Random Forest - Lab

September 18, 2021

Modify the Bagging scratch code in our lecture such that: - Calculate for oob evaluation for each bootstrapped dataset, and also the average score - Change the code to "without replacement" - Put everything into a class Bagging. It should have at least two methods, fit(X_train, y_train), and predict(X_test) - Modify the code from above to randomize features. Set the number of features to be used in each tree to be sqrt(n), and then select a subset of features for each tree. This can be easily done by setting our DecisionTreeClassifier max_features to 'sqrt'

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
                test_size=0.3, shuffle=True, random_state=42)
```

[2]:
```
import random, math
from sklearn.tree import DecisionTreeClassifier
from scipy import stats
from sklearn.metrics import classification_report, accuracy_score
import numpy as np

class RandomForest:

    def __init__(self, n_estimators, bootstrap_ratio, with_no_replacement=True):
        self.n_estimators = n_estimators
        self.bootstrap_ratio = bootstrap_ratio
        self.with_no_replacement = with_no_replacement
        self.tree_params = {'max_depth': 2, 'max_features': 'sqrt'}
        self.models = [DecisionTreeClassifier(**self.tree_params) for _ in␣
↪range(n_estimators)]
```

```
    def fit(self, X, y):
        m, n = X.shape
        sample_size = int(self.bootstrap_ratio * len(X))

        X_samples = np.zeros((self.n_estimators, sample_size, n))
        y_samples = np.zeros((self.n_estimators, sample_size))

        X_samples_oob = []
        y_samples_oob = []

        for i in range(self.n_estimators):
            oob_index = []
            indices = []
            for j in range(sample_size):
                index = random.randrange(m)
                if self.with_no_replacement:
                    while index in indices:
                        index = random.randrange(m)
                indices.append(index)
                oob_index.append(index)

                X_samples[i,j,:] = X[index]
                y_samples[i,j] = y[index]

            mask = np.zeros((m), dtype=bool)
            mask[oob_index] = True
            X_samples_oob.append(X[~mask])
            y_samples_oob.append(y[~mask])

        oob_score = 0
        print("Out of bag score for each tree")
        for i, model in enumerate(self.models):
            X_train = X_samples[i]
            y_train = y_samples[i]
            model.fit(X_train, y_train)

            X_test = np.array(X_samples_oob[i])
            y_test = np.array(y_samples_oob[i])

            y_hat = model.predict(X_test)

            oob_score += accuracy_score(y_test, y_hat)
            print(f"Tree {i}", accuracy_score(y_test, y_hat))
        self.avg_oob_score = oob_score / len(self.models)
        print(f'Average out of bag score: {self.avg_oob_score}')
    def predict(self, X): #<---X_test
        #make prediction and return the probabilities
```

```
        predictions = np.zeros((self.n_estimators, X.shape[0]))
        for i, model in enumerate(self.models):
            yhat = model.predict(X)
            predictions[i, :] = yhat
        return stats.mode(predictions)[0][0]
```

[3]:
```
model = RandomForest(n_estimators=5, bootstrap_ratio=0.8)
model.fit(X_train, y_train)
y_hat = model.predict(X_test)
print(classification_report(y_test, y_hat))
```

```
Out of bag score for each tree
Tree 0 0.8095238095238095
Tree 1 0.9047619047619048
Tree 2 0.9047619047619048
Tree 3 0.6190476190476191
Tree 4 1.0
Average out of bag score: 0.8476190476190476
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45
```

[ ]:
```

```