Machine Learning using Spark:

Module 5, Lesson 5  
Build a Clustering Application Hands-On Lab

## Overview

In this lab, students will learn how to build a KMeans Clustering algorithm on a Spark cluster in HDInsight. Students will use the Jupyter notebook to build and test the application. The application uses Gutenberg data at the example of HDInisghts.

## Objectives

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

* Read the input data
* Create functions to reformat the input data into a form that can be used by MLlib
* Build pipeline for number of natural language processing
* Run and see how the KMeans model processes the input data

## Prerequisites

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

* A web browser
* A Spark cluster in Azure HDInsight ([Module 5 Lesson 2 Lab](https://github.com/MSFTImagine/computerscience/tree/master/Complimentary%20Course%20Content/Module5/Labs))

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 KMeans analysis application

## Exercise 1: Write a KMeans Application

You will use Spark to perform some clustering analysis on Gutenberg 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 /example/data/gutenberg/ of HDFS.

## Background

Clustering is a very common machine learning task. It is the process of reviewing input data and sorting them into groups without any label. For example, take a clustering algorithm that accepts article information as input and groups the article into two categories: sports and entertainment.

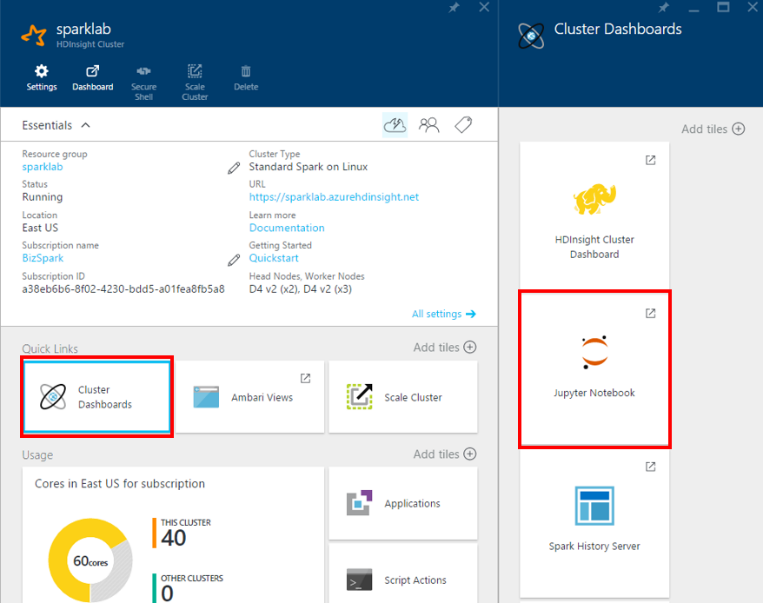
KMeans is an algorithm that can be used for clustering. It is particularly useful for clustering the input data into K groups.

In the steps below, you develop a KMeans model to see how K groups are built.

1. Create Jupyter notebook with Spark kernel

From the Azure Portal, navigate to the Spark cluster that the student just created – the default number of work nodes should be 4. If it is less than 2 nodes, it may not work as it does not have enough resources. . 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 **Spark** 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 Spark kernel a user friendly name, for example, M5\_Lab5\_KMeans.



1. Import required types for this application

The Spark 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.

import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.clustering.KMeans

import org.apache.spark.sql.functions.\_

import sqlContext.implicits.\_

1. Construct Input DataFrame by reading the input data from example data set and cache it to memory.

Input data set is a twitter data in json file format at *HdiSamples/HdiSamples/TwitterTrendsSampleData/tweets.txt*.

The sqlContext can be used to perform transformations on structured data. First, load the sample data in Json format – *tweets.txt* into a Spark SQL dataframe: dataDF.

The data is in Json format. We can use Scala’s json library to parse each line.

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

val dataDF = sqlContext.read.json("wasb:////HdiSamples/HdiSamples/TwitterTrendsSampleData/tweets.txt").cache()

dataDF.show(2) // Show the 2 records/rows in the data frame

dataDF.columns // List the columns file:///G:/Downloads/L5\_Lab9\_KMeans.html

1. Convert the sentence in lower letters to words using tokenizer

The Json file has been converted to a Dataframe. We can convert the input sentence to lower letters and then, each lowered data frame is converted to words.

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

import org.apache.spark.ml.feature.{RegexTokenizer, StopWordsRemover, HashingTF, IDF, Normalizer}

val numClusters = 10

val numFeatures = 2000

val tokenizer = new RegexTokenizer().setInputCol("text").setOutputCol("words").setPattern("\\W+")

val dataWordsDF = tokenizer.transform(dataDF)

dataWordsDF.printSchema

dataWordsDF.select("words").first

You should see output similar to:

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

# THIS IS AN OUTPUT

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

res147: org.apache.spark.sql.Row = [WrappedArray(tresorit, cloud, storage, get, an, extra, 500mb, free, webspace, when, you, sign, up, with, my, http, t, co, nln3x3o7ov, http, t, co, els1ultvg5)]

1. Remove the stopwords using StopWordsRemover

The words has too many stop words so that we have to remove it using *StopWordsRemover*. In an empty cell, paste the following code example and press SHIFT + ENTER.

val remover = new StopWordsRemover().setInputCol("words").setOutputCol("noStopWords")

val noStopWordsListDF = remover.transform(dataWordsDF)

noStopWordsListDF.printSchema

noStopWordsListDF.select("id", "words", "noStopWords").show(5)

You should see output similar to:

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

# THIS IS AN OUTPUT

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

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

| id| words| noStopWords|

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

|537350547683434497|[tresorit, cloud,...|[tresorit, cloud,...|

|537350549692510208|[check, out, my, ...|[check, suite, ju...|

|537350568231337985|[what, s, on, the...|[s, horizon, clou...|

|537350570513014784|[onlynashy, super...|[onlynashy, super...|

|537350572576620544|[smoothhemmo, cc,...|[smoothhemmo, cc,...|

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

only showing top 5 rows

1. Calculate TF (Term-Frequency) and set number of features

The words data needs TF to see how many times it occurs for each twitter with the number of features. In this example, we set the number of features to 2,000. The higher the number of features, the more processing time it needs but the more accurate for clustering.

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

val hashingTF = new HashingTF().setInputCol("noStopWords").setOutputCol("hashingTF").setNumFeatures(numFeatures)

val featurizedDataDF = hashingTF.transform(noStopWordsListDF)

featurizedDataDF.printSchema

featurizedDataDF.select("id", "noStopWords", "hashingTF").show(3)

This should give output similar to the following:

|# -----------------

# THIS IS AN OUTPUT

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

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

| id| noStopWords| hashingTF|

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

|537350547683434497|[tresorit, cloud,...|(2000,[116,157,17...|

|537350549692510208|[check, suite, ju...|(2000,[116,203,39...|

|537350568231337985|[s, horizon, clou...|(2000,[71,115,116...|

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

only showing top 3 rows |

1. Calculate IDF (Inverse Document Frequency) and normalizing it to the even scale

The noStopWords data need IDF to calculate its rank with DF-IDF computation. Thus, IDF will be calculated. Then, IDF needs to be normalized to be evenly scaled to other twitter data.

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

val normalizer = new Normalizer().setInputCol("idf").setOutputCol("features")

val idf = new IDF().setInputCol("hashingTF").setOutputCol("idf")

val idfModel = idf.fit(featurizedDataDF)

1. Set KMeans model with the features and prediction column

We have to build pipeline with Tokenizer, Remover, HashingTF, IDF, Normalizer, KMeans model. Then, We have to train the model with *dataLoweredDF* data above.

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

val kmeans = new KMeans().setFeaturesCol("features").setPredictionCol("prediction").setK(numClusters).setSeed(0)

val pipeline = new Pipeline().setStages(Array(tokenizer, remover, hashingTF, idf, normalizer, kmeans))

// train the model

val model = pipeline.fit(dataDF)

1. Run the test data set

Based on the input data modified, we develop a model for the clustering algorithm. However, we train and test the model with the same data set. Test data set actually should be another data set in order to check out the accuracy of the model.

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

val predictionsDF = model.transform(dataDF)

predictionsDF.groupBy("prediction").count().show(numClusters)

This should give output similar to the following, which shows the total K groups and the number of data set for each group:

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

# THIS IS AN OUTPUT

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

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

|prediction|count|

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

| 0| 7|

| 1| 11|

| 2| 6|

| 3| 15|

| 4| 11|

| 5| 103|

| 6| 40|

| 7| 1|

| 8| 4|

| 9| 2|

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

1. Run SQL with the table in memory

We can create a table and name it "predictionsDF". Then, we can run SQL to see what hashtags exist for group 0, 3, 4, 5, 6.

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

**%%sql**

select entities.hashtags[0].text, entities.hashtags[1].text, entities.hashtags[2].text,

entities.hashtags[3].text, prediction from predictionsDF

where prediction = 0 AND

entities.hashtags[0].text IS NOT NULL AND

entities.hashtags[1].text IS NOT NULL AND

entities.hashtags[2].text IS NOT NULL AND

entities.hashtags[3].text IS NOT NULL

limit 30

This should give output similar to the following, which shows Cloud, Blogs... for group 0:

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

# THIS IS AN OUTPUT

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

Type:Table Pie Line Area Bar

\_c0 \_c1 \_c2 \_c3 prediction

Cloud Blogs Top10 TCN 0

Cloud Blogs Top10 TCN 0

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

**%%sql**

select entities.hashtags[0].text, entities.hashtags[1].text, entities.hashtags[2].text,

entities.hashtags[3].text, prediction from predictionsDF

where prediction = 2 AND

entities.hashtags[0].text IS NOT NULL AND

entities.hashtags[1].text IS NOT NULL AND

entities.hashtags[2].text IS NOT NULL AND

entities.hashtags[3].text IS NOT NULL

limit 30

This should give output similar to the following, which shows more general words such as social, big data, DevOPs....Cloud... for group 2:

Type:

| **\_c0** | **\_c1** | **\_c2** | **\_c3** | **prediction** |
| --- | --- | --- | --- | --- |
| Cloud | BigData | DevOps | IoT | 2 |
| Cloud | BigData | DevOps | IoT | 2 |
| Cloud | BigData | DevOps | IoT | 2 |
| Cloud | BigData | DevOps | IoT | 2 |
| Cloud | BigData | DevOps | IoT | 2 |
| Cloud | BigData | DevOps | IoT | 2 |
| Cloud | BigData | DevOps | IoT | 2 |

You may try predictions 7 and 1, 3, 4, 5, 6, 8, 9 to see what the output looks like.

## Summary

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

* Create a predictive analysis application that uses KMeans for clustering.
* Parse natural language input data in Json into a new text data with the pipeline that is composed of the 5 stages: Tokenizer, Remover, HashingTF, IDF, Normalizer, KMeans model
* Setup train data and run the new model against the test data that is train data