Putting it All Together

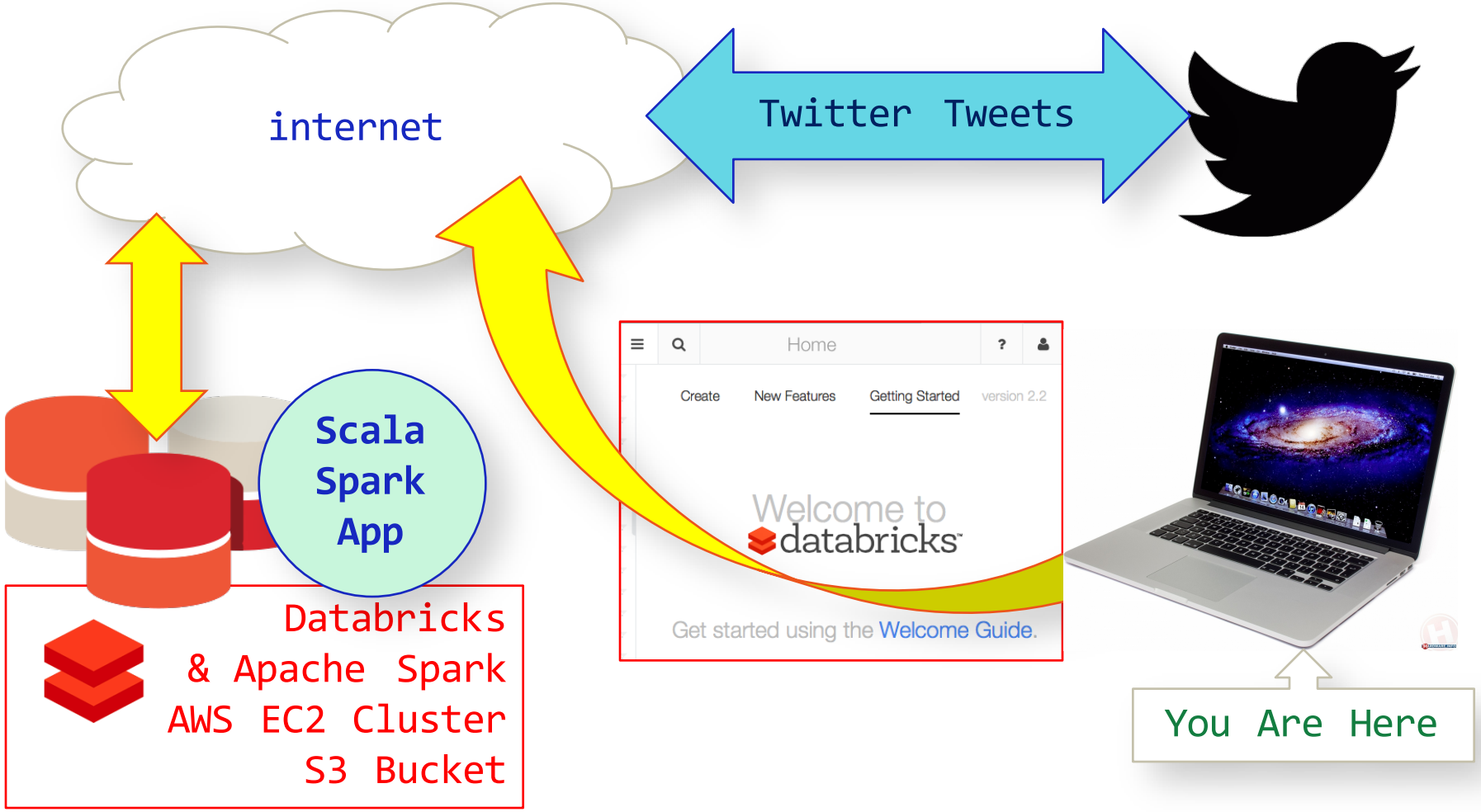
Databricks, Streaming, DataFrames and ML

