

## Supplement 6: Decision Trees and Random Forest

```
%matplotlib inline
import numpy as np
import pandas as pd

from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from scipy.stats import mode
```

### 6.3 Programming Task: Song popularity prediction using Random Forest

The goal of this task is to train a random forest model that predicts the song popularity using the datasets already provided in task 4.3

```
# Read data

#TODO

train_data = pd.read_csv('train-songs.csv')
test_data = pd.read_csv('test-songs.csv')

train_X = train_data.drop('popular', axis=1)
train_y = train_data['popular']
test_X = test_data.drop('popular', axis=1)
test_y = test_data['popular']

rd_model = RandomForestClassifier(n_estimators=100, random_state=42)
rd_model.fit(train_X, train_y)

predictions = rf_model.predict(test_X)
accuracy= accuracy_score(test_y, predictions)

print(f"Random Forest Model Accuracy: {accuracy:.2f}")
```

i. Implement a function that draws a bootstrap sample of size N from the train dataset, where N can be specified by the user.

```
def generate_bootstrap(train_X,train_y,N):
    indices = np.random.choice(len(train_X), size=N, replace = True)

    bootstrap_X = train_X.iloc[indices]
    bootstrap_y = train_y.iloc[indices]

    return bootstrap_X, bootstrap_y
```

ii. Complete the implementation of the random forest algorithm. For this task you may use the DecisionTreeClassifier from the scikit-learn library. The other parts of the random forest algorithm must be implemented using only Scipy/Numpy.

```
class RandomForest:
    def __init__(self,n_trees,max_features,max_samples,min_node_size, max_depth):

        self.n_trees = n_trees
        self.max_features = max_features
        self.max_samples = max_samples
        self.min_node_size = min_node_size
        self.max_depth = max_depth

        self.trees = [
            tree.DecisionTreeClassifier(
                max_features=self.max_features,
                max_depth=self.max_depth,
                min_samples_leaf=self.min_node_size
            ) for _ in range(self.n_trees)
        ]

    def train(self,train_X,train_y):
```

```

        for i in range(self.n_trees):

            boot_X, boot_y = generate_bootstrap(train_X, train_y, self.max_samples)

            self.trees[i].fit(boot_X, boot_y)

def predict(self, test_X):
    tree_preds = np.array([t.predict(test_X) for t in self.trees])
    y_predictions, count = mode(tree_preds, axis=0)
    return y_predictions.flatten()

```

iii. Train the model for the dataset from train-songs.csv using the parameters given below.

Parameter	Value
Number of trees	100
Maximum features per tree	2
Bootstrap sample size	20000
Minimum node size	1
Maximum tree depth	10

Note: The bootstrap sample size is the same as train dataset size in this task.

```

# Note: Run this cell without any changes. The model will train if the implementation of subtask (ii) is correct.

random_forest_model = RandomForest(n_trees=100, max_samples=20000, max_depth=10, min_node_size=1, max_features=2 )

random_forest_model.train(train_X, train_y)

```

iv. Calculate the accuracy of the model using the test dataset and compare your results with the RandomForestClassifier from the scikit-learn library using the following parameters.

```

# TODO Run predict for test data and calculate accuracy

y_pred = random_forest_model.predict(test_X)

accuracy = np.mean(y_pred == test_y)
print(f"Custom Random Forest Model Accuracy: {accuracy:.2f}")

```

```

# TODO: Train and predict using scikit-learn library
from sklearn.metrics import accuracy_score

sklearn_rf = RandomForestClassifier(
    n_estimators=100,
    max_features=2,
    max_depth=10,
    min_samples_leaf=1,
    random_state=42
)

sklearn_rf.fit(train_X, train_y)

sklearn_pred = sklearn_rf.predict(test_X)

sklearn_accuracy = accuracy_score(test_y, sklearn_pred)
print(f"Scikit-learn RandomForestClassifier Accuracy: {sklearn_accuracy * 100:.2f}%")

```

