Gradient boosting evaluates the difference of the residuals for each tree and then uses that information to determine how to split the feature space in the successive tree.

Gradient boosting employs a pseudo-gradient in computing the residuals. This gradient is the direction of quickest improvement to the loss function. The residual variance is minimized as the gradient moves in the direction of steepest descent. This movement is the same as the stochastic gradient descent algorithm discussed in Chapter 16.

Tree Depth/Number of Trees

Gradient boosting can be controlled by choosing the tree depth as a hyper-parameter to the model. In practice, a tree depth of 1 performs well, as each tree consists of just a single split. Also, the number of trees can affect the model accuracy, because gradient boosting can overfit if the number of successive trees is vast.

Shrinkage

The shrinkage hyper-parameter λ controls the learning rate of the gradient boosting model. An arbitrarily small value of λ may necessitate a larger number of trees to obtain a good model performance. However, with a small shrinkage size and tree depth $d = 1$, the residuals slowly improve by creating more varied trees to improve the worst performing areas of the model. Rule of thumb: shrinkage size is 0.01 or 0.001.

Stochastic Gradient Boosting with Scikit-learn

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

SGB for Classification

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

```
# import packages
from sklearn.ensemble import GradientBoostingClassifier
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
sgb_classifier = GradientBoostingClassifier()

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

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

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

'Output":
Accuracy: 0.92
```

SGB for Regression

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

```
# import packages
from sklearn.ensemble import GradientBoostingRegressor
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
sgb_reg = GradientBoostingRegressor ()

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

# make predictions on the test set
predictions = sgb_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: 2.86
```

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()
```