# Random Forest Classifiers

## Project Guide

## Project Overview

EXPECTED TIME: 2 HRS

This assignment uses a variety of decision tree based classifiers to attempt prediction of w hether or not a customer w ill default on their loans. Our data comes from the UCI machine learning Repository on default of credit card clients

Activities w ill include:

- Manipulating DataFrames
- Visualizing data
- Calculation of impurity measures
- Using `sklearn` 's tree and forest models
- Evaluate the effects of hyperparameter tuning

Motivation: Decision Trees and Forests offer easy to understand yet fairly advanced models w ith a variety of hyper-parameters to increase or decrease complexity to tune betw een bias and variance

Problem:
Given a number of personal variables, (sex, education, marriage status, age); and recent payment history, attempt to predict w hether or not a customer w ill default in the next month.

Data:Defaults of credit card clients

Please see above link for a complete description of the data.

```
Data Set Information:

This research aimed at the case of customersâ€™ default payments in Taiwan and compares the predictive accuracy of probability of default among six data mining methods. From the perspect


Attribute Information:

This research employed a binary variable, default payment (Yes = 1, No = 0), as the response variable. This study reviewed the literature and used the following 23 variables as explanato
X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
X2: Gender (1 = male; 2 = female).
X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
X4: Marital status (1 = married; 2 = single; 3 = others).
X5: Age (year).
X6 - X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repaymen
X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in
X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.
```

## Part 1: Acquire, Explore, and Preprocess Data

Import / Read in Data

```
# Import necessary packages
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#
%matplotlib inline

# Read in Data
df = pd.read_excel("../resource/asnlib/publicdata/default of credit card clients.xls", header = 1)

df.rename(columns = {"PAY_0":"PAY_1"}, inplace = True) #renaming mis-named column
df.head()
```

Explore the Data

Check for nulls

```
print("Data Shape: " , df.shape, "\n")
df.info()
```

Appears no null data.

Investigate Categorical Variables

Question 1:

The "`default payment next month`" w ill be the target used for classification. Investigate its distribution for the question below.

```
### GRADED

### Assign an int to `ans0` corresponding to the total number of non-default records
### Assign an int to `ans1` corresponding to the total number of default records

### Note the a "0" in the 'default payments next month' column means "non-default" and a 1 means "default"

### YOUR ANSWERS BELOW

ans0 = 23364
ans1 = 6636
```

Question 2:

```
### GRADED

### This proportion of default and non-default records means we are dealing with

### 'a') Balanced classes
### 'b') Unbalanced classes
### Assign the character associated with your choice as a string to ans1

### YOUR ANSWER BELOW

ans1 = 'b'
```

Investigating the Features:

```
df['EDUCATION'].value_counts()
```

Note: The code-book only describes the education variable as having four values (1-4), yet, here, there are seven values (0-6).
In some cases this might be grounds to throw out these unknow n values (0,5,6). For now, w e w ill leave them in, assuming that they have some (unknow n to us) meaning.

```
df['MARRIAGE'].value_counts()
```

Note: Again, the code book only describes three values for marriage (1-3), yet here, "0" also appears. Given w hat w e saw above, w e might assume the "0" in these categorical variables is functionally used for the "null" value

```
df['SEX'].value_counts()
```

Note: A slight imbalance exists representation of men and w omen, w ith w omen making up a little over 60% of our observations. Thankfully this column contains no "0"s.

A Closer look at the "PAY" variables

```
df['PAY_1'].value_counts()
```

Question 3:

```
### GRADED

### True or False:
### The code-book (description of variables that came with the data set; copied above and in link)
### described what a "0" in the 'Pay_#' columns means?

### Assign boolean answer to ans1
### YOUR ANSWER BELOW

ans1 = False
```

Investigate relationship between "Pay", "Bill_amt" and "Pay_amt" variables:

```
for i in [-2,-1,0,1,2,8]:
    print(df[df['PAY_1']==i][['PAY_1','BILL_AMT1','PAY_AMT1']].head(8), "\n")
```

