Tree Pruning

Tree pruning is a technique for dealing with model overfitting when growing trees. Fully grown trees have a high tendency to overfit with high variances when applied to unseen samples.

Pruning involves growing a large tree and then pruning or clipping it to create a sub-tree. By doing so, we can have a full picture of the tree performance and then select a sub-tree that results in a minimized error measure on the test dataset. The technique for selecting the best sub-tree is called the cost complexity pruning or the weakest link pruning.

Strengths and Weaknesses of CART

One of the significant advantages of CART models is that they perform well on linear and non-linear datasets. Moreover, CART models implicitly take care of feature selection and work well with high-dimensional datasets.

On the flip side, CART models can very easily overfit the dataset and fail to generalize to new examples. This downside is mitigated by aggregating a large number of decision trees in techniques like Random forests and boosting ensemble algorithms.

CART with Scikit-learn

In this section, we will implement a classification and regression decision tree classifier with Scikit-learn.

Classification Tree with Scikit-learn

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

```
# import packages
from sklearn.tree import DecisionTreeClassifier
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
tree_classifier = DecisionTreeClassifier()

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

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

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

'Output":
Accuracy: 0.97
```

Regression Tree with Scikit-learn

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

```
# import packages
from sklearn.tree import DecisionTreeRegressor
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
tree_reg = DecisionTreeRegressor()

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

# make predictions on the test set
predictions = tree_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: 4.93
```

Random Forests

Random forest is a robust machine learning algorithm and is often the algorithm of choice for many classification and regression problems. It is a popular algorithm in machine learning competitions.

Random forest builds an ensemble classifier from a combination of several decision tree classifiers. This does an excellent job of reducing the variance that may be found in a single decision tree classifier.