XGBoost (Extreme Gradient Boosting)

XGBoost which is short for Extreme Gradient Boosting makes a couple of computational and algorithmic modifications to the stochastic gradient boosting algorithm. This enhanced algorithm is a favorite in machine learning practice due to its speed and has been the winning algorithm in many machine learning competitions. Let's go through some of the modifications made by the XGBoost algorithm.

1. Parallel training: XGBoost supports parallel training over multiple cores. This has made XGBoost extremely fast compared to other machine learning algorithms.
2. Out of core computation: XGBoost facilitates training from data not loaded into memory. This feature is a huge advantage when you're dealing with large datasets that may not necessarily fit into the RAM of the computer.
3. Sparse data optimization: XGBoost is optimized to handle and speed up computation with sparse matrices. Sparse matrices contain lots of zeros in its cells.

XGBoost with Scikit-learn

This section will implement XGBoost with Scikit-learn for both regression and classification use cases.

XGBoost for Classification

In this code example, we will build a XGBoost classification model to predict the species of flowers from the Iris dataset.

```
# import packages
from xgboost import XGBClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# load dataset
data = datasets.load_iris()
```

```
# separate features and target
X = data.data
y = data.target

# split in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True)

# create the model
xgboost_classifier = XGBClassifier()

# fit the model on the training set
xgboost_classifier.fit(X_train, y_train)

# make predictions on the test set
predictions = xgboost_classifier.predict(X_test)

# evaluate the model performance using accuracy metric
print("Accuracy: %.2f" % accuracy_score(y_test, predictions))

'Output":
Accuracy: 0.95
```

XGBoost for Regression

In this code example, we will build a XGBoost regression model to predict house prices from the Boston house-prices dataset.

```
# import packages
from xgboost import XGBRegressor
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt

# load dataset
data = datasets.load_boston()

# separate features and target
X = data.data
y = data.target
```

```
# split in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=True)

# create the model
xgboost_reg = XGBRegressor()

# fit the model on the training set
xgboost_reg.fit(X_train, y_train)

# make predictions on the test set
predictions = xgboost_reg.predict(X_test)

# evaluate the model performance using the root mean square error metric
print("Root mean squared error: %.2f" % sqrt(mean_squared_error(y_test,
predictions)))

'Output':
Root mean squared error: 3.69
```

In this chapter, we surveyed and implemented ensemble machine learning algorithms that combine weak decision tree learners to create a strong classifier for learning regression and classification problems. In the next chapter, we will discuss more techniques for implementing supervised machine learning models with Scikit-learn.

CHAPTER 24

More Supervised Machine Learning Techniques with Scikit-learn

This chapter will cover using Scikit-learn to implement machine learning models using techniques such as

- Feature engineering
- Resampling methods
- Model evaluation methods
- Pipelines for streamlining machine learning workflows
- Techniques for model tuning

Feature Engineering

Feature engineering is the process of systematically choosing the set of features in the dataset that are useful and relevant to the learning problem. It is often the case that irrelevant features negatively affect the performance of the model. This section will review some techniques implemented in Scikit-learn for selecting relevant features from a dataset. The techniques surveyed include

- Statistical tests to select the best *k* features using the **SelectKBest** module
- Recursive feature elimination (RFE) to recursively remove irrelevant features from the dataset

© Ekaba Bisong 2019
E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform*,
https://doi.org/10.1007/978-1-4842-4470-8_24