Unclear exactly how these "PAY" variables w ork. Possibly they should be treated as categorical data instead of discrete and interval data, but, for this modeling task, keep them as interval data.

```
### Define function for creating histograms
def pay_hist(df, cols, ymax):
    plt.figure(figsize= (10,7)) # define fig size

    for index, col in enumerate(cols): # For each column passed to function
        plt.subplot(2,3, index +1) # plot on new subplot
        plt.ylim(ymax = ymax) # standardize ymax
        plt.hist(df[col]) # create hist
        plt.title(col) # title with column names
    plt.tight_layout(); # make sure titles don't overlap
```

```
pay_cols = ["PAY_"+str(n) for n in range(1,7)]
pay_amt_cols = ['PAY_AMT' + str(n) for n in range(1,7)]
bill_amt_cols = ['BILL_AMT' + str(n) for n in range(1,7)]
```

```
pay_hist(df, pay_cols, 20000)
```

Note: Clearly the "0" is the majority class for all of the "PAY" variables. But, unclear w hat a "0" means as it is not in the code book.

```
pay_hist(df, pay_amt_cols, 20000)
```

```
df[pay_amt_cols].boxplot();
```

```
df_no_0_pay_amt_1 = df[df["PAY_AMT1"]!=0]
df_no_0_pay_amt_1["PAY_AMT1"].hist()
```

Even taking out all the PAY_AMT of 0, most payments stay close to 0 w ith a long tail.

```
log_pay_amt1 = np.log10(df_no_0_pay_amt_1["PAY_AMT1"])
plt.hist(log_pay_amt1)
plt.title("Log10-Transformed values for 'PAY_AMT1' (Excluding 0s)");
```

Log Transformation (used above) or $\sqrt[4]{x}$ transformations (w hich automatically deals w ith 0s) can be a good w ay of looking more closely at skew ed data. Above, w e see that most of the repayment amounts are in the 1000's of dollars.

```
pay_hist(df, bill_amt_cols, 23000)
```

```
df[bill_amt_cols].boxplot();
```

Preprocessing

Currently "Sex" is coded as 2 for "female" and 1 for "male". As this is encoded as binary w e w ill change the column name to "FEMALE" and subtract 1 from each value - thus 1 w ill be "Female" and 0 "Male".

Both Education and Marriage are categorical w ith multiple options. `pd.get_dummies()` w ill allow us to create n-1 binary features to encode the n categories.

Note regarding `get_dummies()`

`pd.get_dummies()` is **NOT** appropriate in many ML applications.

For Example: Currently w e have the values 0-6 in the "EDUCATION" feature. Suppose that w hen w e split the training and test data, all 14 of the "0" values ended up in the test set. Running `pd.get_dummies()` on the test set w ould add 6 columns w here the training set w ould only have 5 columns added from that action. then, w hatever model had been fit w ould not know how to deal that new feature / category.

The function is used here for simplicities' sake

```
df['SEX'] = df['SEX']-1 # change vals of 'sex' to 0,1

df.rename(columns = {'SEX':'FEMALE', "default payment next month":"default"}, inplace = True) # rename col names

for col, pre in zip(["EDUCATION", "MARRIAGE"],["EDU","MAR"]): # get dummies and rename cols for ed and marraige
    df = pd.concat([
        df.drop(col, axis = "columns"), pd.get_dummies(df[col], prefix = pre, drop_first = True)],
    axis = 'columns')

df.head()
```

Question 4:

```
### GRADED
### Why is it inappropriate to use `pd.get_dummies()` when preprocessing data?

### 'a') it destroys data by removing categorical data
### 'b') it will skew results of models
### 'c') it cannot deal with "new" categories possibly found in a test set
### 'd') it is not a real method
### Assign character corresponding to your choice as string to ans1.

### YOUR ANSWER BELOW

ans1 = 'c'
```

Question 5

```
### GRADED
### Why are 'dummy' variables necessary?
### 'a') It helps to expand the number of features
### 'b') Models cannot deal multi-class categorical data
### 'c') It translates categorical data into quantitative data
### Assign character corresponding to your choice as string to ans1.

### YOUR ANSWER BELOW

ans1 = 'b'
```

Trees and Forests: Intro

Given a target (categorical or continuous), a decision tree iteratively splits data. It splits data at the value in w hichever feature that creates the greatest separation among the target variable.

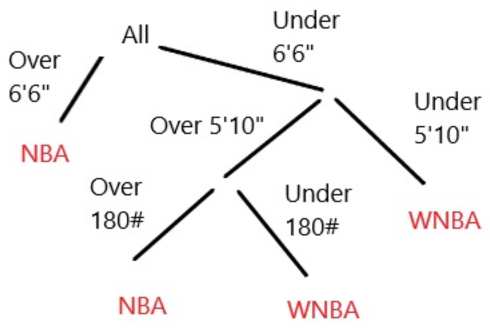
For example, say the target variable on a dataset of professional basketball players is league: NBA and WNBA. The available features are height and w eight.

Split 1: Although there are WNBA players over 6'8" tall, that is very rare. Thus, a decision tree might split the data at the height of 6'8", 6'7" or 6'6". At that split the tree w ould predict that any basketball player over 6'6" plays in the NBA.

Split 2: For players shorter than 6'6", the tree might decide to split that data again at 5'10" and predict that basketball players under 5'10" play in the WNBA.

Split 3: For those players betw een 5'10" and 6'6", maybe the tree might discriminate on w eight, predicting that all those players w ho w eigh more than 180 Lbs. play in the NBA.

The resulting tree could be visualized as below



Depth

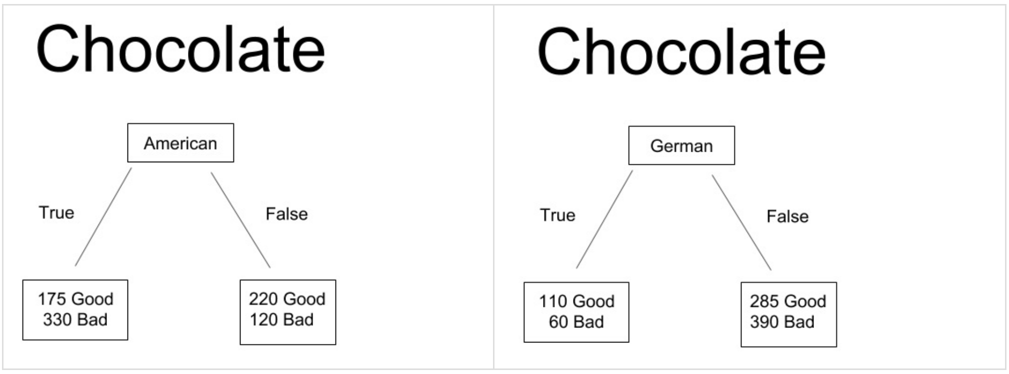
The above tree has a depth of 2.
The maximum depth to w hich a is allow ed to grow can be specified w ith `max_depth` in `sklearn`. By default, `max_depth` it is set to `None` w hich means the tree w ill grow until all leaves (terminal nodes) are pure, or until other user-specified criteria are met. Importantly, `max_depth` can impact the amount of time it takes to build a tree (this becomes especially important w hen starting to w ork w ith forests.)

Splitting

The splits in the above trees w ere determined intuitively. Thankfully decision Trees do not make their decisions using intuition. In the `sklearn` package tw o splitting criterion are available for classifiers; "gini" and "entropy". In general, "gini" splitting favors larger partitions, w here "entropy" favors splitting of smaller groups that are of a single class.

More on Gini/Entropy

Gini and Entropy Calculations



Above w e have a split in a data set regarding good and bad chocolate.

You w ill be asked to calculate the gini impurity in the next questions.

The Split on "American" show s 175 good chocolates and 330 bad chocolates w here `American == True`. There are 200 good and 120 bad chocolates w here `American == False`

The Split on "German" show s 110 good chocolates and 60 bad chocolates w here `German == True`. There are 285 good and 390 bad chocolates w here `German == False`

As a reminder; the gini-index for a node is: $$1- \sum_{j=1}^n p^2_j$$
Where there are n classes and $p_j$ is the frequency of class j in that node.
Finally, the indexes for each of these nodes is weighted by the proportion of data at each node, then summed.
Remember gini-indicies closer to 0 are more "pure"

