Assignment 3, Carlos Figueroa (cdf5579) Machine Learning, Fall 2023

In this notebook, we will work with the dataset: https://archive.ics.uci.edu/ml/datasets/Spambase in order to study different methods for Desicion trees.

Now, let's start with some analysis on the data, and cleaning procedures.

Part 1: Spam email classification using Decision Trees (50 Points)

In this part, you must classify the above data set using Decision Trees. The code must be written in Python and you can use any Python package to solve the question. You must report the best classification accuracy achieved across 20 different seeds. The accuracies must be reported for the decision tree created using (1) Gini impurity, and (2) Shannon information gain (Shannon I.G. refers to the Shannon Information Gain.). The program script for this part must be named ⟨NetID⟩ hw4 part1.py.

In [85]:
```
#Install packages
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from matplotlib.dates import DateFormatter
import matplotlib.ticker as mtick
#import matplotlib as plt
```

In [86]:
```
#load the dataframe
df = pd.read_csv("spambase.data", names = range(1,59))
#column 58 is the target
```

In [87]:
```
#lets bring those column names, we know this from HW3
df.columns  = ["word_freq_make", "word_freq_address", "word_freq_all", "word_freq_3d",
                        "word_freq_our", "word_freq_over", "word_freq_remove", "word_freq_ir
                        "word_freq_order", "word_freq_mail", "word_freq_receive", "word_freq
                        "word_freq_people", "word_freq_report", "word_freq_addresses", "word
                        "word_freq_business", "word_freq_email", "word_freq_you", "word_freq
                        "word_freq_your", "word_freq_font", "word_freq_000", "word_freq_mone
                        "word_freq_hpl", "word_freq_george", "word_freq_650", "word_freq_lab
                        "word_freq_telnet", "word_freq_857", "word_freq_data", "word_freq_41
                        "word_freq_technology", "word_freq_1999", "word_freq_parts", "word_f
                        "word_freq_cs", "word_freq_meeting", "word_freq_original", "word_fre
                        "word_freq_edu", "word_freq_table", "word_freq_conference", "char_fr
                        "char_freq_[", "char_freq_!", "char_freq_$", "char_freq_hash", "capi
                        "capital_run_length_longest", "capital_run_length_total", "target"]
```

Now lets talk a little about what we are about to do

First, lets break it into parts and standarize our X components

In [88]:
```
#lets break it
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
Y = df["target"] #just need the target column for Y

#three last variables
X = df.drop(["target"], axis=1)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state =
```

In [89]:

```
#now we standarize our training and test variables, not necessary for the Y's
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Now, lets import the packages, and apply the model accross 20 different seeds

Please note that the criterion is an input of the tree.DecisionTreeClassifier in the SKlearn model, so we will just create a loop to go through the 20 different seeds using one and the other.

Moreover, Shannon Information Gain is called Entropy when applying the SKlearn model. The term entropy (in information theory) goes back to Claude E. Shannon, and that's where the name comes from. What Criterion means as an input, is what is the Information Gain formula going to use as a parameter (either Gini or Shannon).

We are using the same seed for the split and tree, as mentioned in the recitation.

In [90]:

```
#packages needed
from sklearn import tree
from sklearn.metrics import accuracy_score

criterions = ['gini', 'entropy']

#best gini is set to zero
best_g = 0

#best Shannon information gain is also set to zero
best_sh = 0

for criterion in criterions:

    for seed in range(1,21):

        #we do the split and normalization
        X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)

        #lets start the model

        clf = tree.DecisionTreeClassifier(criterion = criterion, random_state = seed)
        clf = clf.fit(X_train,Y_train)
        pred = clf.predict(X_test)
        current_accuracy = accuracy_score(Y_test,pred)

        #Now, lets evaluate these results

        if criterion == "gini":
            if current_accuracy > best_g:
                best_g = current_accuracy
```

```
                print("Updates on best seed number using Gini Criterion:", seed)

        else: #which means that criterion is Shannon
            if current_accuracy > best_sh:
                best_sh = current_accuracy
                print("Updates on best seed number using Shannon IG Criterion:", seed)


print("--------------------------------------------------------------------------------")
print("Best test accuracy result obtained using Gini Criterion", best_g)
print("Best test accuracy result obtained using Shannon IG Criterion", best_sh)
```

