Machine Learning using Spark:

Module 5, Lesson 6  
Build a Predictive Analysis Application Hands-On Lab

## Overview

In this lab, students will learn how to build a predictive analysis algorithm on a Spark cluster in HDInsight. Students will use the Jupyter notebook to build and test the application. The application uses food inspection data acquired through the *City of Chicago* data portal.

## Objectives

In this hands-on lab you will learn how to:

* Setup a dataframe to accept the input data
* Create functions to reformat the input data into a form that can be used by MLlib
* Parse natural language input data into a feature vector form
* Setup test data and run the new model against the test data

## Prerequisites

The following are required to complete this hands-on lab:

* A web browser
* A Spark cluster in Azure HDInsight

Note : The Azure portal is continually improved and changed. The steps in this exercise reflect the user interface of the Microsoft Azure portal at the time of writing, but may not match the latest design of portal.

## Exercises

This hands-on lab includes the following exercises:

* Exercise 1: Write a predictive analysis application

## Background

Classification is a very common machine learning task. It is the process of reviewing input data and sorting them into categories. The classification algorithm determines how to assign “labels” to the input data provided by the user. For example, take a classification algorithm that accepts stock information as input and classifies the stock into two categories: sell stocks and retain stocks.

Logistic regression is an algorithm that can be used for classification. It is particularly useful for classifying the input data into one of two categories, typically true or false. This type of classification is called a binary classification.

## Exercise 1: Write Predictive Analysis Application

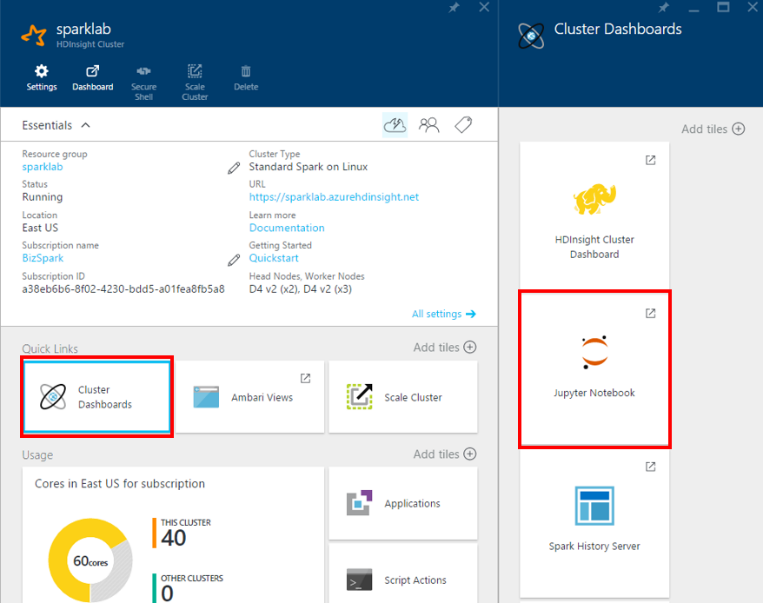
You will use Spark to perform some predictive analysis on food inspection data (Food\_Inspections1.csv) that was acquired through the City of Chicago data portal. This dataset contains information about food inspections that were conducted in Chicago, including information about each food establishment that was inspected, the violations that were found (if any), and the results of the inspection. The CSV data file is already available in the storage account associated with the cluster at /HdiSamples/HdiSamples/FoodInspectionData/Food\_Inspections1.csv.

In the steps below, you develop a model to see what it takes to pass or fail a food inspection.Create Jupyter notebook with PySpark kernel

1. Open a Jupyter Notebook

From the Azure Portal, navigate to the Spark cluster that the student just created. It can be found under **Browse All>HDInsight Clusters**.

From QuickLinks, select **Cluster Dashboards** and then select **Jupyter Notebook**. If prompted enter the admin credentials for the cluster. This will open up a browser window loaded with Jupyter notebook



Create **New** and select a PySpark kernel..

You may also reach the Jupyter Notebook for your cluster by opening the following URL in your browser. https://CLUSTERNAME.azurehdinsight.net/jupyter (Replace CLUSTERNAME with the name of student cluster)

1. Name the PySpark kernel with a user friendly name



1. Import required types for this application

The PySpark kernel includes the context for Spark and Hive. The student does not have to create these contexts explicitly.