Question 6:

```
### GRADED
### Calculate the gini impurity for the split on 'American'.

### Answer will be tested to 4 decimal places
### Assign float answer to ans1

### YOUR ANSWER BELOW
def calcGI(g1, b1, g2, b2):
    t = sum([g1,b1,g2,b2])

    def calcNode(g,b, t):
        tn = g+b
        prop = ((g+b)/t)
        node = (1- ((g/tn)**2)- ((b/tn)**2))
        # print(prop, node)
        return prop*node

    return calcNode(g1,b1, t) + calcNode(g2,b2,t)

ans1 = calcGI(175, 330, 220, 120) #--> ~ .4544
```

Question 7:

```
### GRADED
### Same instructions as above, but this time, calculate gini impurity for split on `German`
### Assign float answer to ans1

### YOUR ANSWER BELOW

def calcGI(g1, b1, g2, b2):
    t = sum([g1,b1,g2,b2])
    def calcNode(g,b, t):
        tn = g+b
        prop = ((g+b)/t)
        node = (1- ((g/tn)**2)- ((b/tn)**2))
        print(prop, node)
        return prop*node

    return calcNode(g1,b1, t) + calcNode(g2,b2,t)

ans1 = calcGI(110, 60, 285, 390) #--> 0.4816336
```

Question 8:

```
### GRADED
### According to the gini values calcualted above, which split is better?

### 'a') American
### 'b') German
### Assign character associated with your choice as string to ans1.

### YOUR ANSWER BELOW

ans1 = 'a'
```

Forests

"Forests" are collections of decision trees designed to protect against over-fitting.
A single decision tree (particularly one that is allowed to grow to any depth), may be prone to overfitting. Algorithmically, the tree is designed to continue to make splits until it has completely classified all of the available data, and/or exhausted every possible split. Thus, no complexity and no particular is too fine for the tree to split upon. A tree might be "pruned" (by setting max_depth) to protect against over fitting, but, a "forest" of trees might also be used instead.

Building Tree / Forest Models

Question 9:

Baseline Accuracy

```
### GRADED
### Assume we are predicting default/not-default.
### What percent (int between 0 and 100) of observations would be correctly predicted if the majority class
### were predicted every time.
### E.g., What is the baseline accuracy?
### Assign int to ans1

### YOUR ANSWER BELOW

ans1=max(df['default'].value_counts()/df.shape[0])*100 # --> ~78
```

Import models and metrics; create train-test split

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, BaggingClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

# Create tts
X_train, X_test, y_train, y_test = train_test_split(
    df.drop("default", axis = 'columns'), df['default'],
    test_size = .3, random_state = 1738)

# Instantiate tree and forest models
dt = DecisionTreeClassifier()
bag = BaggingClassifier()
rf = RandomForestClassifier()
et = ExtraTreesClassifier()
```

Look at performance of classifiers using default parameters

```
dt.fit(X_train, y_train)
print("Decision Tree: \n", classification_report(y_test, dt.predict(X_test)), "\n")


print("-----------")

bag.fit(X_train, y_train)
print("Bagging: \n", classification_report(y_test, bag.predict(X_test)), "\n")



print("-----------")

rf.fit(X_train, y_train)
print("Random Forest: \n", classification_report(y_test, rf.predict(X_test)), "\n")

print("------------")

et.fit(X_train, y_train)
print("Extra Trees: \n", classification_report(y_test, et.predict(X_test)), "\n")
```

Question 10

Values of different predictions

```
### GRADED
### Are all predictions in this data set equal in business value?
### e.g. is correctly predicting a "default" as valuable as predicting a "non-default"
### 'a') Equal values
### 'b') Unequal values
### Assign string associated with your choice to ans1