Databricks Training Team

September 2015

databricks™

Use Case :: Twitter Streaming App



Running in Databricks

- We will walk through a Scala application running in Databricks, and have hands-on labs.
  - Three parts of the app will be created
    1. Streaming only: write tweets to filesystem.
    2. Streaming plus DataFrames: parquet
    3. Add Machine Learning (ML) KMeans
- Scala code examples all compile and run in Databricks attached to a Spark 1.4 cluster (unless otherwise noted).
- The application will leverage:
  - Twitter4j API[1]
  - Spark Streaming API
  - Spark DataFrames API
  - Spark ML KMeans API[2]

Writing Code in Databricks

- When writing code in Databricks (or any Scala REPL) it helps to wrap the code in an object, for scope-control reasons:

```
object Runner extends Serializable {
  /* code */
}
```

  - Databricks becomes the "Driver" of the Spark app.
  - Values and variables can be picked up and sent to the Spark Workers for use by functions.
    - Sometimes inadvertently
    - Sometimes this causes bugs difficult to track in the REPL or UI

Writing Code in Databricks

- Use the Factory methods to create (or get) a **`StreamingContext:`**
  - Best practice and better for recovery:

```
/* Experimental: return the "active" StreamingContext
(that is, started but not stopped) */
StreamingContext.getActiveOrCreate(…)

/* Either recreate a StreamingContext from checkpoint data
or create a new StreamingContext */
StreamingContext.getOrCreate(…)

/*Get the currently active context, if there is one.
Active means started but not stopped. */
StreamingContext.getActive()
```

Creating a Stream in Scala (1 of 3)

```
import org.apache.spark.sql._
import org.apache.spark.streaming._
import org.apache.spark.streaming.twitter.TwitterUtils
import twitter4j.auth._
import twitter4j.conf._
import twitter4j.Status // A tweet

//How long between batches, each batch becomes an RDD
val BatchInterval = 60 // seconds

//Write to cluster later when we have a batch of tweets
val ParquetFile = "dbfs:/mnt/strata-nyc-dev-bootcamp
/tweets.parquet”
```

Creating a Stream in Scala (2 of 3)

```
//Within the object Runner …
def start() = {

    val ssc = //Spark StreamingContext

        StreamingContext.getActiveOrCreate(
            createStreamingContext
        )

    ssc.start()
}
def stop() = {                          //Just stop streaming
    StreamingContext.getActive.map { ssc =>    //lambda
        ssc.stop(stopSparkContext=false)
    }
}
```

Creating a Stream in Scala (3 of 3)

```
//Our implementation of scala.Function0<StreamingContext>
private def createStreamingContext(): StreamingContext =
{
   val ssc = new StreamingContext(sc,
    Seconds(BatchInterval))

    /*TwitterUtils needs StreamingContext and auth details.
    * @transient do not serialze vals no one else needs them. */

    @transient val tconf =
        TwitterCredentialsFactory.getCredentials()
    @transient val auth = new OAuthAuthorization(tconf)

    //Here tweets is a JavaDStream[Status]
    val tweets = TwitterUtils.createStream(ssc, Some(auth))

    /* much more code */

    ssc // we promised to return the StreamingContext
}
```

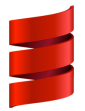
Filter the Stream

- Now we have too many tweets coming over to us.
- Filter them down to only interesting Spark tweets:

```
//What we want to see tweets about:
val contains = Array("#spark", "#apachespark", "#strata",
"#databricks", "#hadoop", "#bigdata", " spark ", " strata ",
"databricks", "big data", "hadoop")

//Status is a Decorator around the tweet text itself:
val sparkTweets = tweets.filter { status =>     //lambda
    contains.exists { word =>
        status.getText.toLowerCase.contains(word)
    }
}
```

Each Batch Becomes an RDD

```
/* Use foreachRDD to process the Tweet Stream... there will
be one RDD created by SparkStreaming per batch... */

sparkTweets.foreachRDD { rdd =>

    // What shall we do with each tweet/status?
    if (! rdd.isEmpty) {      //only if tweets present...

    //Let's write out a file and see what we found
    rdd.map( status =>
        status.getText()).coalesce(1)//One part-00000 file
          .saveAsTextFile(
            "dbfs:/mnt/strata-nyc-dev-bootcamp/twitter_" +
               new Date().getTime()) //unique & easy to read
    }
}
```



Hands On :: Twitter Config

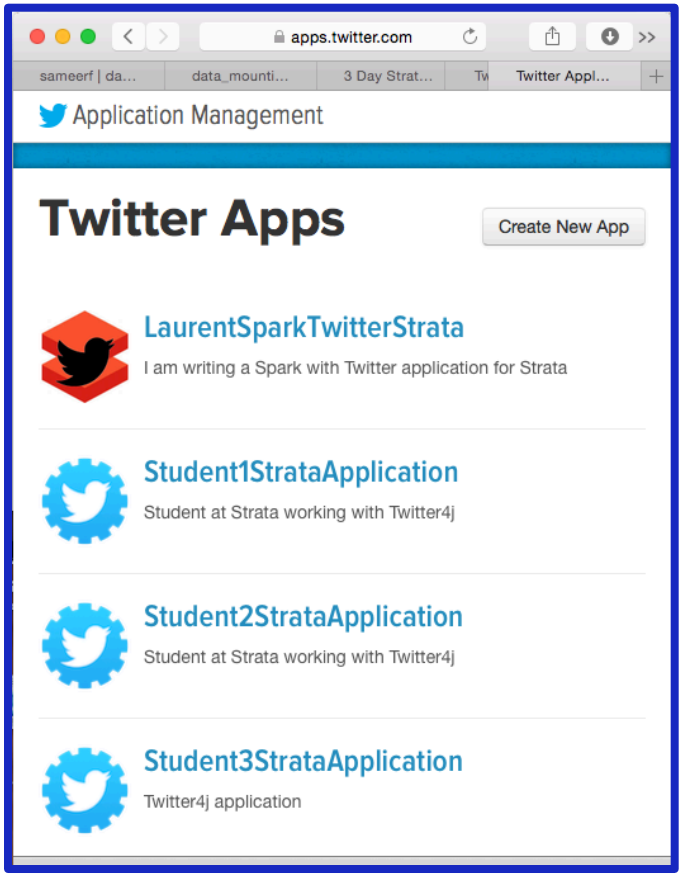
Our first Twitter lab exercise is:
Twitter_Streaming_Lab01_Scala

You will need application credentials for this lab.

If you have a Twitter account:
https://apps.twitter.com

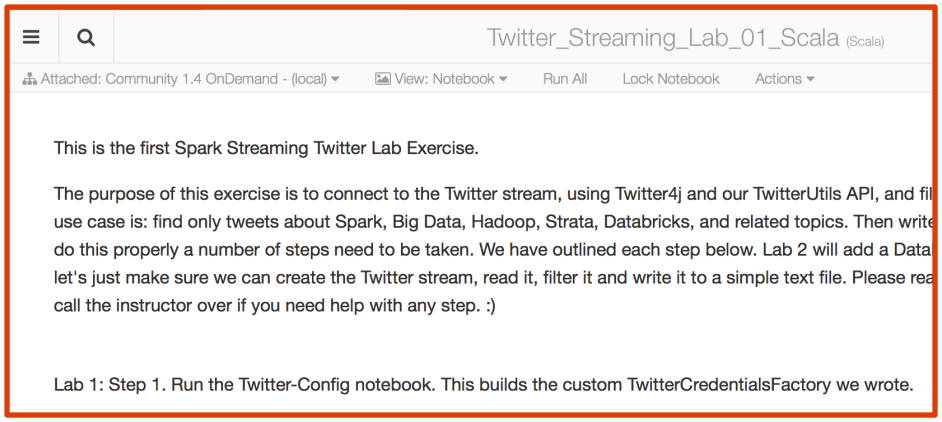
Or use our custom
`TwitterCredentialsFactory`
in the Scala lab code.