To begin the exercise, import the types required for this scenario. To do so, paste the following code snippet in a cell and press SHIFT + ENTER. Pressing SHIFT + ENTER executes the entry in the current cell and moves the cursor to the next cell.

from pyspark.ml import Pipeline

from pyspark.ml.classification import LogisticRegression

from pyspark.ml.feature import HashingTF, Tokenizer

from pyspark.sql import Row

from pyspark.sql.functions import UserDefinedFunction

from pyspark.sql.types import \*

1. Construct Input DataFrame

The sqlContext can be used to perform transformations on structured data. First, load the sample data – Food\_Inspection1.csv into a Spark SQL dataframe

The data is in raw CSV format. We can use Python’s CSV library to parse each line.

In an empty cell, paste the following code example and press SHIFT + ENTER.

def csvParse(s):

import csv

from StringIO import StringIO

sio = StringIO(s)

value = csv.reader(sio).next()

sio.close()

return value

inspections = sc.textFile('wasb:///HdiSamples/HdiSamples/FoodInspectionData/Food\_Inspections1.csv')\

.map(csvParse)

1. Inspect the data

The CSV file has been converted to a RDD. We can display one row of data to get a better understanding of the schema of the input data.

In an empty cell, paste the following code example and press SHIFT + ENTER.

inspections.take(1)

You should see output similar to:

# -----------------

# THIS IS AN OUTPUT

# -----------------

[['413707',

'LUNA PARK INC',

'LUNA PARK DAY CARE',

'2049789',

"Children's Services Facility",

'Risk 1 (High)',

'3250 W FOSTER AVE ',

'CHICAGO',

'IL',

'60625',

'09/21/2010',

'License-Task Force',

'Fail',

'24. DISH WASHING FACILITIES: PROPERLY DESIGNED, CONSTRUCTED, MAINTAINED, INSTALLED, LOCATED AND OPERATED - Comments: All dishwashing machines must be of a type that complies with all requirements of the plumbing section of the Municipal Code of Chicago and Rules and Regulation of the Board of Health. OBSEVERD THE 3 COMPARTMENT SINK BACKING UP INTO THE 1ST AND 2ND COMPARTMENT WITH CLEAR WATER AND SLOWLY DRAINING OUT. INST NEED HAVE IT REPAIR. CITATION ISSUED, SERIOUS VIOLATION 7-38-030 H000062369-10 COURT DATE 10-28-10 TIME 1 P.M. ROOM 107 400 W. SURPERIOR. | 36. LIGHTING: REQUIRED MINIMUM FOOT-CANDLES OF LIGHT PROVIDED, FIXTURES SHIELDED - Comments: Shielding to protect against broken glass falling into food shall be provided for all artificial lighting sources in preparation, service, and display facilities. LIGHT SHIELD ARE MISSING UNDER HOOD OF COOKING EQUIPMENT AND NEED TO REPLACE LIGHT UNDER UNIT. 4 LIGHTS ARE OUT IN THE REAR CHILDREN AREA,IN THE KINDERGARDEN CLASS ROOM. 2 LIGHT ARE OUT EAST REAR, LIGHT FRONT WEST ROOM. NEED TO REPLACE ALL LIGHT THAT ARE NOT WORKING. | 35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS - Comments: The walls and ceilings shall be in good repair and easily cleaned. MISSING CEILING TILES WITH STAINS IN WEST,EAST, IN FRONT AREA WEST, AND BY THE 15MOS AREA. NEED TO BE REPLACED. | 32. FOOD AND NON-FOOD CONTACT SURFACES PROPERLY DESIGNED, CONSTRUCTED AND MAINTAINED - Comments: All food and non-food contact equipment and utensils shall be smooth, easily cleanable, and durable, and shall be in good repair. SPLASH GUARDED ARE NEEDED BY THE EXPOSED HAND SINK IN THE KITCHEN AREA | 34. FLOORS: CONSTRUCTED PER CODE, CLEANED, GOOD REPAIR, COVING INSTALLED, DUST-LESS CLEANING METHODS USED - Comments: The floors shall be constructed per code, be smooth and easily cleaned, and be kept clean and in good repair. INST NEED TO ELEVATE ALL FOOD ITEMS 6INCH OFF THE FLOOR 6 INCH AWAY FORM WALL. ',

'41.97583445690982',

'-87.7107455232781',

'(41.97583445690982, -87.7107455232781)']]

1. Create schema for input file

The output gives us a pretty good understanding of the schema. The file includes name of establishment, type of establishment, the inspection data, the location, etc. Select a few columns that seem useful for our predictive analysis and group the results as a dataframe that we can pass to our Transformers and Estimators. We will also create a temporary table to facilitate querying the data.