### YOUR ANSWER BELOW

ans1 = 'b'
```

Part 3: Hyper Parameter Tuning

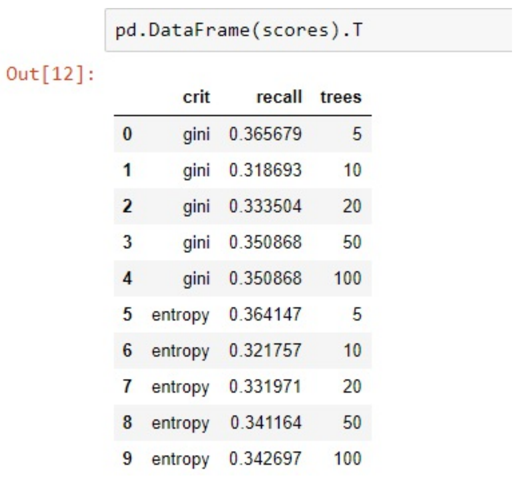
It appears both the Random Forest and the Extra Trees perform in a similarly effective manner. For this section, we will try to increase the performance of the Random Forest by tuning a couple hyper parameters, namely `criterion` and `estimators`

Because we are more interested in forecasting defaults than non-defaults, we will optimize on the recall of defaults -- recall is the proportion of defaults predicted over total defaults.

The below will not run on Vocareum due to processing constraints, thus the output is copied below:

```
# %%time
# from sklearn.metrics import recall_score
# criterion = ['gini', 'entropy']
# n_estimators = [5,10,20, 50, 100]
# scores = dict()
# i = 0
# for c in criterion:
#     for e in n_estimators:
#         rf = RandomForestClassifier(n_estimators = e, criterion = c, random_state = 1738)
#         rf.fit(X_train, y_train)
#         scores[i] = {'recall':recall_score(y_test, rf.predict(X_test)), 'trees' :e, "crit":c}
#         i+=1
```

```
# pd.DataFrame(scores).T
```



```
pd.DataFrame(scores).T
```

Out[12]:

	crit	recall	trees
0	gini	0.365679	5
1	gini	0.318693	10
2	gini	0.333504	20
3	gini	0.350868	50
4	gini	0.350868	100
5	entropy	0.364147	5
6	entropy	0.321757	10
7	entropy	0.331971	20
8	entropy	0.341164	50
9	entropy	0.342697	100

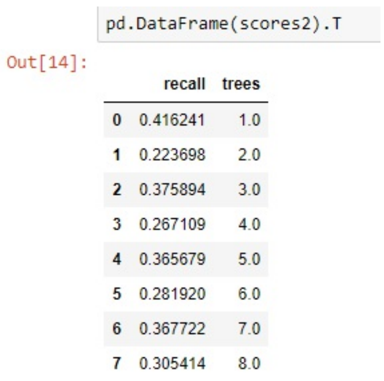
Question 11:

```
### GRADED
### According to the output shown in the picture above, the model with the best recall score featured:
### How many trees? Answer with int assigned to ans1.

### Which splitting method?
### 'a') entropy or 'b')gini.
### Put the character coressponding to your selection as a string in ans2

### YOUR ANSWER BELOW

ans1 = 5
ans2 = 'b'
```

With a general sense that maybe a `RandomForestClassifier` w ill perform best using `gini` splittling w ith somew here around 5 trees, further hyper-parameter tuning below.

```
# n_estimators = [1,2,3,4,5,6,7,8]
# scores2 = dict()
# i = 0
# for e in n_estimators:
#     rf = RandomForestClassifier(n_estimators = e, criterion = 'gini', random_state = 1738)
#     rf.fit(X_train, y_train)
#     scores2[i] = {'recall':recall_score(y_test, rf.predict(X_test)), 'trees' :e}
#     i+=1
```

```
# pd.DataFrame(scores2).T
```



```
pd.DataFrame(scores2).T
```

Out[14]:

	recall	trees
0	0.416241	1.0
1	0.223698	2.0
2	0.375894	3.0
3	0.267109	4.0
4	0.365679	5.0
5	0.281920	6.0
6	0.367722	7.0
7	0.305414	8.0

Question 12:

```
### GRADED
### On drilling down and focusing on 'gini' random forests with between 1 and 8 trees it becomes clear that
### hyper-parameter tuning here is creating:

### 'a') more consistency and better result
### 'b') more consistency and worse results
### 'c') no identifiable increase in consistency
### Assign the character corresponding to your choice as a string to ans1

### YOUR ANSWER BELOW

ans1 = 'c'
```

While it might be tempting to try to tune-and-tune-and-tune hyper-parameters to increase scores, in many cases (as in the example above) hyper-parameter tuning mostly results in ***not*** creating a better model, but a model that does a better job of predicting the test set.