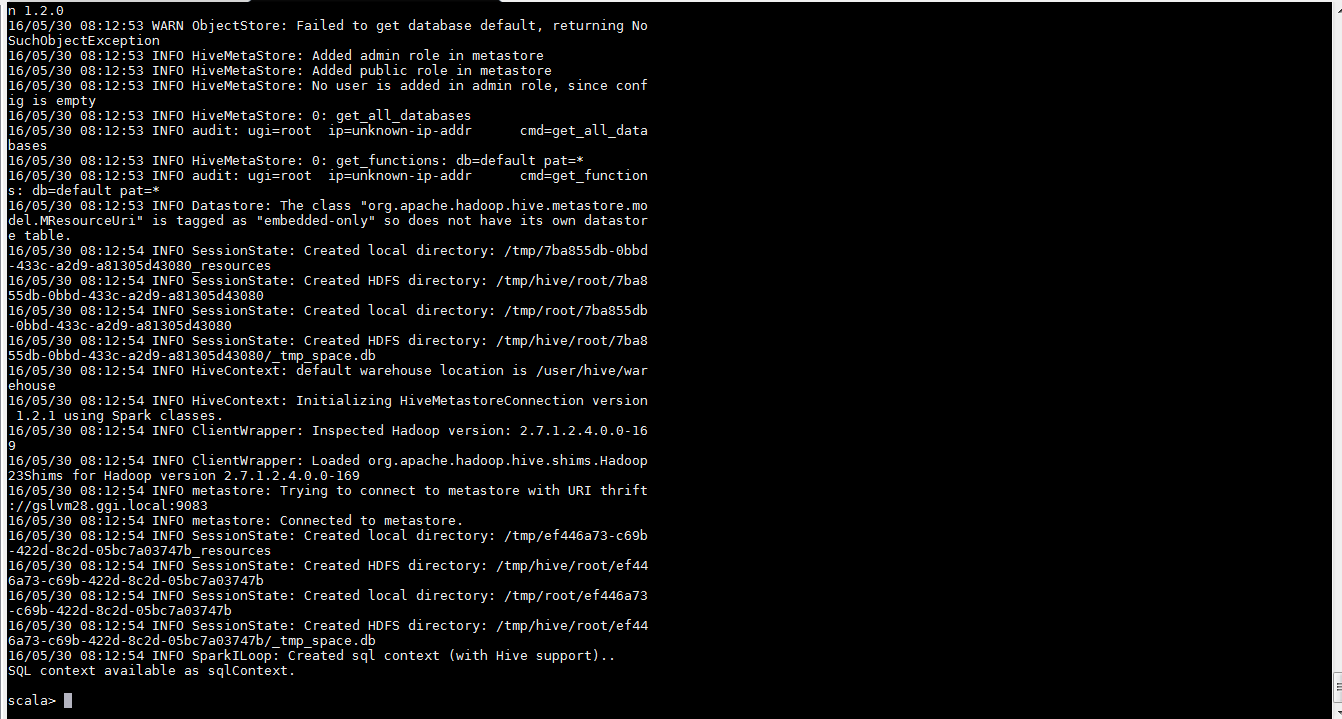
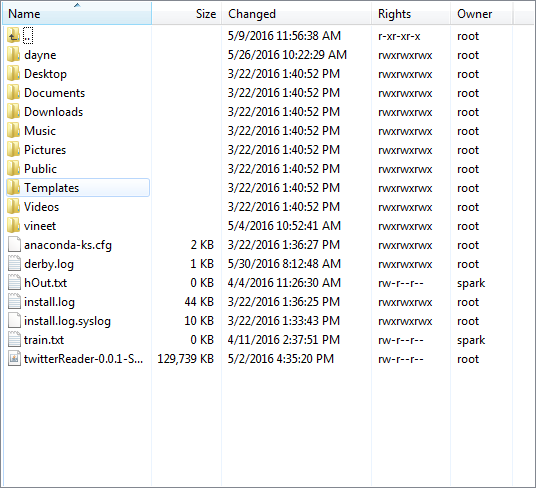
# Spark Tutorial – Mining Twitter Data

1. Connect to spark shell. (open MobaXterm and ssh as root gslvm32) to bash shell
2. Once at bash shell, start spark shell with spark-shell cmd, some config messages will appear just wait
3. We’re going to have to load a databricks package using the –package option so type exit and restart spark-shell with spark-shell --packages com.databricks:spark-csv\_2.11:1.4.0, this will load some jar files from the web for us. Your screen should look like this



