Introduction to PySpark (Model Tuning and Selection)

Create the Modeler

The `Estimator` you'll be using is a `LogisticRegression` from the `pyspark.ml.classification` submodule.

```
# Import LogisticRegression
from pyspark.ml.classification import LogisticRegression

# Create a LogisticRegression Estimator
lr = LogisticRegression()
```

Create the evaluator

The first thing you need when doing cross validation for model selection is a way to compare different models. Luckily, the `pyspark.ml.evaluation` submodule has classes for evaluating different kinds of models. Your model is a binary classification model, so you'll be using the `BinaryClassificationEvaluator` from the `pyspark.ml.evaluation` module.

This evaluator calculates the area under the ROC. This is a metric that combines the two kinds of errors a binary classifier can make (false positives and false negatives) into a simple number. You'll learn more about this towards the end of the chapter!

```
# Import the evaluation submodule
import pyspark.ml.evaluation as evals

# Create a BinaryClassificationEvaluator
evaluator = evals.BinaryClassificationEvaluator(metricName = "areaUnderROC")
```

Make a Grid

Next, you need to create a grid of values to search over when looking for the optimal hyperparameters. The submodule `pyspark.ml.tuning` includes a class called `ParamGridBuilder` that does just that (maybe you're starting to notice a pattern here; PySpark has a submodule for just about everything!).

You'll need to use the `.addGrid()` and `.build()` methods to create a grid that you can use for cross validation. The `.addGrid()` method takes a model parameter (an attribute of the model `Estimator`, `lr`, that you created a few exercises ago) and a list of values that you want to try. The `.build()` method takes no arguments, it just returns the grid that you'll use later.

```
# Import the tuning submodule
import pyspark.ml.tuning as tune

# Create the parameter grid
grid = tune.ParamGridBuilder()

# Add the hyperparameter
grid = grid.addGrid(lr.regParam, np.arange(0, .1, .01))
grid = grid.addGrid(lr.elasticNetParam, [0, 1])

# Build the grid
grid = grid.build()
```

Make the Validator

The submodule `pyspark.ml.tuning` also has a class called `CrossValidator` for performing cross validation. This `Estimator` takes the modeler you want to fit, the grid of hyperparameters you created, and the evaluator you want to use to compare your models.

The submodule `pyspark.ml.tune` has already been imported as `tune`. You'll create the `CrossValidator` by passing it the logistic regression `Estimator` `lr`, the parameter `grid`, and the `evaluator` you created in the previous exercises.

```
# Create the CrossValidator
cv = tune.CrossValidator(estimator=lr,
                         estimatorParamMaps=grid,
                         evaluator=evaluator
                         )
```

Fit the Models

You're finally ready to fit the models and select the best one!

Unfortunately, cross validation is a very computationally intensive procedure. Fitting all the models would take too long on DataCamp.

To do this locally you would use the code:

```
# Fit cross validation models
models = cv.fit(training)

# Extract the best model
best_lr = models.bestModel
```

Remember, the training data is called `training` and you're using `lr` to fit a logistic regression model. Cross validation selected the parameter values `regParam=0` and `elasticNetParam=0` as being the best. These are the default values, so you don't need to do anything else with `lr` before fitting the model.

```
# Call lr.fit()
best_lr = lr.fit(training)

# Print best_lr
print(best_lr)
```

Evaluate the Model

Remember the test data that you set aside waaaaaay back in chapter 3? It's finally time to test your model on it! You can use the same evaluator you made to fit the model.

```
# Use the model to predict the test set
test_results = best_lr.transform(test)

# Evaluate the predictions
print(evaluator.evaluate(test_results))
```