**MACHINE LEARNING WITH TREE BASED MODELS IN PYTHON**

**Chapter 1, 'Classification and Regression Trees'.**

**Classification-tree**

* Sequence of if-else questions about individual features.
* Objective: infer class labels.
* Able to capture non-linear relationships between features and labels.
* Don't require feature scaling (ex: Standardization, ..)

# **Train your first classification tree**

In this exercise you'll work with the [**Wisconsin Breast Cancer Dataset**](https://www.kaggle.com/uciml/breast-cancer-wisconsin-data) from the UCI machine learning repository. You'll predict whether a tumor is malignant or benign based on two features: the mean radius of the tumor (radius\_mean) and its mean number of concave points (concave points\_mean).

The dataset is already loaded in your workspace and is split into 80% train and 20% test. The feature matrices are assigned to X\_train and X\_test, while the arrays of labels are assigned to y\_train and y\_test where class 1 corresponds to a malignant tumor and class 0 corresponds to a benign tumor. To obtain reproducible results, we also defined a variable called SEED which is set to 1.

**# Import DecisionTreeClassifier from sklearn.tree**

**from sklearn.tree import DecisionTreeClassifier**

**# Instantiate a DecisionTreeClassifier 'dt' with a maximum depth of 6**

**dt = DecisionTreeClassifier(max\_depth = 6, random\_state=SEED)**

**# Fit dt to the training set**

**dt.fit(X\_train, y\_train)**

**# Predict test set labels and show the first five predictions**

**y\_pred = dt.predict(X\_test)**

**print(y\_pred[0:5])**

# **Evaluate the classification tree**

Now that you've fit your first classification tree, it's time to evaluate its performance on the test set. You'll do so using the accuracy metric which corresponds to the fraction of correct predictions made on the test set.

The trained model dt from the previous exercise is loaded in your workspace along with the test set features matrix X\_test and the array of labels y\_test.

**# Import accuracy\_score**

**from sklearn.metrics import accuracy\_score**

**# Predict test set labels**

**y\_pred = dt.predict(X\_test)**

**# Compute test set accuracy**

**acc = accuracy\_score(y\_pred,y\_test)**

**print("Test set accuracy: {:.2f}".format(acc))**

<script.py> output:

Test set accuracy: 0.89

# **Logistic regression vs classification tree**

A classification tree divides the feature space into **rectangular regions**. In contrast, a linear model such as logistic regression produces only a single linear decision boundary dividing the feature space into two decision regions.

We have written a custom function called plot\_labeled\_decision\_regions() that you can use to plot the decision regions of a list containing two trained classifiers. You can type help(plot\_labeled\_decision\_regions) in the IPython shell to learn more about this function.

X\_train, X\_test, y\_train, y\_test, the model dt that you've trained in an earlier [**exercise**](https://campus.datacamp.com/courses/machine-learning-with-tree-based-models-in-python/classification-and-regression-trees?ex=2) , as well as the function plot\_labeled\_decision\_regions() are available in your workspace.

**# Import LogisticRegression from sklearn.linear\_model**

**from sklearn.linear\_model import LogisticRegression**

**# Instatiate logreg**

**logreg = LogisticRegression(random\_state=1)**

**# Fit logreg to the training set**

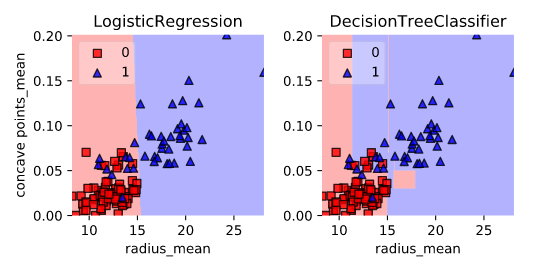
**logreg.fit(X\_train, y\_train)**

**# Define a list called clfs containing the two classifiers logreg and dt**

**clfs = [logreg, dt]**

**# Review the decision regions of the two classifiers**

**plot\_labeled\_decision\_regions(X\_test, y\_test, clfs)**



Notice how the decision boundary produced by logistic regression is linear while the boundaries produced by the classification tree divide the feature space into rectangular regions.