In an empty cell, paste the following code example and press SHIFT + ENTER.

schema = StructType([

StructField("id", IntegerType(), False),

StructField("name", StringType(), False),

StructField("results", StringType(), False),

StructField("violations", StringType(), True)])

df = sqlContext.createDataFrame(inspections.map(lambda l: (int(l[0]), l[1], l[12], l[13])) , schema)

df.registerTempTable('CountResults')

1. Inspect the data that has been imported

The data has been saved as a dataframe of four columns (id, name, results and violations) on which we can perform the analysis. A temporary table called *CountResults* has also been created. Review a small sample of the data.

In an empty cell, paste the following code example and press SHIFT + ENTER.

df.show(5)

This should give output similar to the following:

+------+--------------------+-------+--------------------+

| id| name|results| violations|

+------+--------------------+-------+--------------------+

|413707| LUNA PARK INC| Fail|24. DISH WASHING ...|

|391234| CAFE SELMARIE| Fail|2. FACILITIES TO ...|

|413751| MANCHU WOK| Pass|33. FOOD AND NON-...|

|413708|BENCHMARK HOSPITA...| Pass| |

|413722| JJ BURGER| Pass| |

+------+--------------------+-------+--------------------+

only showing top 5 rows

1. Understand the data

Get a sense of what our dataset contains. For example, what are the different values in the *results* column..

In an empty cell, paste the following code example and press SHIFT + ENTER.

df.select('results').distinct().show()

This should give output similar to the following:

# -----------------

# THIS IS AN OUTPUT

# -----------------

+--------------------+

| results|

+--------------------+

| Fail|

|Business Not Located|

| Pass|

| Pass w/ Conditions|

| Out of Business|

+--------------------+

1. Visualize the distribution of the outcomes

Let’s visualize the results data to get a better sense of the distribution of these outcomes. We have saved *CountResults*, a temporary table, to facilitate running SQL queries on the data.

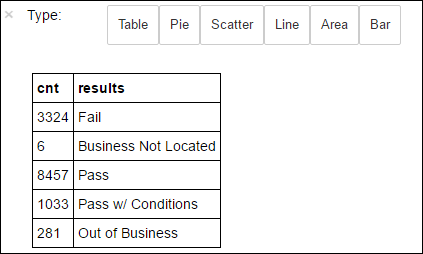
In an empty cell, paste the following code example and press SHIFT + ENTER.

%%sql -o countResultsdf

SELECT results, COUNT(results) AS cnt FROM CountResults GROUP BY results

The *%%SQL* magic followed by -o countResultsdf saves a persisted copy of the query on the local machine.

Output should be similar to the following:



We will use Matplotlib, a library used to construct visualization of data, to construct a visualization of the data. The plot must be created from a locally persisted dataframe. The *%%local* magic ensures that the code is run locally.

In an empty cell, paste the following code example and press SHIFT + ENTER.

%%local

%matplotlib inline

import matplotlib.pyplot as plt

labels = countResultsdf['results']

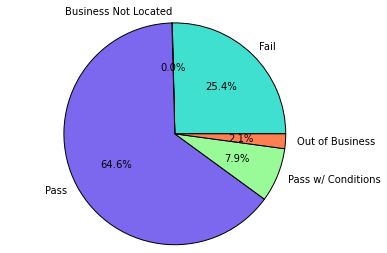
sizes = countResultsdf['cnt']

colors = ['turquoise', 'seagreen', 'mediumslateblue', 'palegreen', 'coral']

plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=colors)

plt.axis('equal')

You should see output similar to the following:



There are 5 distinct results of an inspection. They are:

* Business not located
* Fail
* Pass
* Pass with conditions
* Out of Business

1. Develop model of inspection outcome

Based on the input data and results of the inspection, we develop a model for the classification algorithm. We are using logistic regression which is a binary classification algorithm, therefore it makes sense to classify the results into 2 groups; pass or fail. A pass with conditions is still a pass as well and the model should be trained as such. Data with other results ("Business Not Located", "Out of Business") are not useful so will removed from the training set. The two categories only represent a small minority so this should be ok.

1. Convert current dataframe to fit the mode’

Convert existing dataframe into a new dataframe where each inspection is represented by a label-violations pair. The label will be pass inspection or did not pass inspection and represented as 0.0 for failure and 1.0 for pass.

In an empty cell, paste the following code example and press SHIFT + ENTER.

def labelForResults(s):

if s == 'Fail':

return 0.0

elif s == 'Pass w/ Conditions' or s == 'Pass':

return 1.0

else:

return -1.0

label = UserDefinedFunction(labelForResults, DoubleType())

labeledData = df.select(label(df.results).alias('label'), df.violations).where('label >= 0')

1. Inspect the new dataframe

labeledData.take(1)

You should see output similar to the following:

# -----------------

# THIS IS AN OUTPUT

# -----------------

[Row(label=0.0, violations=u"41. PREMISES MAINTAINED FREE OF LITTER, UNNECESSARY ARTICLES, CLEANING EQUIPMENT PROPERLY STORED - Comments: All parts of the food establishment and all parts of the property used in connection with the operation of the establishment shall be kept neat and clean and should not produce any offensive odors. REMOVE MATTRESS FROM SMALL DUMPSTER. | 35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS - Comments: The walls and ceilings shall be in good repair and easily cleaned. REPAIR MISALIGNED DOORS AND DOOR NEAR ELEVATOR. DETAIL CLEAN BLACK MOLD LIKE SUBSTANCE FROM WALLS BY BOTH DISH MACHINES. REPAIR OR REMOVE BASEBOARD UNDER DISH MACHINE (LEFT REAR KITCHEN). SEAL ALL GAPS. REPLACE MILK CRATES USED IN WALK IN COOLERS AND STORAGE AREAS WITH PROPER SHELVING AT LEAST 6' OFF THE FLOOR. | 38. VENTILATION: ROOMS AND EQUIPMENT VENTED AS REQUIRED: PLUMBING: INSTALLED AND MAINTAINED - Comments: The flow of air discharged from kitchen fans shall always be through a duct to a point above the roofline. REPAIR BROKEN VENTILATION IN MEN'S AND WOMEN'S WASHROOMS NEXT TO DINING AREA. | 32. FOOD AND NON-FOOD CONTACT SURFACES PROPERLY DESIGNED, CONSTRUCTED AND MAINTAINED - Comments: All food and non-food contact equipment and utensils shall be smooth, easily cleanable, and durable, and shall be in good repair. REPAIR DAMAGED PLUG ON LEFT SIDE OF 2 COMPARTMENT SINK. REPAIR SELF CLOSER ON BOTTOM LEFT DOOR OF 4 DOOR PREP UNIT NEXT TO OFFICE.")]

1. Create logistic regression model from input dataframe

We now have to convert the labeled data into a format that can be analyzed by logistic regression. The input should be a set of label-feature vector pairs, where the vector of numbers represent the relevance of the feature in some way. The “violations” column is in free form text format and must somehow be converted to an array of real numbers that a machine can easily understand.

A standard machine learning technique for processing natural language is to create an index of distinct words with a relative frequency count of that word.

MLLib provides an easy way to perform this operation. We first parse the violations string into a set of tokens to get the individual words of the string. We then use HashingTF to convert each set of tokens into a feature vector. This feature vector is what we will pass the logistic regression algorithm to construct a model. We'll conduct all of these steps in sequence using a "pipeline".

In an empty cell, paste the following code example and press SHIFT + ENTER.

tokenizer = Tokenizer(inputCol="violations", outputCol="words")

hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")

lr = LogisticRegression(maxIter=10, regParam=0.01)

pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

model = pipeline.fit(labeledData)

1. Evaluate the model on a separate test dataset

The model we have just created can be used to predict the results of new inspections based on the violations observed. We will now use a second dataset, Food\_Inspections2.csv to evaluate the strength of our newly created model. This second dataset is in the same directory as our original dataset.

We will create a new dataframe – predictionsDF – that contains the predictions generated by our new model This dataframe will also be saved into a temporary table Predictions to facilitate querying.

