New Package Introduction - Introduction to Bagging and Random Forest

Now, Let's go through some of the common functions used in this LVC implemented through the Scikit-learn library

Bagging

Bagging, often known as bootstrap aggregation, is an ensemble learning approach for reducing variance in a noisy dataset. Bagging selects a random sample of data from a training set with replacement.

Random Forest

Random forest is a type of Supervised Machine Learning Algorithm that is commonly used in classification and regression issues. It constructs decision trees from several samples and uses their majority vote for classification and average for regression.

Random Forest regressor

```
sklearn.ensemble.RandomForestRegressor(n_estimators=100, *, criterion='squared_error', max_depth=None, min_sampl
```

Example

```
from sklearn.ensemble import RandomForestRegressor
rf_regressor = RandomForestRegressor()
rf_regressor.fit(x_train, y_train)
```

You can refer to the random forest regressor sklearn documentation for a better understanding of the parameters and attributes here

Similarly, we can implement the random forest classifier through the sklearn library.

Random Forest classifier

```
sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split
```

Example

```
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier()
rf_classifier.fit(x_train, y_train)
```

You can refer to the random forest classifier sklearn documentation for a better understanding of the parameters and attributes here

Bagging Regressor

A Bagging regressor is an ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregates their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

```
sklearn.ensemble.BaggingRegressor(base_estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bo
```

Example

```
from sklearn.ensemble import BaggingRegressor
model = BaggingRegressor()
model.fit(x_train, y_train)
```

You can refer to the bagging regressor sklearn documentation for a better understanding of the parameters and attributes here

Boosting

Boosting is an ensemble learning strategy that transforms a group of weak learners into strong learners in order to reduce training errors. Boosting involves selecting a random sample of data, fitting it with a model, and then training it sequentially. That is, each model attempts to compensate for the shortcomings of its predecessor. Each cycle combines the weak rules of each classifier to generate one strict prediction rule.

Adaboost Regressor

```
sklearn.ensemble.AdaBoostRegressor(base_estimator=None, *, n_estimators=50, learning_rate=1.0, loss='linear', ra
```

Example

```
from sklearn.ensemble import AdaBoostRegressor
adaboost_model = AdaBoostRegressor()
adaboost_model.fit(x_train, y_train)
```

You can refer to the Adaboost regressor sklearn documentation for a better understanding of the parameters and attributes here

Gradient boosting regressor

```
sklearn.ensemble.GradientBoostingRegressor(*, loss='squared_error', learning_rate=0.1, n_estimators=100, subsamp
```

Example

```
from sklearn.ensemble import GradientBoostingRegressor
gradientboost_model = GradientBoostingRegressor()
gradientboost_model.fit(x_train, y_train)
```

You can refer to the Gradient boosting regressor sklearn documentation for a better understanding of the parameters and attributes here

Regression Evaluation metrics

It is necessary to obtain the accuracy on training data, But it is also important to get a genuine and approximate result on unseen data otherwise model is of no use. Below is an example of how we can check the regression model's efficiency through its respective available metrics

Example

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
#making predictions on the test data using the trained model
y_pred = model.predict(x_test)
```

```
print("mse = ", mean_sqaured_error(y_test, y_pred))
print("R^2 score = ", r2_score(y_test, y_pred))
print("mae = ", mean_absolute_error(y_test, y_pred))
```

You can refer to the sklearn regression metrics here for more details.

XGBOOST

XGBoost is an open-source software library that implements optimized distributed gradient boosting machine learning algorithms under the Gradient Boosting framework. It stands for Extreme Gradient Boosting, which is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library. It provides a parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems.

Library Installation

We can install the xgboost library in jupyter notebook using the pip command

```
!pip install xgboost
```

User Guide

This library helps us to implement both the Xgboost regressor and classifier. Now let's see how we can import the necessary methods

Xgboost Regression

```
from xgboost import XGBRegressor
xgb_regressor = XGBRegressor()
xgb_regressor.fit(x_train, y_train)
```

You can refer to the Xgboost regressor documentation for a better understanding of the parameters and attributes here

Xgboost Classification

```
from xgboost import XGBClassifier
xgb_classifier = XGBClassifier()
xgb_classifier.fit(x_train, y_train)
```

You can refer to the Xgboost classifier documentation for a better understanding of the parameters and attributes here

Happy Learning!

< Previous

Next >