Ensemble Techniques

To learn more about using Ensemble Methods, we are going to use the 'ML Marathon' dataset to test the results of XGBoost, Random Forest, and AdaBoost. All of which are different ensemble packages that will classify our data and return an accuracy score depending on the test data partition.

Before we start implementing those ensemble methods, we first have to do some exploratory data analysis and examine how can we clean our data to better fit the methods we are about to use. Here's a first look at our dataset.

In [2]:
```python
import pandas as pd
df = pd.read_csv('data.csv')
print(df.head())
df.info()
```

```
   age          job  marital  education default  balance housing loan  \
0   38   technician  married   tertiary      no      127     yes   no
1   41    housemaid  married    primary      no      365      no   no
2   39   management   single   tertiary      no     2454     yes   no
3   49  blue-collar  married    primary      no     6215     yes   no
4   37     services  married  secondary      no     1694     yes  yes

    contact  day month  duration  campaign  pdays  previous poutcome deposit
0  cellular   14   oct       113         1     50         2  success      no
1  cellular    8   aug       203         5     -1         0  unknown      no
2  cellular    4   may       716         3    263         2  failure     yes
3  cellular   11   may       549         1     -1         0  unknown      no
4  cellular   29   jan       404         2    251         6  failure      no
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8371 entries, 0 to 8370
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        8371 non-null   int64
 1   job        8371 non-null   object
 2   marital    8371 non-null   object
 3   education  8371 non-null   object
 4   default    8371 non-null   object
 5   balance    8371 non-null   int64
 6   housing    8371 non-null   object
 7   loan       8371 non-null   object
 8   contact    8371 non-null   object
 9   day        8371 non-null   int64
 10  month      8371 non-null   object
 11  duration   8371 non-null   int64
 12  campaign   8371 non-null   int64
 13  pdays      8371 non-null   int64
 14  previous   8371 non-null   int64
 15  poutcome   8371 non-null   object
 16  deposit    8371 non-null   object
dtypes: int64(7), object(10)
memory usage: 1.1+ MB
```

Exploratory Data Analysis

The first part of our EDA is to see if there are any missing values and drop columns if necessary to maintain the validity of our results. In this case, there are none so nothing needs to be done.

In [3]:
```python
na_values = df.isna().mean(axis=0)
print(na_values)
```

```
age          0.0
job          0.0
marital      0.0
education    0.0
default      0.0
balance      0.0
housing      0.0
loan         0.0
contact      0.0
day          0.0
month        0.0
duration     0.0
campaign     0.0
pdays        0.0
previous     0.0
poutcome     0.0
deposit      0.0
dtype: float64
```

Next, we designate a feature to be the target to which the classification will be executed on, store it, and drop it from the actual data.

In [4]:
```python
dropped = df.drop("deposit", axis=1)
target = df.deposit
```

For better utilization of ensemble algorithms, we have to scale all of our numerical data and factor categorical data. We will do this by creating a pipeline that will factor all the categorical data and another pipeline that will scale all the numerical data to have a mean of 0 and a variance of 1.

In [5]:
```python
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder

catpipe = Pipeline(
    steps=[
        ("impute", SimpleImputer(strategy="most_frequent")),
        ("oh-encode", OneHotEncoder(handle_unknown="ignore", sparse=False)),
    ]
)

from sklearn.preprocessing import StandardScaler

numpipe = Pipeline(
    steps=[("impute", SimpleImputer(strategy="mean")),
           ("scale", StandardScaler())]
)
```

Let's combine both pipelines and apply it to all of the data.

In [6]:
```python
categorical_data = dropped.select_dtypes(exclude="number").columns
numerical_data = dropped.select_dtypes(include="number").columns

from sklearn.compose import ColumnTransformer

full_processor = ColumnTransformer(
    transformers=[
        ("numeric", numpipe, numerical_data),
        ("categorical", catpipe, categorical_data),
    ]
)
```

XGBoost

Now that we have prepared our data we are going to import XGBoost, process the previously analyzed data and split it into train and test, then examine the XGBoost algorithm's accuracy using default hyper-parameters.

In [23]:
```python
import xgboost as xgb
xgboost_class = xgb.XGBClassifier()

dep_proc = full_processor.fit_transform(dropped)
indep_proc = SimpleImputer(strategy="most_frequent").fit_transform(
    target.values.reshape(-1, 1)
)

# split the data into train and test
from sklearn.model_selection import train_test_split
dep_train, dep_test, indep_train, indep_test = train_test_split(
    dep_proc, indep_proc, stratify=indep_proc, random_state=1121218
)

# make predictions based on model, find accuracy score and print it
# measure accuracy and time to make the predictions
from sklearn.metrics import accuracy_score
xgboost_class.fit(dep_train, indep_train)
import time
start_time = time.time()
preds = xgboost_class.predict(dep_test)
print("--- %.6s seconds ---" % (time.time() - start_time))
print('Accuracy: ', accuracy_score(indep_test, preds))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/_label.py:98: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/_label.py:133: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
--- 0.0076 seconds ---
Accuracy:  0.8413760152890588
```

Random Forest

The next ensemble algorithm we are going to use is Random Forest. Lucky for us, this algorithm is very easy to implement since it's found in the widely-used sklearn package. We've already done the hard work of cleaning our data and splitting it into train and test so all we have to do is use the sklearn package to make predictions and find the accuracy score for random forest.

In [24]:
```python
from sklearn.ensemble import RandomForestClassifier
rf_class = RandomForestClassifier(n_estimators = 100)

rf_class.fit(dep_train, indep_train)

# make predictions based on model, find accuracy score and print it
# measure accuracy and time to make the predictions
import time
start_time = time.time()
rf_pred = rf_class.predict(dep_test)
print("--- %.6s seconds ---" % (time.time() - start_time))
from sklearn.metrics import accuracy_score
print('Accuracy: ', accuracy_score(indep_test, rf_pred))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: DataConversionWarning: A column-vector y was passed w
hen a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  after removing the cwd from sys.path.
```

```
--- 0.0595 seconds ---
Accuracy:  0.8394648829431438
```

AdaBoost

The last algorithm we are going to use is AdaBoost. This algorithm is similar to Random Forest in a lot of ways except that it's decision trees only have a dpeth of 1. To analyze the accuracy of this algorithm we are going to do the same thing we did with Random Forest but using the AdaBoost libraries instead.

In [25]:
```python
from sklearn.ensemble import AdaBoostClassifier
adb_class = AdaBoostClassifier()
adb_class.fit(dep_train, indep_train)

# measure accuracy and time to make the predictions
import time
start_time = time.time()
adb_pred = adb_class.predict(dep_test)
print("--- %.6s seconds ---" % (time.time() - start_time))
from sklearn.metrics import accuracy_score
print('Accuracy: ', accuracy_score(indep_test, adb_pred))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
--- 0.0350 seconds ---
Accuracy:  0.831820353559484
```