In an empty cell, paste the following code example and press SHIFT + ENTER.

testData = sc.textFile('wasb:///HdiSamples/HdiSamples/FoodInspectionData/Food\_Inspections2.csv')\

.map(csvParse) \

.map(lambda l: (int(l[0]), l[1], l[12], l[13]))

testDf = sqlContext.createDataFrame(testData, schema).where("results = 'Fail' OR results = 'Pass' OR results = 'Pass w/ Conditions'")

predictionsDf = model.transform(testDf)

predictionsDf.registerTempTable('Predictions')

predictionsDf.columns

You should see output similar to the following:

# -----------------

# THIS IS AN OUTPUT

# -----------------

['id',

'name',

'results',

'violations',

'words',

'features',

'rawPrediction',

'probability',

'prediction']

Lets look at the prediction for the first entry in the test data set.

predictionsDf.take(1)

The model we generated and its model.transform() method will apply the same transformation to any new data with the same schema and arrive at a prediction as to its results. We can do some simple statistics to measure how accurate our model is.

numSuccesses = predictionsDf.where("""(prediction = 0 AND results = 'Fail') OR

(prediction = 1 AND (results = 'Pass' OR

results = 'Pass w/ Conditions'))""").count()

numInspections = predictionsDf.count()

print "There were", numInspections, "inspections and there were", numSuccesses, "successful predictions"

print "This is a", str((float(numSuccesses) / float(numInspections)) \* 100) + "%", "success rate"

The output will look similar to :

# -----------------

# THIS IS AN OUTPUT

# -----------------

There were 9315 inspections and there were 8087 successful predictions

This is a 86.8169618894% success rate

We see that using logistic regression, Spark has produced a fairly accurate model to predict the outcome of a new inspection based on examining the violations.

1. Finally, create a presentation to visualize the accuracy of the prediction

Using the temporary table that we created earlier, create 4 queries that summarize the following:

|  |  |  |  |
| --- | --- | --- | --- |
| Actual Result | Predicted Result | Name of Query |  |
| Fail | 0.0 = Fail | True positive | Predicted fail and they failed |
| Pass | 0.0 = Fail | False positive | Predicted fail but they passed |
| Fail | 1.0 = Pass | True negative | Predicted pass but they failed |
| Pass | 1.0 = Pass | False negative | Predicted pass and they passed |

In the name of the query, True means that our model predicted a no pass. Positive means that there really was a no pass while negative means there wasn’t’. So in the above query, the prediction model would be incorrect for the False positive and True negative cases.

The -q turns off Jupytor’s automatic visualization and -o saves the output to the local machine allowing us to use the *%%local* magic.

In an empty cell, paste the following code example and press SHIFT + ENTER.

%%sql -q -o true\_positive

SELECT count(\*) AS cnt FROM Predictions WHERE prediction = 0 AND results = 'Fail'

%%sql -q -o false\_positive

SELECT count(\*) AS cnt FROM Predictions WHERE prediction = 0 AND (results = 'Pass' OR results = 'Pass w/ Conditions')

%%sql -q -o true\_negative

SELECT count(\*) AS cnt FROM Predictions WHERE prediction = 1 AND results = 'Fail'

%%sql -q -o false\_negative

SELECT count(\*) AS cnt FROM Predictions WHERE prediction = 1 AND (results = 'Pass' OR results = 'Pass w/ Conditions')

Use Matplotlib to generate some plots for better visualization.

In an empty cell, paste the following code example and press SHIFT + ENTER.

%%local

%matplotlib inline

import matplotlib.pyplot as plt

labels = ['True positive', 'False positive', 'True negative', 'False negative']

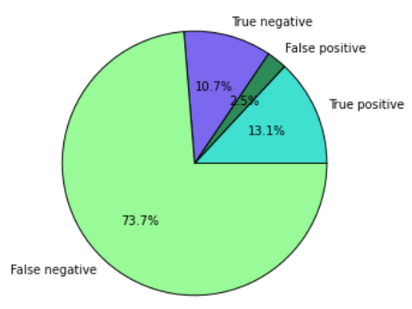
sizes = [true\_positive['cnt'], false\_positive['cnt'], true\_negative['cnt'], false\_negative['cnt'],]

colors = ['turquoise', 'seagreen', 'mediumslateblue', 'palegreen', 'coral']

plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=colors)

plt.axis('equal')

You should see the following output:



1. Shutdown the Jupyter notebook.

After trying the various visualization types, shut down the notebook to release any resources. Click the **File** menu on the notebook and **Close and Halt.**

1. Delete the Spark Cluster

HDInsight clusters are billed on a per minute basis. In order to avoid costly charges, delete your cluster after you have finished using it.

## Summary

In this hands-on lab, you learned how to:

* Create a predictive analysis application that uses logistic regression for binary classification.
* Modify input data into feature vector form so that logistic regression can compute on it
* Parse natural language input data into a feature vector form
* Setup test data and run the new model against the test data