# ID3 Decision Tree Algorithm

DWDM Lab 7 - 180911202

Nikesh Kumar

May 29, 2021

## Contents

# 1 Decision Tree Algorithm

---

## 1.1 Algorithm Implementation

[1]:
```
import pandas as pd
import numpy as np
import math
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.metrics import accuracy_score
```

[2]:
```
class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""

# Finding Entropy for every attribute
def entropy(train_set: pd.DataFrame, labels: dict):
    pos = 0.0
    neg = 0.0
    for _, row in train_set.iterrows():
        if row["labels"] == labels["positive"]:
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))

# Finding Info Gain for every attribute
def info_gain(train_set: pd.DataFrame, attr: list, labels: dict):
    unique_values = np.unique(train_set[attr])
    gain = entropy(train_set,labels)
    for unique_value in unique_values:
        subdata = train_set[train_set[attr] == unique_value]
        sub_e = entropy(subdata,labels)
        gain -= (float(len(subdata)) / float(len(train_set))) * sub_e
    return gain
```

```python
# Selecting best attribute from attrb list
def attribute_selection(train_set: pd.DataFrame, attrs: list, labels: dict):
    max_gain = 0
    max_feat = ""
    for feature in attrs:
        gain = info_gain(train_set, attrs, labels)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    return max_feat,max_gain

# ID3 decision tree implementation
def ID3(train_set: pd.DataFrame, attrs: list, labels: dict):
    root = Node()
    max_feat,max_gain = attribute_selection(train_set,attrs,labels)
    root.value = max_feat
    unique_values = np.unique(train_set[max_feat])
    for unique_value in unique_values:
        subdata = train_set[train_set[max_feat] == unique_value]
        if entropy(subdata,labels) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = unique_value
            newNode.pred = np.unique(subdata["labels"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = unique_value
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs, labels)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

def predict(test_tuple: pd.Series, model: Node,match:bool = False):
    if model.isLeaf:
```

```
        return model.pred
    if match:
        for child in model.children:
            return predict(test_tuple,child,match=False)
    else:
        test_value = test_tuple[model.value]
        for child in model.children:
            if child.value == test_value:
                return predict(test_tuple,child,match=True)

def predict_values(test_dataset: pd.DataFrame, model: Node,map_dict: dict):
    y_pred = []
    for index,test_tuple in test_dataset.iterrows():
        # print(predict(test_tuple,model))
        predicted_value = predict(test_tuple,model)
        if predicted_value != None:
            y_pred.append(predicted_value[0])
        else:
            print("Change TRAINING DATASET")
            break
    return list(map(map_dict.get,y_pred))
```

## 1.2 Dataset I : Identifying Edible and Poisonous Mushrooms

https://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agaricus-lepiota.data

### 1.2.1 Dataset Preparation

[3]:
```
features = ["labels","cap-shape","cap-surface","cap-color","bruises?
 ↪","odor","gill-attachment","gill-spacing","gill-size","gill-colot","stalk-shape
","stalk-root","stalk-surface-above-ring","stalk-surface-below-ring","
stalk-color-above-ring","stalk-color-below-ring","veil-type","veil-color
","ring-number","ring-type","spore-print-color","population","habitat"]
dataset = pd.read_csv("./agaricus-lepiota.data",names=features,nrows=1000)
features.remove("labels")
dataset.head()
```

[3]:

| | labels | cap-shape | cap-surface | ... | habitat |
|---|---|---|---|---|---|
| 0 | p | x | s | ... | u |
| 1 | e | x | s | ... | g |
| 2 | e | b | s | ... | m |
| 3 | p | x | y | ... | u |
| 4 | e | x | s | ... | g |

[5 rows x 23 columns]

[4]:
```
class_labels = np.unique(dataset["labels"])
class_labels = {
    "positive": class_labels[1],
    "negative": class_labels[0]
}
class_labels
```

[4]:
```
{'positive': 'p', 'negative': 'e'}
```

### 1.2.2 Generating training and testing data

[5]:
```
X = dataset
y = dataset["labels"]
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.
 ↪25,stratify=y)
```

### 1.2.3 Running the ID3 Algorithm

[6]:
```
model1 = ID3(X_train, features,class_labels)
printTree(model1)
```

```
cap-shape
        b ->  ['e']
        f
                cap-surface
                        f ->  ['e']
                        s
                                cap-color
                                        g ->  ['e']
                                        n
                                                bruises?
                                                        f ->  ['e']
                                                        t ->  ['p']
                                        w
                                                bruises?
                                                        f ->  ['e']
                                                        t
                                                                odor
                                                                        a -> ['e']
                                                                        l -> ['e']
                                                                        p -> ['p']
                                        y ->  ['e']
                        y
                                cap-color
                                        n
                                                bruises?
                                                        t
                                                                odor
                                                                        a -> ['e']
                                                                        l -> ['e']
                                                                        p -> ['p']
                                        w ->  ['p']
                                        y ->  ['e']
        s ->  ['e']
        x
                cap-surface
                        f ->  ['e']
                        s
                                cap-color
                                        g ->  ['e']
                                        n
                                                bruises?
```

```
                                f ->  ['e']
                                t ->  ['p']
                        w
                                bruises?
                                        f ->  ['e']
                                        t
                                                odor
                                                        a -> ['e']
                                                        l -> ['e']
                                                        p -> ['p']

                        y ->  ['e']
y
        cap-color
                        e ->  ['e']
                        g ->  ['e']
                        n
                                bruises?
                                        t
                                                odor
                                                        a -> ['e']
                                                        l -> ['e']
                                                        n -> ['e']
                                                        p -> ['p']
                        w
                                bruises?
                                        t
                                                odor
                                                        a -> ['e']
                                                        l -> ['e']
                                                        p -> ['p']
                        y ->  ['e']
```