Your decision.



Hands On

Switch back to your hands on notebook, and look at the section entitled **Twitter_Streaming_Lab01_Scala**.



Do the first lab exercise there, which will write to the attached cluster's filesystem.

Lab 1: See Results (1 of 3)

```
display(
    dbutils.fs.ls("dbfs:/mnt/strata-nyc-dev-bootcamp")
)
```

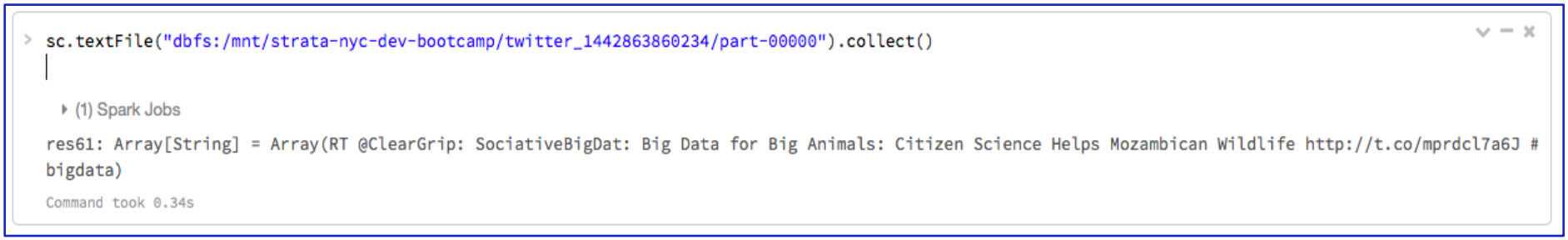
dbfs:/mnt/strata-nyc-dev-bootcamp/twitter_1442861640052/	twitter_1442861640052/	0
dbfs:/mnt/strata-nyc-dev-bootcamp/twitter_1442861700108/	twitter_1442861700108/	0

```
display(dbutils.fs.ls(
    "dbfs:/mnt/strata-nyc-dev-bootcamp/
    twitter_1442861640052")
)
```

path	name	size
dbfs:/mnt/strata-nyc-dev-bootcamp/twitter_1442861640052/_SUCCESS	_SUCCESS	0
dbfs:/mnt/strata-nyc-dev-bootcamp/twitter_1442861640052/part-00000	part-00000	64

Lab 1: Display Results (2 of 3)

```
sc.textFile("dbfs:/mnt/strata-nyc-dev-
bootcamp/twitter_1442863860234/part-00000").collect()
```




```
//OR TRY THIS:
println (
    dbutils.fs.head(
        "dbfs:/mnt/training/twitter/
        twitter_1442863860234/part-00000", 140)
)
```

Lab 1: Display Results (3 of 3)

Adding DataFrames (1 of 2)

```
 //Need this for DataFrame creation
case class Tweet(text: String)
```

```
/* Inside our foreachRDD call... our Function needs this,
 * but the SQLContext from Driver is not Serializable */
val sqlContext =
SQLContext.getOrCreate(SparkContext.getOrCreate())

val sparkTweetsRDD = rdd.map( status =>    //lambda
    Tweet(status.getText())   //need the tweet text only
)

/* The toDF method can drag references via the implicits in
Scala that are not serializable. */
val tweetsDF = sqlContext.createDataFrame(sparkTweetsRDD)
```

Adding DataFrames (2 of 2)

```
 //Need this for Parquet file creation
val ParquetFile = "dbfs:/mnt/strata-nyc-dev-
bootcamp/tweets.parquet"
```


```
/* We use a DataFrame here to make it easy to write out to
a parquet file, for ML K-Means clustering later...
This also disconnects our Streaming work from our ML work.
*/
tweetsDF.coalesce(1) // Make sure there is one partition
 .write.mode(SaveMode.Append) //Grow the file
 .parquet(ParquetFile)
```

Hands On

Switch back to your hands on notebook, and look at the section entitled **Twitter_Streaming_Lab01_Scala**.