```
Updates on best seed number using Gini Criterion: 1
Updates on best seed number using Gini Criterion: 2
Updates on best seed number using Gini Criterion: 5
Updates on best seed number using Gini Criterion: 9
Updates on best seed number using Gini Criterion: 11
Updates on best seed number using Gini Criterion: 13
Updates on best seed number using Gini Criterion: 14
Updates on best seed number using Shannon IG Criterion: 1
Updates on best seed number using Shannon IG Criterion: 5
--------------------------------------------------------------------------------
Best test accuracy result obtained using Gini Criterion 0.9268645908761767
Best test accuracy result obtained using Shannon IG Criterion 0.9319333816075308
```

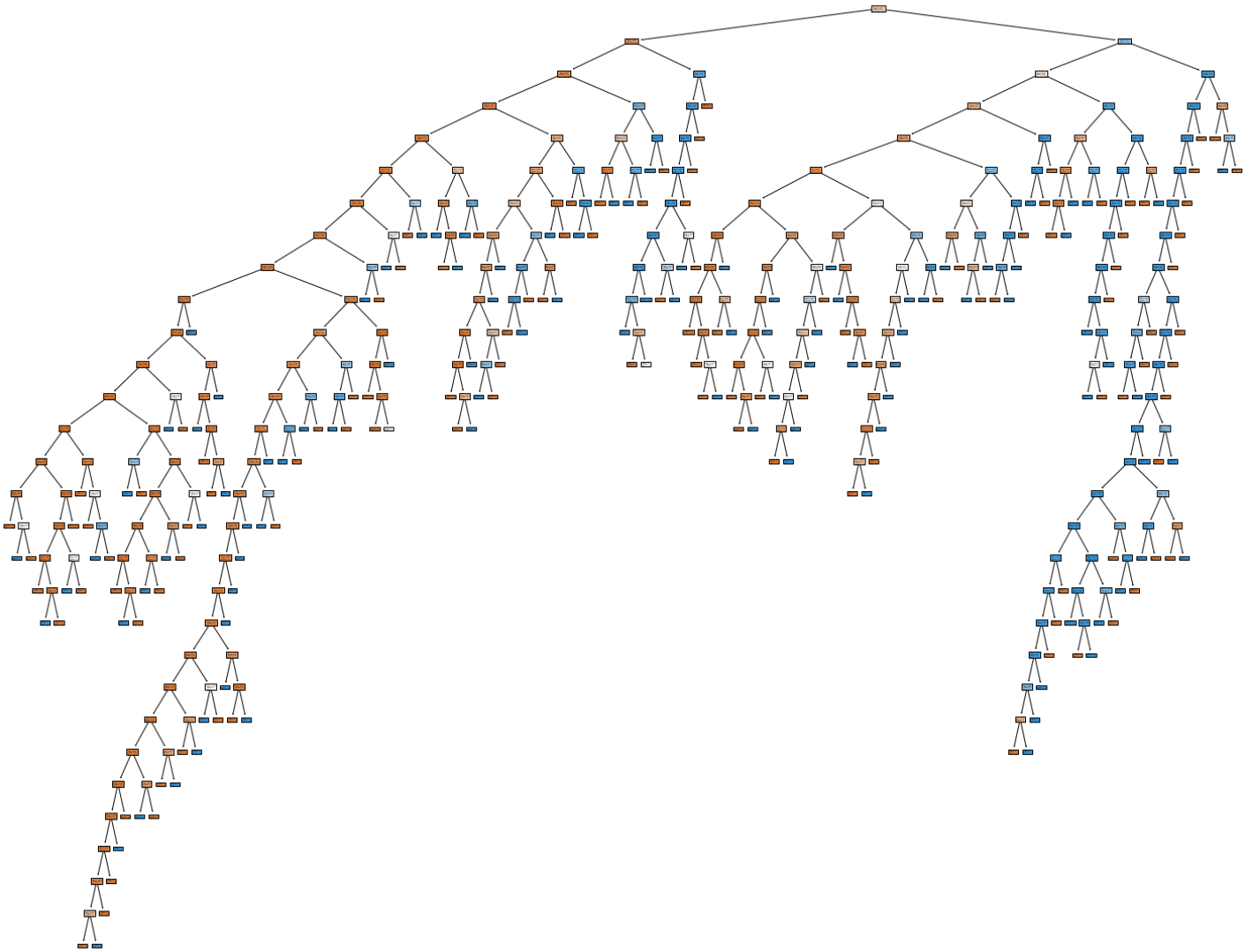
Then, we see that using the 14 random state for Gini, and 5 random state for Shannon IG, we obtain the highest accuracy results for both models. Now that we know the best random state for these models, lets plot them and compare them.

In [91]:

