Random Forest - Lab

September 16, 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
```

[2]:
```
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)
```

[3]:
```
from sklearn.tree import DecisionTreeClassifier
import random
from scipy import stats
from sklearn.metrics import classification_report, accuracy_score
```

[4]:
```
class Bagging:
    def __init__(self, B, boostrap_ratio, without_replacement=True):
        self.B = B
        self.boostrap_ratio = boostrap_ratio
        self.without_replacement = without_replacement
        self.tree_params = {'max_depth': 2, 'criterion':'gini',␣
↪'min_samples_split': 5, 'max_features': 'sqrt'}
        self.models = [DecisionTreeClassifier(**self.tree_params) for _ in␣
↪range(B)]

    def fit(self, X, y):
        #sample size for each tree
```

```
        B = self.B
        m, n = X.shape
        sample_size = int(self.boostrap_ratio * len(X))

        xsamples = np.zeros((B, sample_size, n))
        ysamples = np.zeros((B, sample_size))

        xsamples_oob = []
        ysamples_oob = []

        #subsamples for each model
        for i in range(B):
            oob_idx = []
            idxes = []
            ##sampling with replacement; i.e., sample can occur more than once
            #for the same predictor
            for j in range(sample_size):
                idx = random.randrange(m)   #<----with replacement #change so␣
↪no repetition
                if (self.without_replacement):
                    while idx in idxes:
                        idx = random.randrange(m)
                oob_idx.append(idx)
                idxes.append(idx)
                xsamples[i, j, :] = X_train[idx]
                ysamples[i, j] = y_train[idx]
                #keep track of idx that i did not use for ith tree
            mask = np.zeros((m), dtype=bool)
            mask[oob_idx] = True
            xsamples_oob.append(X[~mask])
            ysamples_oob.append(y[~mask])

        #fitting each estimator
        oob_scores = []
        for i, model in enumerate(self.models):
            _X = xsamples[i, :]
            _y = ysamples[i, :]
            model.fit(_X, _y)

            _X_test = np.asarray(xsamples_oob[i])
            _y_test = np.asarray(ysamples_oob[i])
            yhat = model.predict(_X_test)
            oob_score = accuracy_score(_y_test, yhat)
            oob_scores.append(oob_score)
        self.oob_scores = np.array(oob_scores)
        self.avg_oob_score = self.oob_scores.sum() / len(self.models)
```

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

        yhat = stats.mode(predictions)[0][0]
        return yhat
```

[5]:
```
model = Bagging(B=5, boostrap_ratio=0.8)
model.fit(X_train, y_train)
```

[6]:
```
yhat = model.predict(X_test)
print(classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

[7]:
```
print("Out of Bag scores:")
for i, score in enumerate(model.oob_scores): print(f"Model {i+1}: {score}")
print("Average Out of bag score:", model.avg_oob_score)
```

```
Out of Bag scores:
Model 1: 0.9047619047619048
Model 2: 0.9047619047619048
Model 3: 0.9047619047619048
Model 4: 0.9523809523809523
Model 5: 0.9047619047619048
Average Out of bag score: 0.9142857142857144
```

[8]:
```
#this is the same as RandomForest
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {"n_estimators": [10, 50, 100],
              "criterion": ["gini", "entropy"],
              "max_depth": np.arange(1, 10)}
model = RandomForestClassifier()
```

```
grid = GridSearchCV(model, param_grid)
grid.fit(X, y)

print(grid.best_params_)

model = grid.best_estimator_
model.fit(X_train, y_train)

yhat = model.predict(X_test)

print(classification_report(y_test, yhat))
```

```
{'criterion': 'gini', 'max_depth': 2, 'n_estimators': 10}
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

[ ]: