index

March 22, 2022

1 Gradient Boosting - Lab

1.1 Introduction

In this lab, we'll learn how to use both Adaboost and Gradient Boosting classifiers from scikit-learn!

1.2 Objectives

You will be able to:

- Use AdaBoost to make predictions on a dataset
- Use Gradient Boosting to make predictions on a dataset

1.3 Getting Started

In this lab, we'll learn how to use boosting algorithms to make classifications on the Pima Indians Dataset. You will find the data stored in the file `'pima-indians-diabetes.csv'`. Our goal is to use boosting algorithms to determine whether a person has diabetes. Let's get started!

We'll begin by importing everything we need for this lab. Run cell below:

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix,␣
 ↪classification_report
```

Now, use Pandas to import the data stored in `'pima-indians-diabetes.csv'` and store it in a DataFrame. Print the first five rows to inspect the data we've imported and ensure everything loaded correctly.

[3]:
```
# Import the data
df = pd.read_csv('pima-indians-diabetes.csv')

# Print the first five rows
df.head()
```

[3]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

1.4 Cleaning, exploration, and preprocessing

The target we're trying to predict is the `'Outcome'` column. A `1` denotes a patient with diabetes.

By now, you're quite familiar with exploring and preprocessing a dataset.

In the following cells:

- Check for missing values and deal with them as you see fit (if any exist)
- Count the number of patients with and without diabetes in this dataset
- Store the target column in a separate variable and remove it from the dataset
- Split the dataset into training and test sets, with a `test_size` of 0.25 and a `random_state` of 42

[4]:
```python
# Check for missing values
df.isna().sum()
```

[4]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[5]:
```python
# Number of patients with and without diabetes
dict(df["Outcome"].value_counts())
```

[5]:
```
{0: 500, 1: 268}
```

[6]:
```python
target = df["Outcome"]
df = df.drop("Outcome", axis = 1)
```

[7]:
```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df, target,
                                                    test_size = 0.25,
                                                   random_state = 42)
```

1.5 Train the models

Now that we've explored the dataset, we're ready to fit some models!

In the cell below:

- Instantiate an `AdaBoostClassifier` (set the `random_state` for 42)
- Instantiate a `GradientBoostingClassifer` (set the `random_state` for 42)

[9]:
```
# Instantiate an AdaBoostClassifier
adaboost_clf = AdaBoostClassifier(random_state = 42)

# Instantiate an GradientBoostingClassifier
gbt_clf = GradientBoostingClassifier(random_state = 42)
```

Now, fit the training data to both the classifiers:

[10]:
```
# Fit AdaBoostClassifier
adaboost_clf.fit(X_train, y_train)
```

[10]:
```
AdaBoostClassifier(random_state=42)
```

[11]:
```
# Fit GradientBoostingClassifier
gbt_clf.fit(X_train, y_train)
```

[11]:
```
GradientBoostingClassifier(random_state=42)
```

Now, let's use these models to predict labels on both the training and test sets:

[12]:
```
# AdaBoost model predictions
adaboost_train_preds = adaboost_clf.predict(X_train)
adaboost_test_preds  = adaboost_clf.predict(X_test)

# GradientBoosting model predictions
gbt_clf_train_preds = gbt_clf.predict(X_train)
gbt_clf_test_preds  = gbt_clf.predict(X_test)
```

Now, complete the following function and use it to calculate the accuracy and f1-score for each model:

[14]:
```
def display_acc_and_f1_score(true, preds, model_name):


    acc = accuracy_score(true, preds)
```

```
    f1  = f1_score(true, preds)
    print("Model: {}".format(model_name))
    print("Accuracy: {}".format(acc))
    print("F1-Score: {}".format(f1))

print("Training Metrics")
display_acc_and_f1_score(y_train, adaboost_train_preds, model_name='AdaBoost')
print("")
display_acc_and_f1_score(y_train, gbt_clf_train_preds, model_name='Gradient␣
 ↪Boosted Trees')
print("")
print("Testing Metrics")
display_acc_and_f1_score(y_test, adaboost_test_preds, model_name='AdaBoost')
print("")
display_acc_and_f1_score(y_test, gbt_clf_test_preds, model_name='Gradient␣
 ↪Boosted Trees')
```

```
Training Metrics
Model: AdaBoost
Accuracy: 0.8350694444444444
F1-Score: 0.7493403693931399

Model: Gradient Boosted Trees
Accuracy: 0.9409722222222222
F1-Score: 0.9105263157894736

Testing Metrics
Model: AdaBoost
Accuracy: 0.7239583333333334
F1-Score: 0.618705035971223

Model: Gradient Boosted Trees
Accuracy: 0.7447916666666666
F1-Score: 0.6620689655172414
```

Let's go one step further and create a confusion matrix and classification report for each. Do so in the cell below:

[15]:
```
adaboost_confusion_matrix = confusion_matrix(y_test, adaboost_test_preds)
adaboost_confusion_matrix
```

[15]:
```
array([[96, 27],
       [26, 43]])
```

[16]:
```
gbt_confusion_matrix = confusion_matrix(y_test, gbt_clf_test_preds)
gbt_confusion_matrix
```

[16]:
```
array([[95, 28],
       [21, 48]])
```

[17]:
```
adaboost_classification_report =␣
↪classification_report(y_test,adaboost_test_preds)
print(adaboost_classification_report)
```

```
              precision    recall  f1-score   support

           0       0.79      0.78      0.78       123
           1       0.61      0.62      0.62        69

    accuracy                           0.72       192
   macro avg       0.70      0.70      0.70       192
weighted avg       0.72      0.72      0.72       192
```

[18]:
```
gbt_classification_report = classification_report(y_test,gbt_clf_test_preds)
print(gbt_classification_report)
```

```
              precision    recall  f1-score   support

           0       0.82      0.77      0.79       123
           1       0.63      0.70      0.66        69

    accuracy                           0.74       192
   macro avg       0.73      0.73      0.73       192
weighted avg       0.75      0.74      0.75       192
```

Question: How did the models perform? Interpret the evaluation metrics above to answer this question.

Write your answer below this line: ______________________________

As a final performance check, let's calculate the 5-fold cross-validated score for each model!

Recall that to compute the cross-validation score, we need to pass in:

- A classifier
- All training data
- All labels
- The number of folds we want in our cross-validation score

Since we're computing cross-validation score, we'll want to pass in the entire dataset, as well as all of the labels.

In the cells below, compute the mean cross validation score for each model.

[19]:
```
print('Mean Adaboost Cross-Val Score (k=5):')
print(cross_val_score(adaboost_clf, df, target, cv=5).mean())
```

```
# Expected Output: 0.7631270690094218
```

```
Mean Adaboost Cross-Val Score (k=5):
0.7631270690094218
```

[20]:
```
print('Mean GBT Cross-Val Score (k=5):')
print(cross_val_score(gbt_clf, df, target, cv=5).mean())
# Expected Output: 0.7591715474068416
```

```
Mean GBT Cross-Val Score (k=5):
0.7604702487055428
```

These models didn't do poorly, but we could probably do a bit better by tuning some of the important parameters such as the ***Learning Rate***.

1.6 Summary

In this lab, we learned how to use scikit-learn's implementations of popular boosting algorithms such as AdaBoost and Gradient Boosted Trees to make classification predictions on a real-world dataset!