

Predicting Model with Logistic Regression

A Hortonworks University

Hadoop Training Course

Title: LAB GUIDE: Data Science for the Hortonworks Data Platform

Revision 2

Copyright **©** 2015 Hortonworks Inc 2015 All rights reserved.

All Rights Reserved. Hadoop and the Hadoop elephant logo are trademarks of the Apache Software Foundation.

The contents of this course and all its related materials, including lab exercises and files, are Copyright **©** 2015 Hortonworks Inc.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means electronic, mechanical, photocopying, recording or otherwise without the prior written permission of Hortonworks Inc.

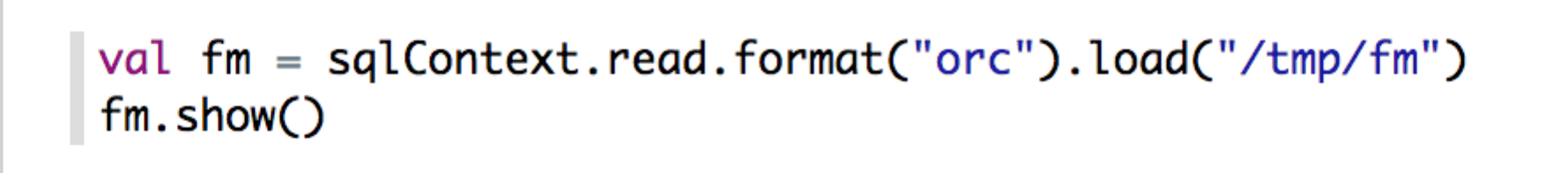
Lab: Predicting Model with Logistic Regression

Lab: Predicting Model with Logistic Regression

|  |  |
| --- | --- |
| **Objective:** | Work with Spark ML |
| **Successful Outcome:** | Create a model to classify data |
| **Before You Begin:** | Your HDP cluster should be up and running in the classroom VM. |

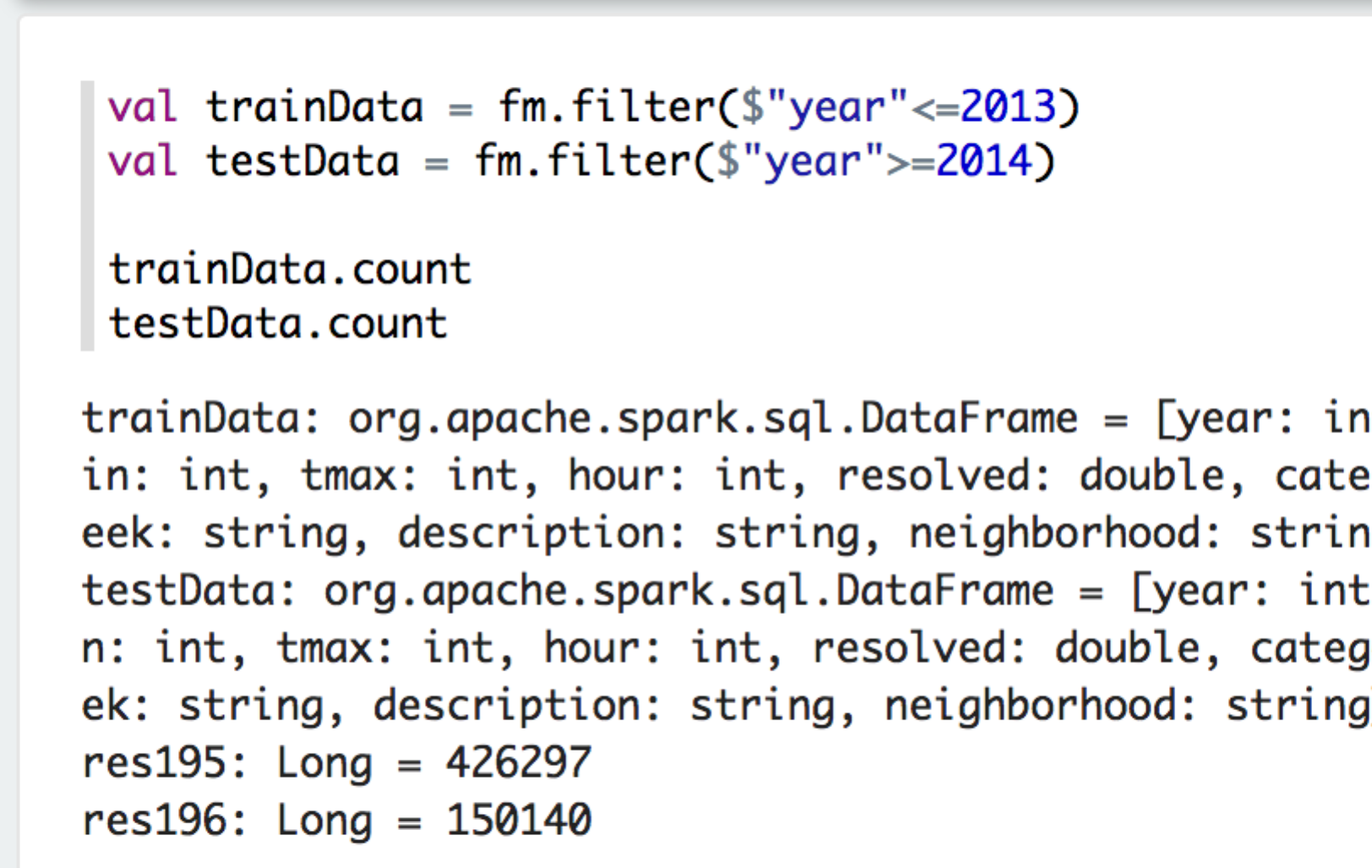
1. Now that we have our features built, lets go ahead and build a model for predicting if a crime will be resolved or not.
   1. First we need to read in the orc file we created previously.

val fm = sqlContext.read.format("orc").load("/tmp/fm")  
fm.show()



* 1. Now lets divide up our data into train and test data set. Normally we’d select a random distribution, but for now, lets use test data as year greater than or equal to 2014 and the train data as the rest.

val trainData = fm.filter($"year"<=2013)  
val testData = fm.filter($"year">=2014)  
  
trainData.count  
testData.count



* 1. Import the ML libraries that will be needed for this part

import org.apache.spark.ml.feature.StringIndexer  
import org.apache.spark.ml.feature.VectorAssembler  
import org.apache.spark.ml.Pipeline

* 1. Spark doesn’t like numbers, so lets go ahead and index our catergory, dayofweek, district, neighborhood and resolved or not. Once that is done, lets assemble these into a VextorAssembler.

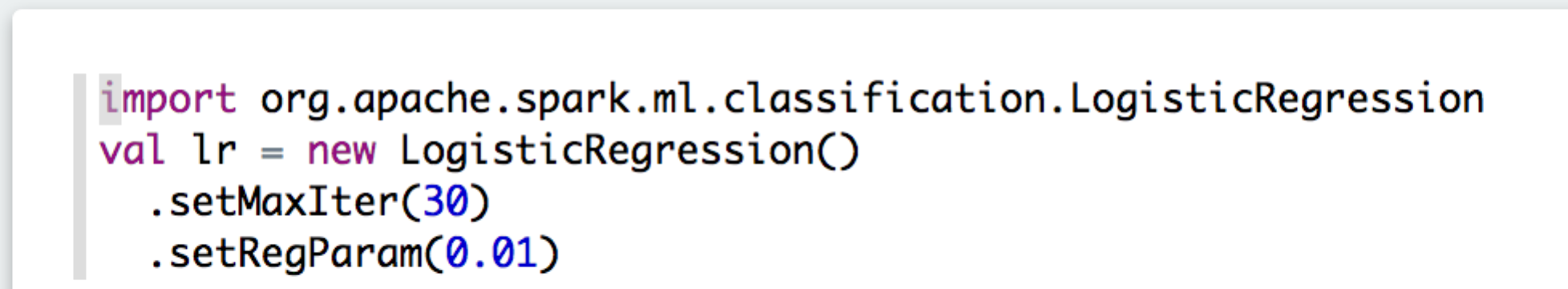
val inx1 = new StringIndexer().setInputCol("category").setOutputCol("cat-inx")  
val inx2 = new StringIndexer().setInputCol("dayofweek").setOutputCol("dow-inx")  
val inx3 = new StringIndexer().setInputCol("district").setOutputCol("dis-inx")  
val inx4 = new StringIndexer().setInputCol("neighborhood").setOutputCol("ngh-inx")  
val inx5 = new StringIndexer().setInputCol("resolved").setOutputCol("label")  
val vecAssembler = new VectorAssembler()  
.setInputCols(Array("month", "hour", "prcp", "tmin", "tmax","cat-inx", "dow-inx", "dis-inx", "ngh-inx"))  
.setOutputCol("features")



* 1. Now we’re ready to set up our LogisticRegression Algorithm to be used after we import it.

