Using Cross Validation

In this exercise, you will use cross-validation to optimize parameters for a regression model.

Prepare the Data

First, import the libraries you will need and prepare the training and test data:

In [ ]:

In [1]:
```
# Import Spark SQL and Spark ML libraries
from pyspark.sql.types import *
from pyspark.sql.functions import *

from pyspark.ml import Pipeline
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import RegressionEvaluator

# Load the source data
csv = spark.read.csv('wasb:///data/flights.csv', inferSchema=True, header=True)

# Select features and label
data = csv.select("DayofMonth", "DayOfWeek", "OriginAirportID", "DestAirportID",

# Split the data
splits = data.randomSplit([0.7, 0.3])
train = splits[0]
test = splits[1].withColumnRenamed("label", "trueLabel")
```

```
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI
7	application_1613740567063_0010	pyspark	idle	Link (http://hn1-rayspa.pk5nseqskssevkfuqghr2123nc.ix.int

```
SparkSession available as 'spark'.
```

Define the Pipeline

Now define a pipeline that creates a feature vector and trains a regression model

In [2]:
```
# Define the pipeline
assembler = VectorAssembler(inputCols = ["DayofMonth", "DayOfWeek", "OriginAirpor
lr = LinearRegression(labelCol="label",featuresCol="features")
pipeline = Pipeline(stages=[assembler, lr])
```

Tune Parameters

You can tune parameters to find the best model for your data. To do this you can use the **CrossValidator** class to evaluate each combination of parameters defined in a **ParameterGrid** against multiple *folds* of the data split into training and validation datasets, in order to find the best performing parameters. Note that this can take a long time to run because every parameter combination is tried multiple times.

In [3]:
```
paramGrid = ParamGridBuilder().addGrid(lr.regParam, [0.3, 0.01]).addGrid(lr.maxIt
cv = CrossValidator(estimator=pipeline, evaluator=RegressionEvaluator(), estimato

model = cv.fit(train)
```

Test the Model

Now you're ready to apply the model to the test data.

In [4]:
```
prediction = model.transform(test)
predicted = prediction.select("features", "prediction", "trueLabel")
predicted.show()
```

```
+--------------------+------------------+---------+
|            features|        prediction|trueLabel|
+--------------------+------------------+---------+
|[1.0,1.0,10140.0,...|  73.62671581440335|       94|
|[1.0,1.0,10140.0,...| -8.892516123547134|      -14|
|[1.0,1.0,10140.0,...| -6.882329878216394|      -11|
|[1.0,1.0,10140.0,...| -5.877236755551024|      -12|
|[1.0,1.0,10140.0,...| -4.872143632885654|      -11|
|[1.0,1.0,10140.0,...| 17.239905065752485|       19|
|[1.0,1.0,10140.0,...|  31.31120878306766|       41|
|[1.0,1.0,10140.0,...| -5.884891239614667|       -9|
|[1.0,1.0,10140.0,...| -5.884891239614667|       -5|
|[1.0,1.0,10140.0,...|-1.8645187489531878|       -1|
|[1.0,1.0,10140.0,...|-0.8594256262878179|        2|
|[1.0,1.0,10140.0,...|-13.927027945312835|      -13|
|[1.0,1.0,10140.0,...| -5.886282963989875|       -2|
|[1.0,1.0,10140.0,...| -4.881189841324505|       -9|
|[1.0,1.0,10140.0,...|-2.8710035959937654|       -3|
|[1.0,1.0,10140.0,...|-15.156652600511714|      -28|
|[1.0,1.0,10140.0,...|-12.141373232515605|      -17|
|[1.0,1.0,10140.0,...|  838.1674085423874|      812|
|[1.0,1.0,10140.0,...| -8.265508122813193|        3|
|[1.0,1.0,10140.0,...| -6.255321877482453|      -17|
+--------------------+------------------+---------+
only showing top 20 rows
```

Examine the Predicted and Actual Values

You can plot the predicted values against the actual values to see how accurately the model has predicted. In a perfect model, the resulting scatter plot should form a perfect diagonal line with each predicted value being identical to the actual value - in practice, some variance is to be expected. Run the cells below to create a temporary table from the **predicted** DataFrame and then retrieve the predicted and actual label values using SQL. You can then display the results as a scatter plot, specifying **-** as the function to show the unaggregated values.

In [5]:
```
predicted.createOrReplaceTempView("regressionPredictions")
```

In [6]:
```
%%sql
SELECT trueLabel, prediction FROM regressionPredictions
```

Type: Table Pie Scatter Line Area Bar

Encoding:

X trueLabel

Y prediction Func. -

Log scale X

Log scale Y

Retrieve the Root Mean Square Error (RMSE)

There are a number of metrics used to measure the variance between predicted and actual values. Of these, the root mean square error (RMSE) is a commonly used value that is measured in the same units as the prediced and actual values - so in this case, the RMSE indicates the average number of minutes between predicted and actual flight delay values. You can use the **RegressionEvaluator** class to retrieve the RMSE.

In [7]:
```
evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction",
rmse = evaluator.evaluate(prediction)
print "Root Mean Square Error (RMSE):", rmse
```

```
Root Mean Square Error (RMSE): 13.246528636
```

In [ ]: