Homework - Week 8

ECON 441B

Mauricio Vargas-Estrada
Master in Quantitative Economics
University of California - Los Angeles

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier

from sklearn.linear_model import LogisticRegression

from imblearn.over_sampling import KMeansSMOTE
```

0.)

Import and Clean data

In [ ]:
```
df = pd.read_csv('data/bank-additional-full (1).csv', sep = ';')
```

In [ ]:
```
df = df.drop(
    [
        "default",
        "pdays",
        "previous",
        "poutcome",
        "emp.var.rate",
        "cons.price.idx",
        "cons.conf.idx",
        "euribor3m",
        "nr.employed"
    ],
    axis = 1
)
```

In [ ]:
```
df = pd.get_dummies(
    df,
    columns = [
        "loan",
        "job",
        "marital",
        "housing",
        "contact",
        "day_of_week",
        "campaign",
        "month",
        "education"
    ],
    drop_first = True
)
```

In [ ]:
```
y = pd.get_dummies(df["y"], drop_first = True)
X = df.drop(["y"], axis = 1)
```

In [ ]:
```
def bar_plot(y):
    obs = len(y)
    plt.bar(
        ["No","Yes"],
        [len(y[y.yes==0])/obs, len(y[y.yes==1])/obs]
    )
    plt.ylabel("Percentage of Data")
    plt.show()
```

In [ ]:
```
bar_plot(y)
```

In [ ]:
```
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(
```

```
    X.astype(int), y.astype(int),
    test_size=0.3,
    random_state=42
)
```

1.)

Based on the visualization above, use your expert opinion to transform the data based on what we learned this quarter.

In [ ]:
```
smote = KMeansSMOTE(
    random_state=42
)
```

In [ ]:
```
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)
```

In [ ]:
```
bar_plot(y_train_smote)
```

`KmeansSMOTE` is a method of oversampling that uses the KMeans algorithm to create synthetic data. It is a combination of `KMeans` and `SMOTE` algorithms. It was applied to the training data to balance the classes.

2.)

Build and visualize a decision tree of Max Depth 3. Show the confusion matrix.

In [ ]:
```
dtree = DecisionTreeClassifier(max_depth = 3)
dtree.fit(X_train_smote, y_train_smote)
```

In [ ]:
```
fig, axes = plt.subplots(
    nrows=1,
    ncols=1,
    figsize=(4,4),
    dpi=300
)

plot_tree(
    dtree,
    filled=True,
    feature_names=X_train_smote.columns,
    class_names=["No","Yes"]
)
```

1b.)

Confusion matrix on out of sample data. Visualize and store as variable

In [ ]:
```
y_pred = dtree.predict(X_test)
y_true = y_test
cm_raw = confusion_matrix(y_true, y_pred)
```

In [ ]:
```
class_labels = ['Negative', 'Positive']

# Plot the confusion matrix as a heatmap
sns.heatmap(
    cm_raw,
    annot=True,
    fmt='d',
    cmap='Blues',
    xticklabels=class_labels,
    yticklabels=class_labels
)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

3.)

Use bagging on your decision tree

In [ ]:
```
bag = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(max_depth=3),
    bootstrap_features=True,
    n_estimators=100,
    n_jobs=-1,
    random_state=42
)
bag.fit(X_train_smote, y_train_smote)
```

In [ ]:
```
confusion_matrix(y_test, bag.predict(X_test))
```

4.)

Boost your tree.

In [ ]:
```
boost = AdaBoostClassifier(
    base_estimator=DecisionTreeClassifier(max_depth=3),
    n_estimators=100,
    random_state=42
)
boost.fit(X_train_smote, y_train_smote)
```

In [ ]:
```
ConfusionMatrixDisplay(
    confusion_matrix(y_test, boost.predict(X_test))
).plot()
```

5.)

Create a superlearner with at least 4 base learner models. Use a logistic reg for your metalearner. Interpret your coefficients and save your CM.

In [ ]:
```
super = LogisticRegression()
```

In [ ]:
```
def X_calc_super(X):
    return np.array(
        [
            bag.predict_proba(X)[:,1],
            boost.predict_proba(X)[:,1],
            dtree.predict_proba(X)[:,1]
        ]
    ).T
```

In [ ]:
```
X_super = X_calc_super(X_train_smote)
```

In [ ]:
```
super.fit(X_super, y_train_smote)
```

In [ ]:
```
ConfusionMatrixDisplay(
    confusion_matrix(y_test, super.predict(X_calc_super(X_test)))
).plot()
```

In [ ]:
```
temp = f"""
Coefficients:
- Bagging : {np.round(super.coef_[0][0], 2)}
- Boosting : {np.round(super.coef_[0][1], 2)}
- D. Tree: {np.round(super.coef_[0][2], 2)}
"""

print(temp)
```

In this simple exercise the best performance was achieved with the `AdaBoost` model. The `LogisticRegression` model was used as a meta learner to combine the predictions of the `bagging`, `boosting`, and `dtree` models. The coefficients of the `DecisionTree` model is negative, which means that it is the least important in the ensemble. The `Boosting` model has the highest coefficient, which means it is the most important in the ensemble.