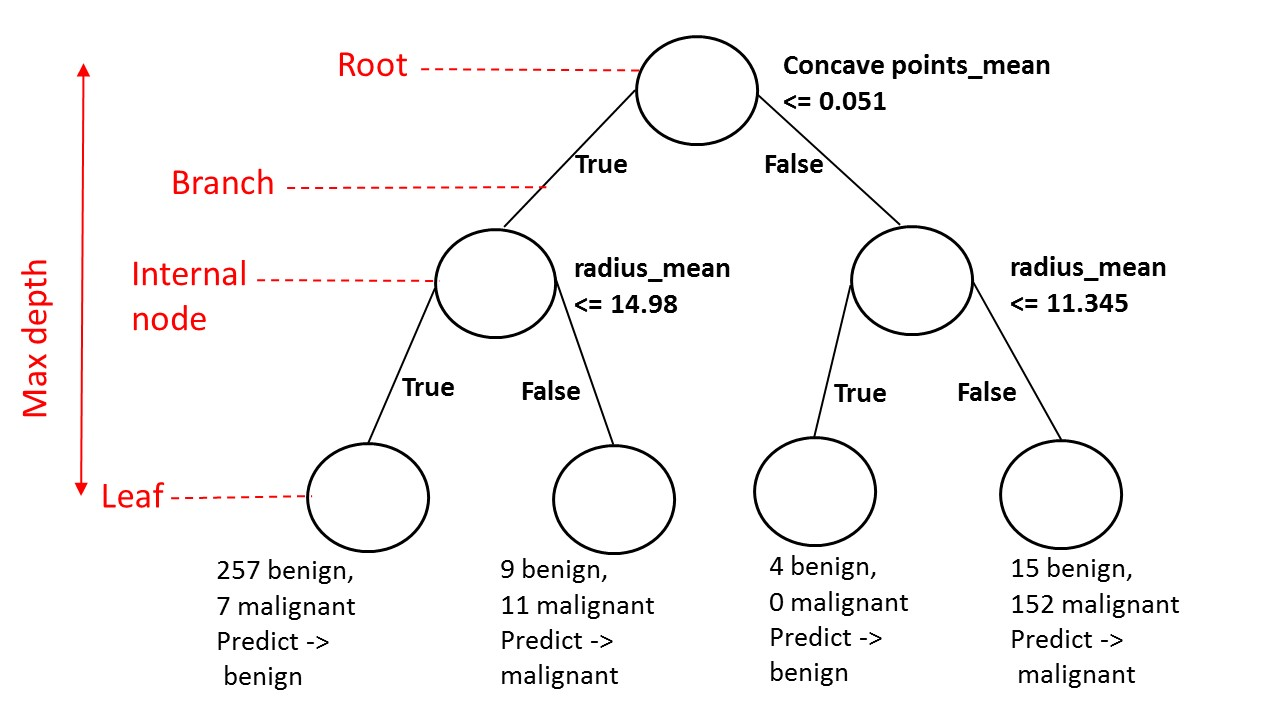
**Classification tree Learning**

**Building Blocks of a Decision-Tree**

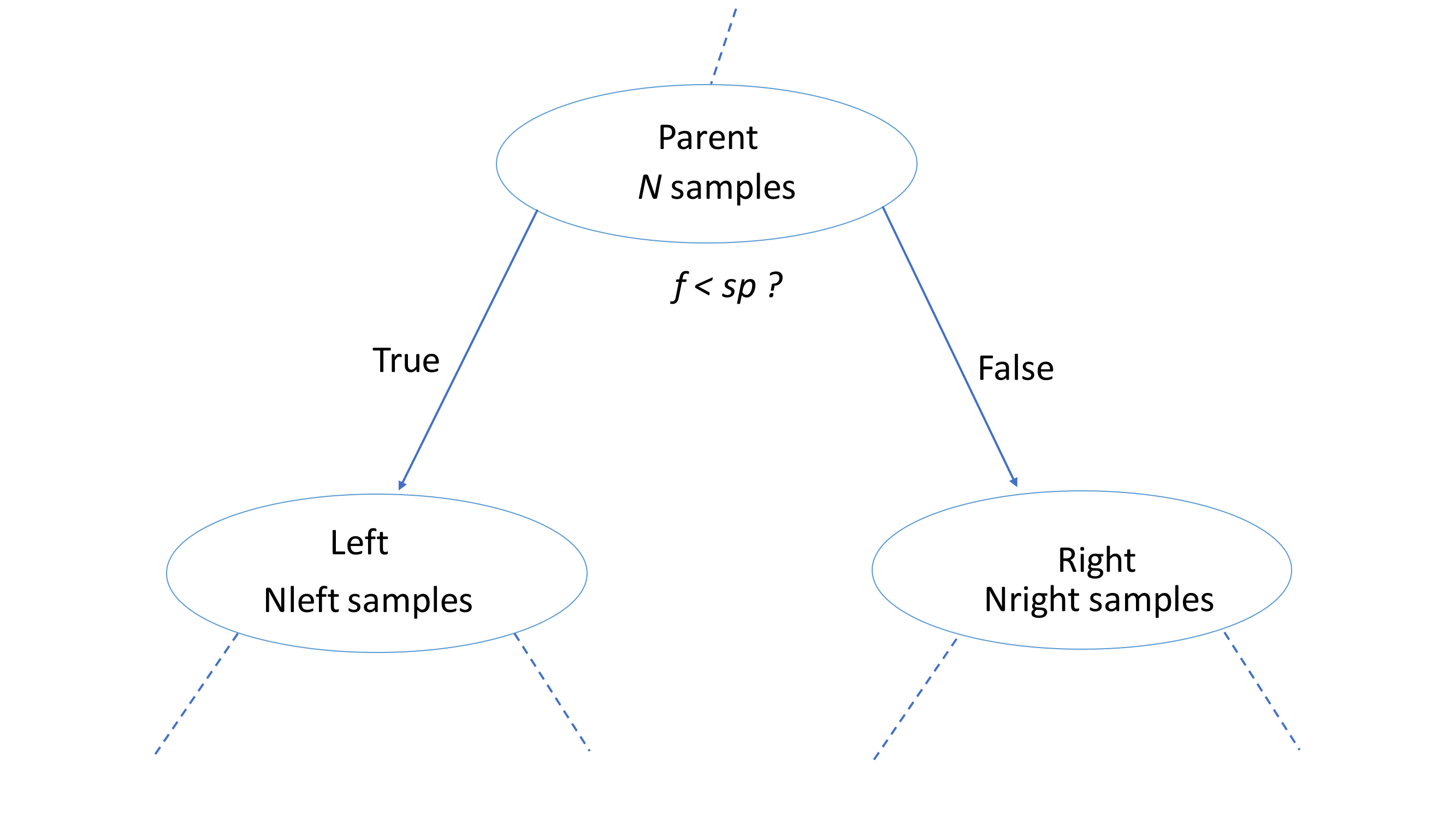
* Decision-Tree: data structure consisting of a hierarchy of nodes.
* Node: question or prediction.

Three kinds of nodes:

* Root: no parent node, question giving rise to two children nodes.
* Internal node: one parent node, question giving rise to two children nodes.
* Leaf: one parent node, no children nodes --> prediction.



**Information Gain (IG)**



****

**Criteria to measure the impurity of a node I(node) :**

* gini index,
* entropy

**Classification-Tree Learning**

* Nodes are grown recursively.

At each node, split the data based on:

* feature f and split-point sp to maximize IG(node) .
* If IG(node) = 0, declare the node a leaf.

**# Import DecisionTreeClassifier**

**from sklearn.tree import DecisionTreeClassifier**

**# Import train\_test\_split**

**from sklearn.model\_selection import train\_test\_split**

**# Import accuracy\_score**

**from sklearn.metrics import accuracy\_score**

**# Split dataset into 80% train, 20% test**

**X\_train, X\_test, y\_train, y\_test= train\_test\_split(X, y, test\_size=0.2, stratify=y,**

**random\_state=1)**

**# Instantiate dt, set 'criterion' to 'gini'**

**dt = DecisionTreeClassifier(criterion='gini', random\_state=1)**

**# Fit dt to the training set**

**dt.fit(X\_train,y\_train)**

**# Predict test-set labels**

**y\_pred= dt.predict(X\_test)**

**# Evaluate test-set accuracy**

**accuracy\_score(y\_test, y\_pred)**

**Using entropy as a criterion**

In this exercise, you'll train a classification tree on the Wisconsin Breast Cancer dataset using entropy as an information criterion. You'll do so using all the 30 features in the dataset, which is split into 80% train and 20% test.

X\_train as well as the array of labels y\_train are available in your workspace.

**# Import DecisionTreeClassifier from sklearn.tree**

**from sklearn.tree import DecisionTreeClassifier**

**# Instantiate dt\_entropy, set 'entropy' as the information criterion**

**dt\_entropy = DecisionTreeClassifier(max\_depth=8, criterion='entropy', random\_state=1)**

**# Fit dt\_entropy to the training set**

**dt\_entropy.fit(X\_train, y\_train)**

**Entropy vs Gini index**

In this exercise you'll compare the test set accuracy of dt\_entropy to the accuracy of another tree named dt\_gini. The tree dt\_gini was trained on the same dataset using the same parameters except for the information criterion which was set to the gini index using the keyword 'gini'.