Do the second lab exercise there, which will write a parquet file to the filesystem and read it back.

Lab 2: See Results (1 of 2)

```
display(dbutils.fs.ls(ParquetFile))
```

```
> display(dbutils.fs.ls(ParquetFile))
```

▸ (2) Spark Jobs

path	name	size
dbfs:/mnt/strata-nyc-dev-bootcamp/tweets.parquet/_SUCCESS	_SUCCESS	0
dbfs:/mnt/strata-nyc-dev-bootcamp/tweets.parquet/_common_metadata	_common_metadata	259
dbfs:/mnt/strata-nyc-dev-bootcamp/tweets.parquet/_metadata	_metadata	1874
dbfs:/mnt/strata-nyc-dev-bootcamp/tweets.parquet/part-r-00000-04c83268-e322-4dd1-b12e-d7bfed4ff4db.gz.parquet	part-r-00000-04c83268-e322-4dd1-b12e-d7bfed4ff4db.gz.parquet	259
dbfs:/mnt/strata-nyc-dev-bootcamp/tweets.parquet/part-r-00000-896731e1-d613-476f-a3f4-3c71bc8d56b2.gz.parquet	part-r-00000-896731e1-d613-476f-a3f4-3c71bc8d56b2.gz.parquet	259
dbfs:/mnt/strata-nyc-dev-bootcamp/tweets.parquet/part-r-00000-ce647dc8-2cfc-4640-ba06-72769a0f1d3d.gz.parquet	part-r-00000-ce647dc8-2cfc-4640-ba06-72769a0f1d3d.gz.parquet	259
dbfs:/mnt/strata-nyc-dev-bootcamp/tweets.parquet/part-r-00001-04c83268-e322-4dd1-b12e-d7bfed4ff4db.gz.parquet	part-r-00001-04c83268-e322-4dd1-b12e-d7bfed4ff4db.gz.parquet	259

```
val dfTweets = sqlContext.read.parquet(ParquetFile)
display(dfTweets.take(10)) // takes a few seconds
```

```
> val dfTweets = sqlContext.read.parquet(ParquetFile)
  display(dfTweets.take(10))
```

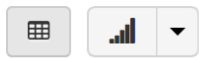
▸ (2) Spark Jobs Cancel

Lab 2: See Results (2 of 2)

```
> val dfTweets = sqlContext.read.parquet(ParquetFile)
display(dfTweets.take(10))
```

▸ (5) Spark Jobs

col_0
Text= RT BMCSoftware ".EMA_Research says TrueSight Intelligence is bringing versatility & ease of use to big data analyt… http://t.co/zAizvCdnhm"
Text= RT @ClearGrip: METAMORF_US: #Analyticship:#BusinessIntelligence, #BigData, #Analytics-Datagres & DataStax dancing with Storage fo… http://t…
Text= #bigdata #SaaS 10 Real #DataScientist Interview Questions http://t.co/UE0n0FCXyy http://t.co/aRtsWwmX4r
Text= RT @smeyer831: BMC is bringing "versatility & ease of use to Big Data Analytics" according to @ema_research http://t.co/DxjZAbKmpX #DevOps …



Big Data

▸ (2) Spark Jobs

col_0
Text= RT @smeyer831: BMC is bringing "versatility & ease of use to Big Data Analytics" according to @ema_research http://t.co/DxjZAbKmpX #DevOps …

Big Data

```
> dfTweets.count

  ▸ (5) Spark Jobs

res76: Long = 8
```

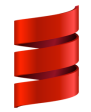
Adding Machine Learning `KMeans`

- KMeans[1] is a clustering ML algorithm.
- We will perform this next for our tweet data.

1. Get ML "training" data (legacy): old Tweets.[2]
2. Featurize the data, turn text into Features.
3. Train the KMeans model on the featurized text.
4. Build the predictions (what categories) the tweets fall into: We chose four categories.
5. Display this using a visualization: pie chart.
6. Show some examples.