[7]:
```
map_dict = {"p":1,"e":0}
y_pred = predict_values(X_train,model1,map_dict)
y_training_data = list(map(map_dict.get,y_train))

print("Training Absolute Error:␣
 ↪",mean_absolute_percentage_error(y_training_data,y_pred))
print("Training Accuracy Score: ",accuracy_score(y_training_data,y_pred))
pd.DataFrame(confusion_matrix(y_training_data,y_pred),index=["true:yes","true:
 ↪no"],columns=["pred:yes","pred:no"])
```

```
Training Absolute Error:  0.0
Training Accuracy Score:  1.0
```

[7]:

| | pred:yes | pred:no |
|---|---|---|
| true:yes | 673 | 0 |
| true:no | 0 | 77 |

[8]:
```
map_dict = {"p":1,"e":0}
y_pred = predict_values(X_test,model1,map_dict)
y_training_data = list(map(map_dict.get,y_test))

print("Testing Absolute Error:␣
 ↪",mean_absolute_percentage_error(y_training_data,y_pred))
print("Testing Accuracy Score: ",accuracy_score(y_training_data,y_pred))
pd.DataFrame(confusion_matrix(y_training_data,y_pred),index=["true:yes","true:
 ↪no"],columns=["pred:yes","pred:no"])
```

```
Testing Absolute Error:  0.0
Testing Accuracy Score:  1.0
```

[8]:

| | pred:yes | pred:no |
|---|---|---|
| true:yes | 225 | 0 |
| true:no | 0 | 25 |

## 1.3 Dataset II: Identifying if it is a good day to play

https://drive.google.com/file/d/1W2R-A5VwckJJ1kZaSroCVJOFzkuIvagS/view

### 1.3.1 Dataset Preparation

[9]:
```
dataset = pd.read_csv("./game.csv")
dataset.rename({"answer":"labels"},axis=1,inplace=True)
features = [feature for feature in dataset]
features.remove("labels")
dataset.head()
```

[9]:

| | outlook | temperature | humidity | wind | labels |
|---|---|---|---|---|---|
| 0 | sunny | hot | high | weak | no |
| 1 | sunny | hot | high | strong | no |
| 2 | overcast | hot | high | weak | yes |
| 3 | rain | mild | high | weak | yes |
| 4 | rain | cool | normal | weak | yes |

[10]:
```
class_labels = np.unique(dataset["labels"])
class_labels = {
    "positive": class_labels[1],
    "negative": class_labels[0]
}
class_labels
```

[10]:
```
{'positive': 'yes', 'negative': 'no'}
```

### 1.3.2 Generating training and testing data

[15]:
```
X = dataset
y = dataset["labels"]
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2)
```

---

### 1.3.3 Running the ID3 Algorithm

[16]:
```
model2 = ID3(X_train, features,class_labels)
printTree(model2)
```

```
outlook
        overcast ->  ['yes']
        rain
                temperature
                        cool ->  ['no']
                        mild
                                humidity
```

```
                                        high
                                                wind
                                                        strong ->  ['no']
                                                        weak ->  ['yes']
                                        normal ->  ['yes']
        sunny
                temperature
                        cool ->  ['yes']
                        hot ->  ['no']
                        mild
                                humidity
                                        high ->  ['no']
                                        normal ->  ['yes']
```

[17]:
```
map_dict = {"yes":1,"no":0}
y_pred = predict_values(X_train,model2,map_dict)
y_training_data = list(map(map_dict.get,y_train))

print("Training Absolute Error:␣
 ↪",mean_absolute_percentage_error(y_training_data,y_pred))
print("Training Accuracy Score: ",accuracy_score(y_training_data,y_pred))
pd.DataFrame(confusion_matrix(y_training_data,y_pred),index=["true:yes","true:
 ↪no"],columns=["pred:yes","pred:no"])
```

```
Training Absolute Error:  0.0
Training Accuracy Score:  1.0
```

[17]:

| | pred:yes | pred:no |
|---|---|---|
| true:yes | 4 | 0 |
| true:no | 0 | 7 |

[18]:
```
map_dict = {"yes":1,"no":0}
y_pred = predict_values(X_test,model2,map_dict)
y_training_data = list(map(map_dict.get,y_test))

print("Testing Absolute Error:␣
 ↪",mean_absolute_percentage_error(y_training_data,y_pred))
print("Testing Accuracy Score: ",accuracy_score(y_training_data,y_pred))
pd.DataFrame(confusion_matrix(y_training_data,y_pred),index=["true:yes","true:
 ↪no"],columns=["pred:yes","pred:no"])
```

```
Testing Absolute Error:  0.3333333333333333
Testing Accuracy Score:  0.6666666666666666
```

[18]:

| | pred:yes | pred:no |
|---|---|---|
| true:yes | 1 | 0 |
| true:no | 1 | 1 |