1. The next step is moving data onto HDFS so we can manipulate it in Spark. One way to do this is to use winscp



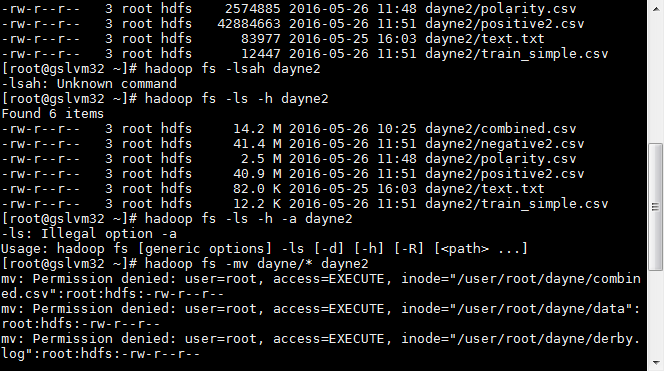
1. Launch winscp from your local machine and connect to gslvm32 (or wherever HDFS is installed)
2. I created a directory called dayne in the gslvm32 bash shell (not the spark-shell) with the command mkdir dayne and chmod 777 dayne to set permissions. I can put data in this folder (csv, text, searchable archive files or anything spark can read) using winscp. This does not transfer data onto Hadoop. First cd into this folder to make it your working directory
3. To get data into the Hadoop file system (HDFS), open bash shell and type the cmd

hadoop fs –mkdir dayne2 to create directory on HDFS

Hadoop fs –ls –h dayne2 (check we’re not overwriting anything)

hadoop fs –mv dayne/file.csv dayne2

We require execute permission on bash to do this and read permission on Hadoop to read file. I have 4 CSV files, including combined.csv (dump of live tweets to predict sentiment), positive2.csv and negative2.csv which contains our labelled training data we’ll use to train our model in Spark and save it to Hadoop so it can be loaded to predict streaming data.



1. Go back to the spark-shell and type

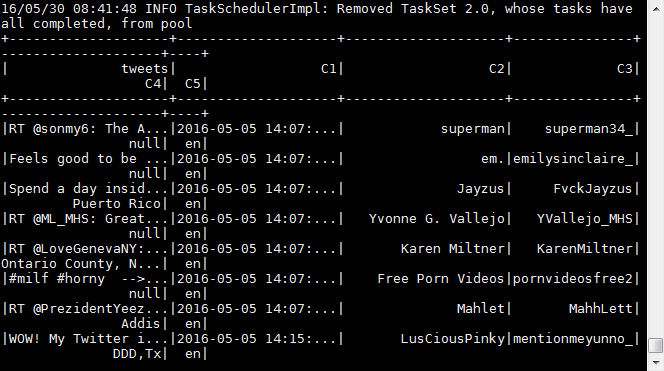
val df= sqlContext.read.format("com.databricks.spark.csv").option("header","false").option("inferSchema","true").load("dayne2/combined.csv").toDF()

This will load our data from combined.csv to a data frame (a distributed table for querying relational data) and even infer the schema from the file for us using our databricks package.

1. If there were no errors, you should see df: org.apache.spark.sql.DataFrame = [C0: string, C1: string, C2: string, C3: string, C4: string, C5: string] on the screen.

Let’s change the column names. There are many ways to do this, we will use a transformation on Data Frames datatype called .withColumnRenamed the syntax is df = df.withColumnRenamed("colName", "newColName").withColumnRenamed("colName2", "newColName2")

1. We should get an error, this is because we declared df to be a val which means we can’t reassign a new value to this name (if we wanted to we should have used var). Call your new Data Frame something else like df2 and don’t forget the val.
2. Use .printSchema to show the renamed column. Note: We can chain together transformations, for example if we wante to change C1 to date we could have done val df2 = df.withColumnRenamed(“C0”,”tweets”).withColumnRenamed(“C1”,”date”). Spark is lazy and doesn’t do any of these computations until we give an action (something that produces a value)
3. Let’s use an action (this lets us do something and always returns a value, unlike a transformation which returns a reference to another RDD). Type df2.printSchema to see the renamed columns.



1. Let’s do the same thing to load our training data, stored in train\_simple.csv.

val training = sqlContext.read.format("com.databricks.spark.csv").option("header","true").option("inferSchema","true").load("dayne2/train\_simple.csv").toDF()

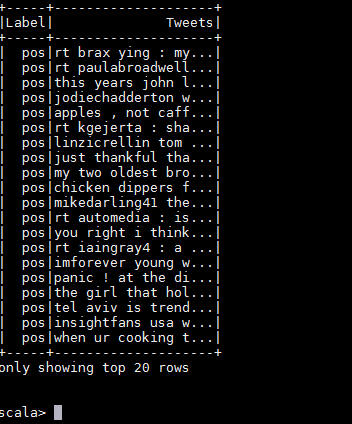
the .toDF() transformation isn’t necessary but makes it clear to anyone reading our code that we want to return a Spark Data Frame and not an RDD (different data structures have different transformations and actions, refer to the documentation located at

<http://spark.apache.org/docs/latest/sql-programming-guide.html>

1. Create a new indexer object val indexer = new StringIndexer().setInputCol("Label").setOutputCol("labels")

val indexed = indexer.fit(training).transform(training).toDF()

indexed.show() // show our updated data frame we named indexed

1. Use the action training.printSchema to display the result.,
2. Our goal is to use spark’s ML library called Mlib (formally MLLib which takes LabelledPoints instead of Data Frames which are slower and requires more boiler plate syntax especially if using PySpark as moving objects from Python to the JVM which Spark handles automatically is expensive, Data Frames help solve this issue).