```
#Lets see these threes

clf = tree.DecisionTreeClassifier(criterion = "gini", random_state = 14)
clf = clf.fit(X_train,Y_train)
pred = clf.predict(X_test)
acc = accuracy_score(Y_test,pred)

fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf, filled=True)
```

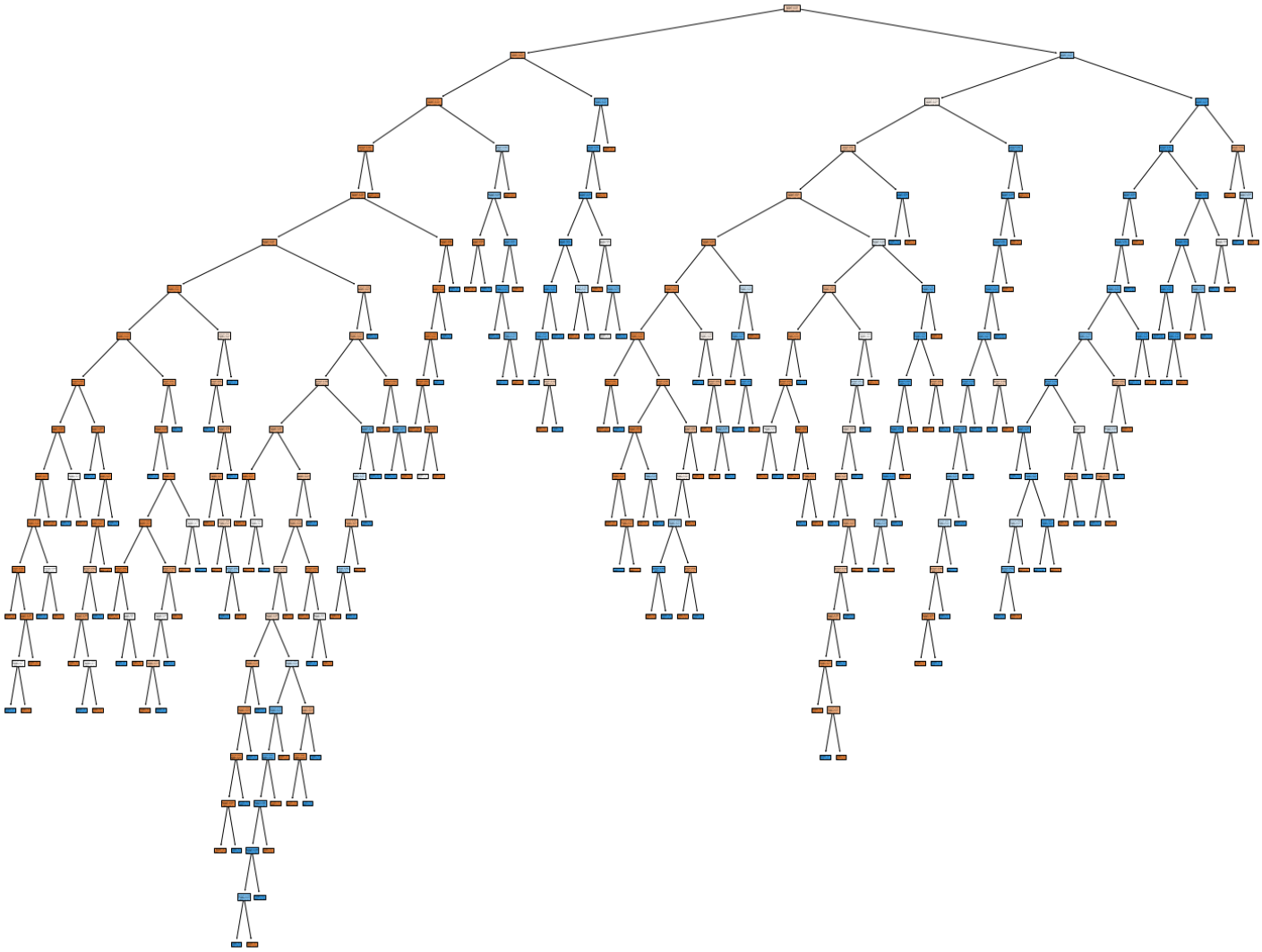


In [92]:
```
print(clf.tree_.node_count)
```

