**Exercise**

**Exercise**

**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.

**Instructions**

**100 XP**

* Import DecisionTreeClassifier from sklearn.tree.
* Instantiate a DecisionTreeClassifier dt of maximum depth equal to 6.
* Fit dt to the training set.
* Predict the test set labels and assign the result to y\_pred.

[**Take Hint (-30 XP)**](javascript:void(0))

**Incorrect Submission**

Have you specified the arguments for dtc() using the right syntax?

# Import DecisionTreeClassifier from sklearn.tree

from sklearn.tree import DecisionTreeClassifier as dtc

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

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

# Fit dt to the training set

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

# Predict test set labels

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

print(y\_pred[0:5])

Traceback (most recent call last):

File "script.py", line 5, in <module>

dt = dtc(depth=6, random\_state=SEED)

TypeError: \_\_init\_\_() got an unexpected keyword argument 'depth'

In [1]: help(dtc.\_\_init\_\_)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

help(dtc.\_\_init\_\_)

NameError: name 'dtc' is not defined

In [2]:

<script.py> output:

[0 0 0 1 0]

In [2]:

+100 XP

Awesome! You've just trained your first classification tree! You can see the first five predictions made by the fitted tree on the test set in the console. In the next exercise, you'll evaluate the tree's performance on the entire test set.

**Exercise**

**Exercise**

**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.

**Instructions**

**100 XP**

* Import the function accuracy\_score from sklearn.metrics.
* Predict the test set labels and assign the obtained array to y\_pred.
* Evaluate the test set accuracy score of dt by calling accuracy\_score() and assign the value to acc.

[**Take Hint (-30 XP)**](javascript:void(0))

# 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\_test, y\_pred)

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

<script.py> output:

Test set accuracy: 0.89

In [1]:

+100 XP

Not bad! Using only two features, your tree was able to achieve an accuracy of 89%!

**Exercise**

**Exercise**

**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.

**Instructions**

**100 XP**

* Import LogisticRegression from sklearn.linear\_model.
* Instantiate a LogisticRegression model and assign it to logreg.
* Fit logreg to the training set.
* Review the plot generated by plot\_labeled\_decision\_regions().

[**Take Hint (-30 XP)**](javascript:void(0))

# Import LogisticRegression from sklearn.linear\_model

from sklearn.linear\_model import LogisticRegression as lr

# Instatiate logreg

logreg = lr(random\_state=1)

# Fit logreg to the training set

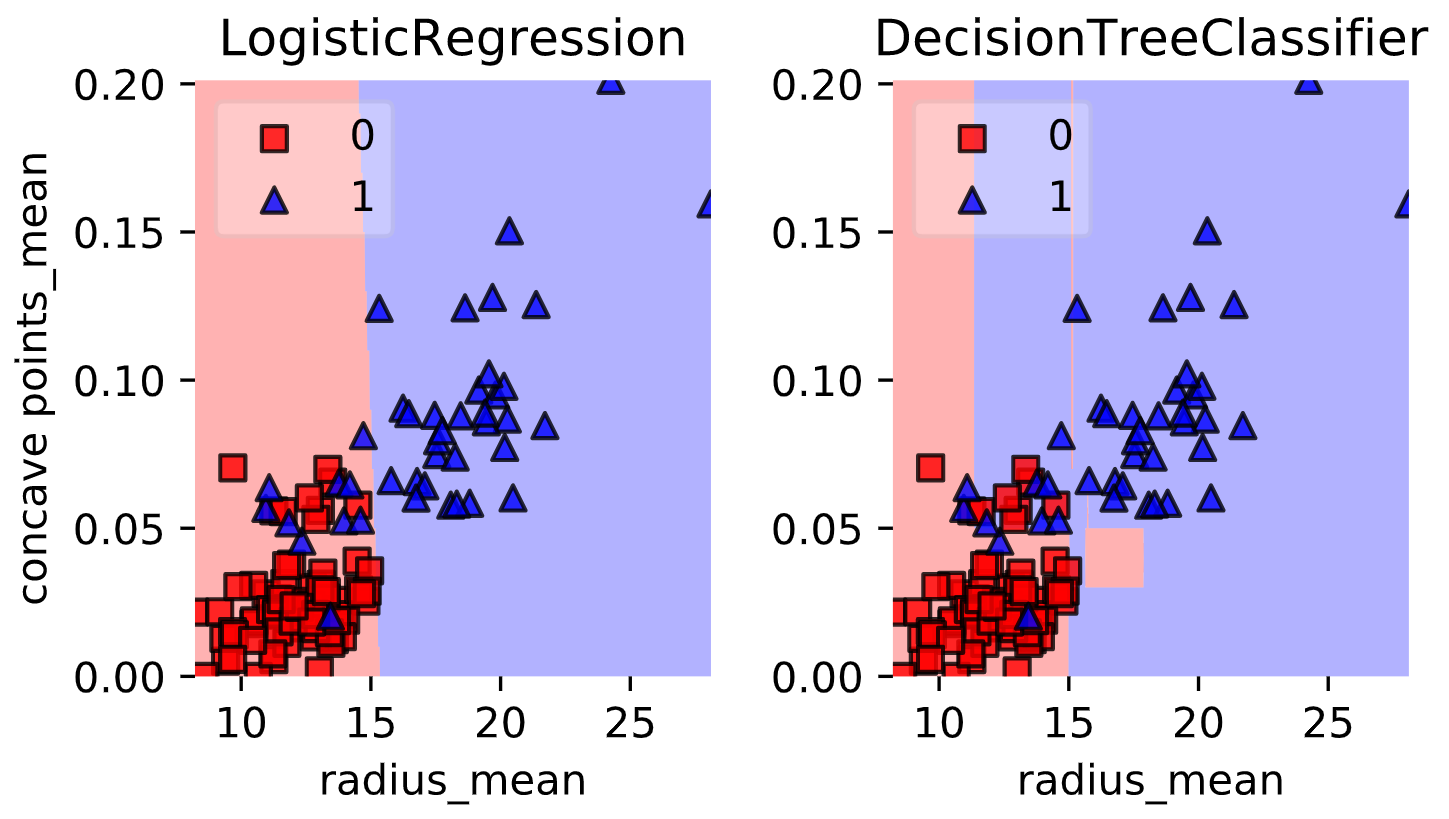
logreg.fit(X\_train, y\_train)

# 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)



+100 XP

Great work! 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.