Machine Learning (ML) preparation

```
import org.apache.spark.ml._
import org.apache.spark.ml.feature.{HashingTF, Tokenizer,
IDF}
import org.apache.spark.mllib.linalg.Vector
import org.apache.spark.sql.Row

//Case class needed for DataFrame
case class Tweet(text: String)

//Raw legacy tweets as a text file for training data
val rawTweets = sc.textFile("/mnt/strata-nyc-dev-bootcamp/
tweets.txt").filter(text => (!text.equals("")))
) // removed blank lines...

//This is the DataFrame[Tweet] representation
val training = rawTweets.map( text => Tweet(text)
).toDF("text")
```



Sample the Training Data

- Example tweets from training data (**`tweets.txt`**):

Attached: Community 1.4 Spot - (local) | View: Notebook | Run All | Lock Notebook | Actions

```
> display(training)
```

▸ (2) Spark Jobs

text
Text= RT @jose_garde: Hospitals use #bigdata platform to improve care, benchmark quality and manage workflow - http://t.co/412uwH0sqX
Text= Hot off the press: Julius Akinyemi: Africa needs to harness big data to really innovate http://t.co/rlXXx26WtH #BigData
Text= Searching big data faster http://t.co/BuR7ei3FPD http://t.co/3lvEU80HRf , johnny heath corpus christi
Text= #ISAO collaboration with #DHS , ICIT and Federal #IT Leaders. #threat #bigdata #leadership http://t.co/kcQZbauZoZ
Text= Wacom Bamboo Spark puts your notes on paper on your smartphone http://t.co/rgnclrEdOX
Text= RT @EurekaMag: Spark generating properties of electrode gels used during defibrillation a potential fire hazard http://t.co/LjuMsVKi4I
Text= RT @Debrahliani: No matter how buff someone is, if there's no spark just don't force it Lool
Text= “Real Time Analytics With Spark Streaming and Cassandra” https://t.co/ST5ARPVTMi

```
> training.count()
```

▸ (1) Spark Jobs

```
res78: Long = 201
```

ML Tokenizer

```
/* Configure an ML pipeline, which will consists of four
stages:  * TOKENIZER, HASHING TF, IDF, and then K-MEANS.

A tokenizer that converts the input string, in our case the
Tweet, to lowercase and then splits it by white spaces. */

val tokenizer = new Tokenizer()
    .setInputCol("text")   // DataFrame column
    .setOutputCol("words") // Tokenized text
```

ML Tokenizer

```
> val tokenOutput = tokenizer.transform(training)
  display(tokenOutput)
```

▸ (2) Spark Jobs

text	words
Text= RT @jose_garde: Hospitals use #bigdata platform to improve care, benchmark quality and manage workflow - http://t.co/412uwH0sqX	▸ ["text=","rt","@jose_garde:","hospitals","use","#bigdata","platform","to","improve","care,","bench
Text= Hot off the press: Julius Akinyemi: Africa needs to harness big data to really innovate http://t.co/rlXXx26W	▸ ["text=","hot","off","the","press:","julius","akinyemi:","africa","needs","to","harness","big","data",

Command took 0.47s

Before

After Tokenization

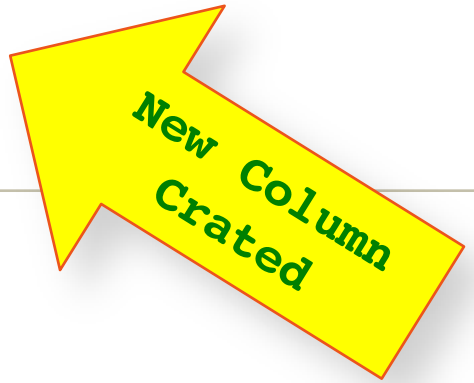
TF-IDF

- Term frequency-inverse document frequency (TF-IDF) is a feature vectorization method widely used in text mining to reflect the importance of a term to a document in the corpus.
  - We will use this technique to help us with Kmeans clustering
  - In MLlib, we separate TF and IDF to make them flexible.[1]

- Term frequency (TF) is the number of times that term appears in document, while document frequency is the number of documents that contains term.
  - If we only use term frequency to measure the importance, it is very easy to over-emphasize terms that appear very often but carry little information about the document, e.g., “a”, “the”, and “of”.
  - If a term appears very often across the corpus, it means it doesn’t carry special information about a particular document.