```
429
```

In [93]:
```
clf = tree.DecisionTreeClassifier(criterion = "entropy", random_state = 5)
clf = clf.fit(X_train,Y_train)
pred = clf.predict(X_test)
acc = accuracy_score(Y_test,pred)

fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf, filled=True)
```



In [94]:
```
print(clf.tree_.node_count)
```

```
353
```

By seeing the trees, and their size, we can see that the model that used Gini as criterion, is denser and larger in node count than the one using Shannon IG. This is interesting because at the same time, Shannon IG has a better accuracy rate than the Gini one, even though it uses less nodes to achieve that result, hinting the this might be the right model to look at this data with. Moreover, both seem to be skewed to the left, which might be associate with the nature of the variables in our dataset.

Now, lets try different random states

In [95]:
```
options = ['gini', 'entropy']

#best gini is set to zero
best_g = 0

#best Shannon information gain is also set to zero
best_sh = 0


for criterion in options:
    for option in range(1,100):

        #we do the split and normalization
        X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_
```

```
        scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)

        #lets start the model
        clf = tree.DecisionTreeClassifier(criterion = criterion, random_state = option)
        clf = clf.fit(X_train,Y_train)
        pred = clf.predict(X_test)
        acc = accuracy_score(Y_test,pred)

        if criterion == "gini":
            if acc > best_g:
                best_g = acc
                print("Updates on best seed number using Gini Criterion:", option)
        else:
            if acc > best_sh:
                best_sh = acc
                print("Updates on best seed number using Shannon IG Criterion:", option)


print("--------------------------------------------------------------------------------")
print("Best test accuracy result obtained using Gini Criterion", best_g)
print("Best test accuracy result obtained using Shannon IG Criterion", best_sh)
```

```
Updates on best seed number using Gini Criterion: 1
Updates on best seed number using Gini Criterion: 2
Updates on best seed number using Gini Criterion: 5
Updates on best seed number using Gini Criterion: 9
Updates on best seed number using Gini Criterion: 11
Updates on best seed number using Gini Criterion: 13
Updates on best seed number using Gini Criterion: 14
Updates on best seed number using Gini Criterion: 98
Updates on best seed number using Shannon IG Criterion: 1
Updates on best seed number using Shannon IG Criterion: 5
--------------------------------------------------------------------------------
Best test accuracy result obtained using Gini Criterion 0.9326574945691528
Best test accuracy result obtained using Shannon IG Criterion 0.9319333816075308
```

Interesting enought, higher random states did affect our model with the Gini Criterion, while the Shannon Criterion did not improve(from our first analysis). now both models are very close, and Gini is actually better.

Provide an intuition for the results observed for the different hyperparameters used.

The higher the seed, the better the accuracy of the model. And the Shannon model is better under low levels of seeds, and Gini is better when seed is larger. This makes sense because in Decision Tree Classifier or Regression, when we want to find the best features that controls the randomness of splitting nodes, random_state is helpful. It will describe the structure of the tree. And that could be one of the reasons why Gini is larger in terms of nodes.

Part 2: Spam email classification using Random Forests (50 Points)

In this part, you must classify the above data set using Random Forests. The code must be written in Python and you can use any Python package to solve the question. For this part, you must fill up the following table with the best classification accuracy achieved across 20 different seeds.

Shannon I.G. refers to the Shannon Information Gain. Provide an intuition for the results observed for the different hyperparameters used. The program script for this part must be named ⟨NetID⟩ hw4 part2.py.

In [53]:
```
#Install packages
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from matplotlib.dates import DateFormatter
import matplotlib.ticker as mtick
#import matplotlib as plt
```

In [54]:
```
#load the dataframe
df = pd.read_csv("spambase.data", names = range(1,59))
#column 58 is the target
```

In [55]:
```
#lets bring those column names, we know this from HW3
df.columns  = ["word_freq_make", "word_freq_address", "word_freq_all", "word_freq_3d",
                        "word_freq_our", "word_freq_over", "word_freq_remove", "word_freq_in
                        "word_freq_order", "word_freq_mail", "word_freq_receive", "word_freq
                        "word_freq_people", "word_freq_report", "word_freq_addresses", "word
                        "word_freq_business", "word_freq_email", "word_freq_you", "word_freq
                        "word_freq_your", "word_freq_font", "word_freq_000", "word_freq_mone
                        "word_freq_hpl", "word_freq_george", "word_freq_650", "word_freq_lab
                        "word_freq_telnet", "word_freq_857", "word_freq_data", "word_freq_41
                        "word_freq_technology", "word_freq_1999", "word_freq_parts", "word_f
                        "word_freq_cs", "word_freq_meeting", "word_freq_original", "word_freq
                        "word_freq_edu", "word_freq_table", "word_freq_conference", "char_fr
                        "char_freq_[", "char_freq_!", "char_freq_$", "char_freq_hash", "capi
                        "capital_run_length_longest", "capital_run_length_total", "target"]
```

Lets break it into parts

In [56]:
```
#lets break it
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

