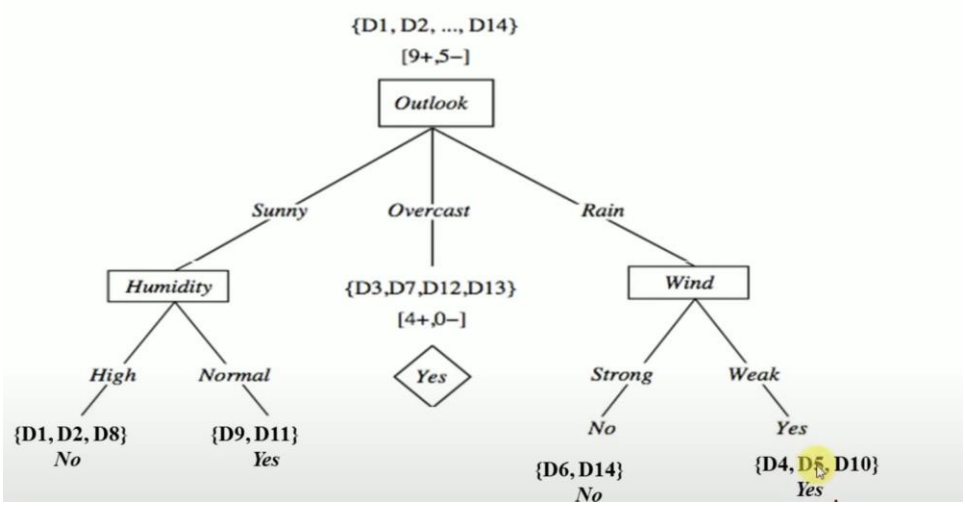


This can be seen here. After Outlook splits, each subset checks if the entropy is pure enough to classify. If it cannot, then that the best split node is considered by going deep into that node (esentially dropping the best feature node.) This occurs in sunny and rainy whereas overcast is pure enough to classify “yes.” When there is a discrepancy and no clear winner, child nodes are identified and function is called again. Here, given that Outlook is “Sunny”, Humidity has the highest information gain and similarly given that Outlook is “Rainy”, Wind has the highest gain therefore, the tree splits of those nodes.

This keeps happening until data cannot be split any further:



Now, finally all have identified the class and the decision tree has been made.



Code Output:

```
aayushmailarpwar@Aayushs-MacBook-Pro ~ % /usr/local/bin/python3
roject_part_a.py
Rank:1 - Outlook, IG: 0.24674981977443933
Rank:2 - Humidity, IG: 0.15183550136234159
Rank:3 - Wind, IG: 0.04812703040826949
Rank:4 - Temperature, IG: 0.02922256565895487


 Decsion Tree using Recursion:


 -  Best Split using IG: Outlook :  0.24674981977443933
 -  -  Value: Sunny
 -  -  -  Best Split using IG: Humidity :  0.9709505944546686
 -  -  -  -  Value: High
 -  -  -  -  -  Class: No
 -  -  -  -  Value: Normal
 -  -  -  -  -  Class: Yes
 -  -  Value: Overcast
 -  -  -  Class: Yes
 -  -  Value: Rain
 -  -  -  Best Split using IG: Wind :  0.9709505944546686
 -  -  -  -  Value: Weak
 -  -  -  -  -  Class: Yes
 -  -  -  -  Value: Strong
 -  -  -  -  -  Class: No
```

Appendix

https://github.com/aayushmailarpwar/Decision-Tree-Implementation-with-Information-Gain

README - https://github.com/aayushmailarpwar/Decision-Tree-Implementation-with-Information-Gain/blob/main/README.md

CODE - https://github.com/aayushmailarpwar/Decision-Tree-Implementation-with-Information-Gain/blob/main/project_part_a.py

Code:

```
import pandas as pd
import numpy as np

# Feature Ranking
```

```
# Function for finding the entropy using input data
def calculate_entropy(input_data):
# initialize entropy
entropy = 0
# eg. Label is either Yes or No in case of Response
labels = input_data.unique()
for label in labels:
# calculate proportion based on number of occurance of example "Sunny" label in "Outlook" attribute
proportion = len(input_data[input_data == label]) / len(input_data)
# subtact with multiplier from entropy calculation
entropy -= proportion * np.log2(proportion)
# now we have entropy for full attribute
return entropy

# IG = entropy(parent) - [average entropy(children)]
def calculate_information_gain(input_data, parent):

# total samples and entropy(parent)
#print("Total Samples:" , len(parent))
#print("Starting Entropy of Parent:", calculate_entropy(parent))

# Find unique class Yes or No
labels = parent.unique()
# for label in labels:
# proportion = len(parent[parent == label])
# print(label, ":", proportion)
# print("\n")
# empty array to store all the entropies
feature_entropy = {}

# Go through each feature
for feature in input_data:
#print("Running analysis on Feature:", feature)
feature_entropy[feature] = 0
# values of the feature target is now identified - "Humidity"'s values
values = input_data[feature].unique()
for value in values:
subset = parent[input_data[feature] == value]
feature_entropy[feature] += len(subset) / len(parent) * calculate_entropy(subset)

information_gain = {}
for feature in feature_entropy:
information_gain[feature] = calculate_entropy(parent) - feature_entropy[feature]
return information_gain

# Given the data from Lecture slides
data = {
```

```
'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast',
'Overcast', 'Rain'],
'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal',
'High'],
'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak',
'Strong'],
'Class': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}

# Using Pandas library to convert above dataset to DataFrame
df = pd.DataFrame(data)

# Separate features and target variable
predictors = df.drop(columns=['Class'])
response = df['Class']



# Call IG function
information_gain_ = calculate_information_gain(predictors, response)

# Rank features based on information gain
ranked_features = sorted(information_gain_, key=information_gain_.get, reverse=True)

# Display the ranked features and their information gain
index = 0
for feature in ranked_features:
index+=1
print(f"Rank:{index} - {feature}, IG: {information_gain_[feature]}")

print("\n\n Decsion Tree using Recursion: \n\n")

# Part 2
#
def recurisve_decision_tree(data, target, current_level=1):
# calculate the info gain with respect to data and target
# info_gains = calculate_information_gain(data, target)
# max is the best splitting node
best_feature_node = max(calculate_information_gain(data, target), key=calculate_information_gain(data, target).get)
# print splitter node
print(" - " * current_level, "Best Split using IG:", best_feature_node, ": ", calculate_information_gain(data,
target)[best_feature_node])
# recursive loop to build child nodes after splitting
for value in data[best_feature_node].unique():
# extracting the subset of data where the current feature equals the current value
subset = data[data[best_feature_node] == value]
# extracting the corresponding subset of target values
subset_target = target[data[best_feature_node] == value]
```

```
print(" - " * (current_level+1), "Value:", value)

# making sure dp belongs to the same class
if len(subset_target.unique()) == 1:
# print the class
print(" - " * (current_level+2), "Class:", subset_target.iloc[0])
else:
# if more classes prevail, split again to find child nodes
childTree = subset.drop(columns=[best_feature_node])
# !!! recursive call !!!
recurisve_decision_tree(childTree, subset_target, current_level+2)

recurisve_decision_tree(predictors,response)
```

References

1. Codebasics, "Machine Learning Tutorial Python - 5: Decision Tree Classification," YouTube, [Online]. Available: https://www.youtube.com/watch?v=coOTEc-0OGw&t=227s.

2. "NumPy Documentation," numpy.org. [Online]. Available: https://numpy.org/.

3. "Pandas Documentation," pandas.pydata.org. [Online]. Available: https://pandas.pydata.org/.