Here is an overview of the newer spark.ml library which introduces pipelines to make iterative algorithms like machine learning much easier to code.

http://spark.apache.org/docs/latest/ml-guide.html

1. In order to use spark.ml we have to load some internal packages from Spark.

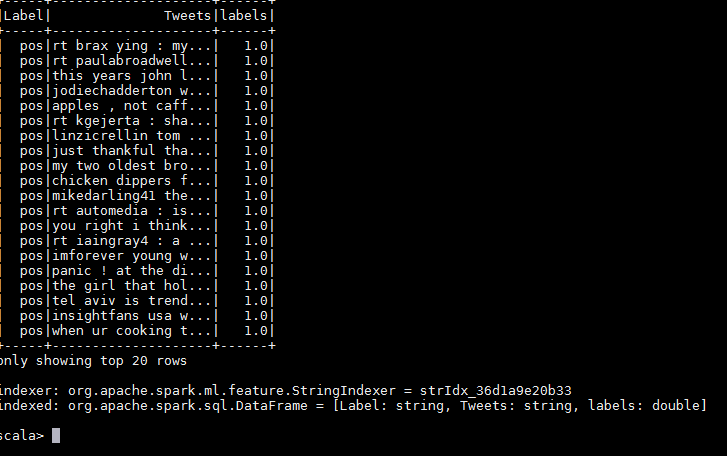
Let’s load the following, note we could have done. We will need the spark.ml, spark.ml.classification, spark.ml.feature and spark.sql.

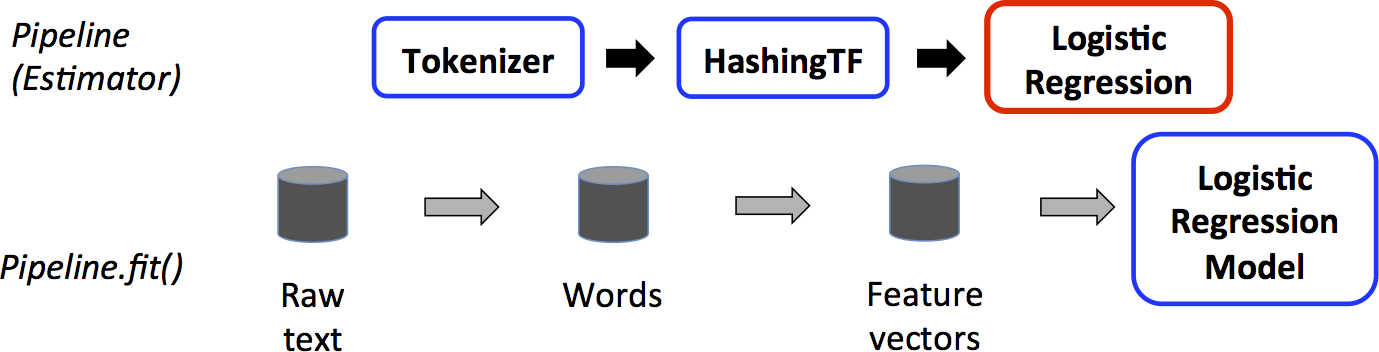
import org.apache.spark.ml.{Pipeline, PipelineModel}import org.apache.spark.ml.classification.LogisticRegression

import org.apache.spark.ml.feature.{HashingTF, Tokenizer}

import org.apache.spark.sql.Row

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

1. Let’s use spark.ml.StringIndexer to create numerical labels for our tweets
2. val indexer = new StringIndexer().setInputCol("Label").setOutputCol("labels")
3. val indexed = indexer.fit(training).transform(training).toDF()
4. indexed.show() //this is our newest Data Frame, we could have also used a Pipeline instead. 



1. Creating a pipeline: Note that the pipeline simplifies the syntax and can work for any DAG (given as an array of transformations)

val indexer = new StringIndexer().setInputCol("Label").setOutputCol("labels")

val tokenizer = new Tokenizer().setInputCol("Tweets").setOutputCol("words")

import org.apache.spark.ml.feature.StopWordsRemover

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

val hashingTF = new HashingTF().setNumFeatures(1000).setInputCol("filtered").setOutputCol("features")

val pipeline = new Pipeline()

.setStages(Array(indexer,tokenizer,remover,hashingTF))

val model = pipeline.fit(training)