Y = df["target"] #just need the target column for Y

#three last variables
X = df.drop(["target"], axis=1)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state =
```

In [57]:
```
#now we standarize our training and test variables, not necessary for the Y's
scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Now, lets import the packages, and apply the model accross 20 different seeds

Please note that the criterion is an input of the tree.DecisionTreeClassifier in the SKlearn model, so we will just create a loop to go through the 20 different seeds using one and the other.

Moreover, Shannon Information Gain is called Entropy when applying the SKlearn model. The term entropy (in information theory) goes back to Claude E. Shannon, and that's where the name comes from. What Criterion means as an input, is what is the Information Gain formula going to use as a parameter (either Gini or Shannon).

We are using the same seed for the split and tree, as mentioned in the recitation.

In [58]:
```
#packages needed
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier

criterions = ['gini', 'entropy']

estimators = [1,3,5,10,15,20,40,70]

#lets create a dictionary to save this data
best_model_accuracy = dict()

for criterion in criterions:

    #we create a key for the criterions
    best_model_accuracy[criterion] = {}

    #the set of different estimators to fit the model
    for estimator in estimators:

        #initialize the estimator key with the criterion corresponding
        model_accs[criterion][estimator] = 0

        #now we go over the 20 seeds, and pick the best accuracy of the 20 trials using ea
        for seed in range (1,21):

            #we do the split and normalization
            X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, ran
            X_train = scaler.fit_transform(X_train)
            X_test = scaler.transform(X_test)

            #Now the model
            clf = RandomForestClassifier(n_estimators = estimator, criterion = criterion,
            clf = clf.fit(X_train, Y_train)
            pred = clf.predict(X_test)
            current_accuracy = accuracy_score(Y_test, pred)

            #now we test if the lastest result with a different seed was better
            if current_accuracy > model_accs[criterion][estimator]:
                 best_model_accuracy[criterion][estimator] = current_accuracy


final_data = pd.DataFrame.from_dict(best_model_accuracy, orient = 'index')
```

In [59]:
```
final_data
```

Out[59]:

	1	3	5	10	15	20	40	70
gini	0.889935	0.918899	0.937726	0.947864	0.953657	0.957277	0.955829	0.956553
entropy	0.879073	0.934106	0.939899	0.948588	0.955105	0.955829	0.955829	0.957277

First, lets note that n_estimators stands fot the number of trees in the forest. Then, as the number of trees grows, it does not always mean the performance of the forest is significantly better than previous forests (fewer trees), and doubling the number of trees is sometimes worthless. It is also possible to state there is a threshold beyond which there is no significant gain, unless a huge computational environment is available. And that is sort of the behavior we see on this example: as the number of trees goes higher, the accuracy goes higher as well, but comparing 40 to 70 trees, this accuracy did not increase substantially, and it opens a good conversation into what is the benefit and compuational cost of adding more trees.

So the more trees, the better because it has more categories to subdivide the data and improve accuracy, but up to a certain point (in terms of price-cost computational analysis).

However, it has better accuracy than the trees from part 1 of the HW, with way less nodes involved in the process, which is interesting. Overall, is the best classification model we have produced in the class so far. Moreover, having a word on random_state, in Random Forest Classifier and Regression, random_state controls the randomness of the bootstrapping of the samples used when building trees and the sampling of the features to consider when looking for the best split at each node. So it also plays a descent role in improving the accuracy of the data (making it a little bit more bulletproof).