learn-co-curriculum / dsc-gradient-boosting-lab Public

View license

1 star 147 forks

Star Watch

Code Issues Pull requests Actions Projects Security Insights

solution

This branch is 7 commits ahead, 7 commits behind master.

sumedh10 fix grammar/typos/spelling ... on Jan 28, 2020 8

View code

README.md

Gradient Boosting - Lab

Introduction

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

Objectives

You will be able to:

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

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:

```
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, classificati
```

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.

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

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

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
0	6	148	72	35	0	33.6

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
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

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

```
# Check for missing values
df.isna().sum()
```

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

```
# Number of patients with and without diabetes
df['Outcome'].value_counts()
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

```
target = df['Outcome']
df = df.drop('Outcome', axis=1)
```

```
# 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, ranc
```

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)

```
# 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:

```
adaboost_clf.fit(X_train, y_train)
```

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                   n_estimators=50, random_state=42)
```

```
gbt_clf.fit(X_train, y_train)
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
```

```
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='auto',
                          random_state=42, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0,
                          warm_start=False)
```

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

```
# 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:

```
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
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 Tr
```

```
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: 0.75
F1-Score: 0.6666666666666666
```

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

```
adaboost_confusion_matrix = confusion_matrix(y_test, adaboost_test_preds)
adaboost_confusion_matrix
```

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

```
gbt_confusion_matrix = confusion_matrix(y_test, gbt_clf_test_preds)
gbt_confusion_matrix
```

```
array([[96, 27],
       [21, 48]])
```

```
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
```

```
gbt_classification_report = classification_report(y_test, gbt_clf_test_preds)
print(gbt_classification_report)
```

```
              precision    recall  f1-score   support

           0       0.82      0.78      0.80       123
           1       0.64      0.70      0.67        69

    accuracy                           0.75       192
   macro avg       0.73      0.74      0.73       192
weighted avg       0.76      0.75      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.

```
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
```

```
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):
0.7591715474068416
```

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***.

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!

Releases

No releases published

Packages

No packages published

Contributors 4

mike-kane Mike Kane

sumedh10 Sumedh Panchadhar

LoreDirick Lore Dirick

fpolchow Forest Polchow

Languages

- **Jupyter Notebook** 100.0%