Spark ML :: TF-IDF

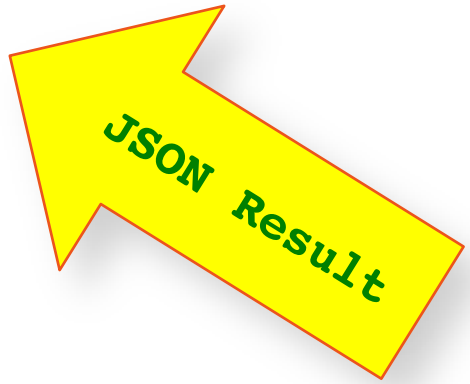
```
/* HashingTF Maps a sequence of terms to their term
frequencies using the hashing trick.[1] */

val hashingTF = new HashingTF()
 .setNumFeatures(1000)
 .setInputCol(tokenizer.getOutputCol) // Link the pipeline
 .setOutputCol("features")             /* Sparse Vector */
```



HashingTF "features" Column

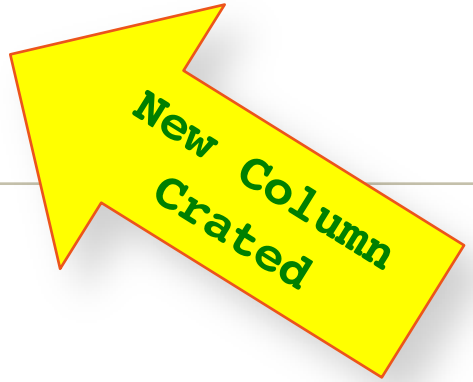
```
val hashingOutput = hashingTF.transform(tokenOutput)
display(hashingOutput)
```

features
{"type":0,"size":1000,"indices":[19,35,45,103,105,135,221,304,408,631,650,705,707,727,773,858],"values":[1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1]}
{"type":0,"size":1000,"indices":[10,39,104,205,242,304,477,501,523,536,631,643,707,738,742,801,935],"values":[1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1]}



Spark ML :: IDF

```
/* Compute the Inverse Document Frequency (IDF) given a
collection of documents.*/

val inverseDocumentFreq = new IDF()
.setInputCol(hashingTF.getOutputCol) // Link the pipeline
.setOutputCol("frequency”)
```



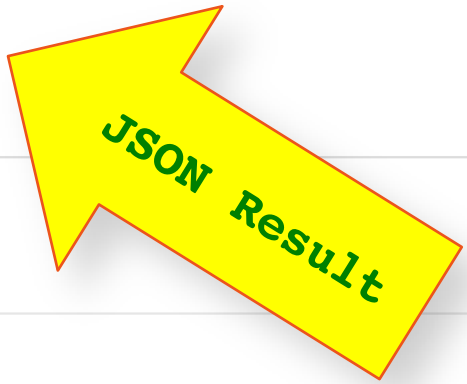
IDF "frequency" Column

```
> //Test to make sure it worked. This IDF is an "Estimator"" rather than a transformer.
  val idfOutput = inverseDocumentFreq.fit(hashingOutput).transform(hashingOutput)
  display(idfOutput)
```