import org.apache.spark.ml.classification.LogisticRegression

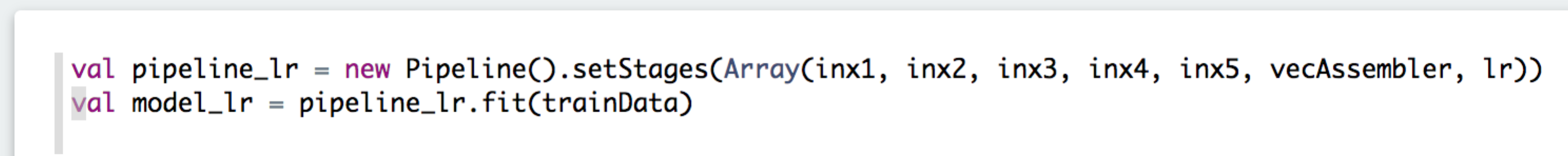
val lr = new LogisticRegression()  
 .setMaxIter(30)  
 .setRegParam(0.01)



* 1. Lets build a pipeline of our stages and logistic regression to create a model to predict weather crime will be resolved or not. Also build the model using the training data.

val pipeline\_lr = new Pipeline().setStages(Array(inx1, inx2, inx3, inx4, inx5, vecAssembler, lr))

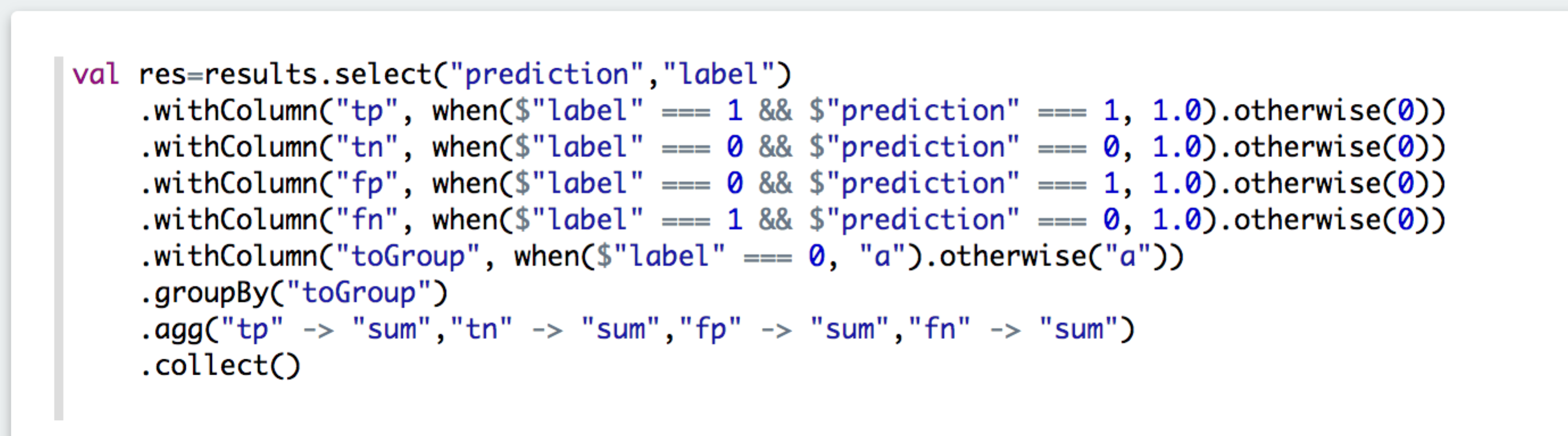
val model\_lr = pipeline\_lr.fit(trainData)



* 1. Now that we have built our model, lets go ahead and test it on our test data to see how good it predicts by using our precision, recall and accuracy metrics.

val results = model\_lr.transform(testData).cache()

val res=results.select("prediction","label")  
 .withColumn("tp", when($"label" === 1 && $"prediction" === 1, 1.0).otherwise(0))  
 .withColumn("tn", when($"label" === 0 && $"prediction" === 0, 1.0).otherwise(0))  
 .withColumn("fp", when($"label" === 0 && $"prediction" === 1, 1.0).otherwise(0))  
 .withColumn("fn", when($"label" === 1 && $"prediction" === 0, 1.0).otherwise(0))  
 .withColumn("toGroup", when($"label" === 0, "a").otherwise("a")) .groupBy("toGroup")  
 .agg("tp" -> "sum","tn" -> "sum","fp" -> "sum","fn" -> "sum")  
 .collect()



val tp = res(0).getDouble(1)  
val tn = res(0).getDouble(2)  
val fp = res(0).getDouble(3)  
val fn = res(0).getDouble(4)  
val precision = tp / (tp+fp)  
val recall = tp / (tp+fn)  
val accuracy = (tp+tn) / (tp+tn+fp+fn)