X\_test, y\_test, dt\_entropy, as well as accuracy\_gini which corresponds to the test set accuracy achieved by dt\_gini are available in your workspace.

**# Import accuracy\_score from sklearn.metrics**

**from sklearn.metrics import accuracy\_score**

**# Use dt\_entropy to predict test set labels**

**y\_pred = dt\_entropy.predict(X\_test)**

**# Evaluate accuracy\_entropy**

**accuracy\_entropy = accuracy\_score(y\_test, y\_pred)**

**# Print accuracy\_entropy**

**print('Accuracy achieved by using entropy: ', accuracy\_entropy)**

**# Print accuracy\_gini**

**print('Accuracy achieved by using the gini index: ', accuracy\_gini)**

<script.py> output:

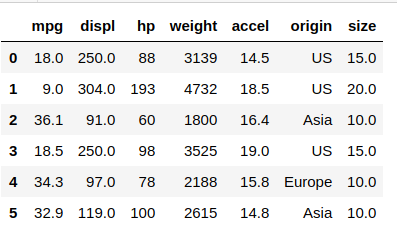
Accuracy achieved by using entropy: 0.929824561404

Accuracy achieved by using the gini index: 0.929824561404

Notice how the two models achieve exactly the same accuracy. Most of the time, the gini index and entropy lead to the same results. The gini index is slightly faster to compute and is the default criterion used in the DecisionTreeClassifier model of scikit-learn.

# **Decision tree for regression**

Let’s train a regression model with the automobile data:



# For the sake of simplicity we’ll consider only one feature: displ (displacement).

**So, weĺl need to predict the energy consumption (mpg) of the automobiles.**

# **Train your first regression tree**

In this exercise, you'll train a regression tree to predict the mpg (miles per gallon) consumption of cars in the [**auto-mpg dataset**](https://www.kaggle.com/uciml/autompg-dataset) using all the six available features.

The dataset is processed for you and is split to 80% train and 20% test. The features matrix X\_train and the array y\_train are available in your workspace.

**# Import DecisionTreeRegressor from sklearn.tree**

**from sklearn.tree import DecisionTreeRegressor**

**# Instantiate dt**

**dt = DecisionTreeRegressor(max\_depth=8,**

**min\_samples\_leaf=0.13,**

**random\_state=3)**

**# Fit dt to the training set**

**dt.fit(X\_train, y\_train)**

**Evaluate the regression tree**

In this exercise, you will evaluate the test set performance of dt using the Root Mean Squared Error (RMSE) metric. The RMSE of a model measures, on average, how much the model's predictions differ from the actual labels. The RMSE of a model can be obtained by computing the square root of the model's Mean Squared Error (MSE).

The features matrix X\_test, the array y\_test, as well as the decision tree regressor dt that you trained in the previous exercise are available in your workspace.

**# Import mean\_squared\_error from sklearn.metrics as MSE**

**from sklearn.metrics import mean\_squared\_error as MSE**

**# Compute y\_pred**

**y\_pred = dt.predict(X\_test)**

**# Compute mse\_dt**

**mse\_dt = MSE(y\_test, y\_pred)**

**# Compute rmse\_dt**

**rmse\_dt = pow(mse\_dt, 1/2)**

**# Print rmse\_dt**

**print("Test set RMSE of dt: {:.2f}".format(rmse\_dt))**

<script.py> output:

Test set RMSE of dt: 4.37

**Linear regression vs regression tree**

In this exercise, you'll compare the test set RMSE of **dt** to that achieved by a linear regression model. We have already instantiated a linear regression model **lr** and trained it on the same dataset as **dt**.

The features matrix **X\_test,** the array of labels **y\_test**, the trained linear regression model **lr**, **mean\_squared\_error** function which was imported under the alias MSE and **rmse\_dt** from the previous exercise are available in your workspace.

**# Predict test set labels**

**y\_pred\_lr = lr.predict(X\_test)**

**# Compute mse\_lr**

**mse\_lr = MSE(y\_test, y\_pred\_lr)**

**# Compute rmse\_lr**

**rmse\_lr = pow(mse\_lr, 1/2)**

**# Print rmse\_lr**

**print('Linear Regression test set RMSE: {:.2f}'.format(rmse\_lr))**

**# Print rmse\_dt**

**print('Regression Tree test set RMSE: {:.2f}'.format(rmse\_dt))**

<script.py> output:

Linear Regression test set RMSE: 5.10

Regression Tree test set RMSE: 4.37

**Chapter 2: The Bias-Variance Tradeoff**