▸ (3) Spark Jobs

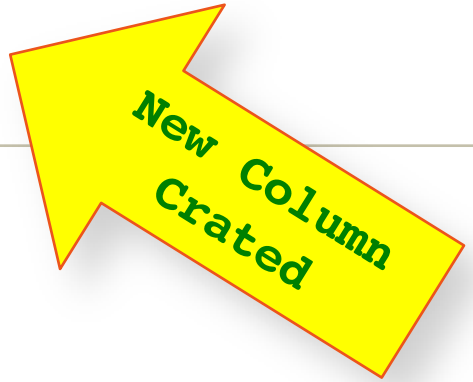
frequency
▸ {"type":0,"size":1000,"indices":[19,35,45,103,105,135,221,304,408,631,650,705,707,727,773,858],"values":[3.516508228173
▸ {"type":0,"size":1000,"indices":[10,39,104,205,242,304,477,501,523,536,631,643,707,738,742,801,935],"values":[1.26521642

Command took 0.59s



ML Normalizer

```
/* L2 (value of each element "L" squared) normalization
of the IDF output is needed. */

val normalizer = new Normalizer()
.setInputCol(inverseDocumentFreq.getOutputCol) //Link
.setOutputCol("norm")
```

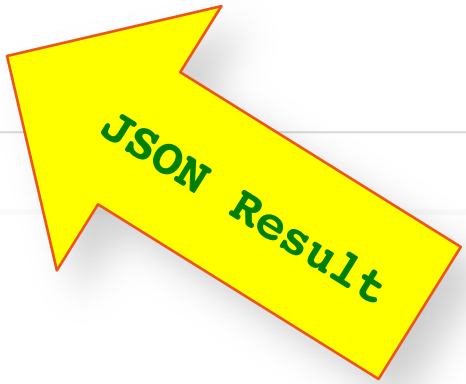


IDF "norm" Column

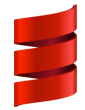
```
> val normOutput = normalizer.transform(idfOutput)
  display(normOutput)
```

▸ (2) Spark Jobs

norm
▸ {"type":0,"size":1000,"indices":[19,35,45,103,105,135,221,304,408,631,650,705,707,727,773,858],"values":[0.25721233798503
▸ {"type":0,"size":1000,"indices":[10,39,104,205,242,304,477,501,523,536,631,643,707,738,742,801,935],"values":[0.093093915

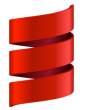


ML KMeans v 1.4 (1 of 3)



```
/* K-means clustering with support for multiple parallel
runs… This is an iterative algorithm that will make
multiple passes over the data, so any RDDs given to it
should be cached by the user. imports not shown */

val km = new KMeans()
.setK(4) //how many clusters (centroids) to group
.setSeed(43L) // Random number generator reproducability

val pipeline = new Pipeline()
 .setStages(Array(tokenizer, hashingTF,
   inverseDocumentFreq, normalizer)) //The pipeline so far

val pipelineModel = pipeline.fit(training)

val pipelineResult = pipelineModel.transform(training)
```

Kmeans Prediction (2 of 3)



```
val kmInput = pipelineResult.select("norm")
.rdd.map(_(0).asInstanceOf[Vector]) //sparse Vectors
.cache() //iterative algorithm

val kmModel = km.run(kmInput)

// training data predictions only
val kmPredictions = kmModel.predict(kmInput)

val sparkTweetsDF = // new fresh tweets
sqlContext.read.parquet("dbfs:/mnt/strata-nyc-dev-bootcamp/
tweets.parquet")

val newTweetsTransformed =
pipelineModel.transform(sparkTweetsDF)
```

Kmeans Prediction (3 of 3)

```
val kmNewTweetsInput =
newTweetsTransformed.select("norm").rdd
 .map(_(0).asInstanceOf[Vector])

val newTweetText = newTweetsTransformed.select("text")
 .rdd.map(_(0).asInstanceOf[String])

val newTweetsPrediction = kmModel.predict(kmNewTweetsInput)

/* We combine the new tweet text with the new tweet
prediction (group in KMeans) using a zip association */

val textAndPrediction =
newTweetText.zip(newTweetsPrediction)

println(textAndPrediction.collect().mkString("\n"))
```

Text and Prediction

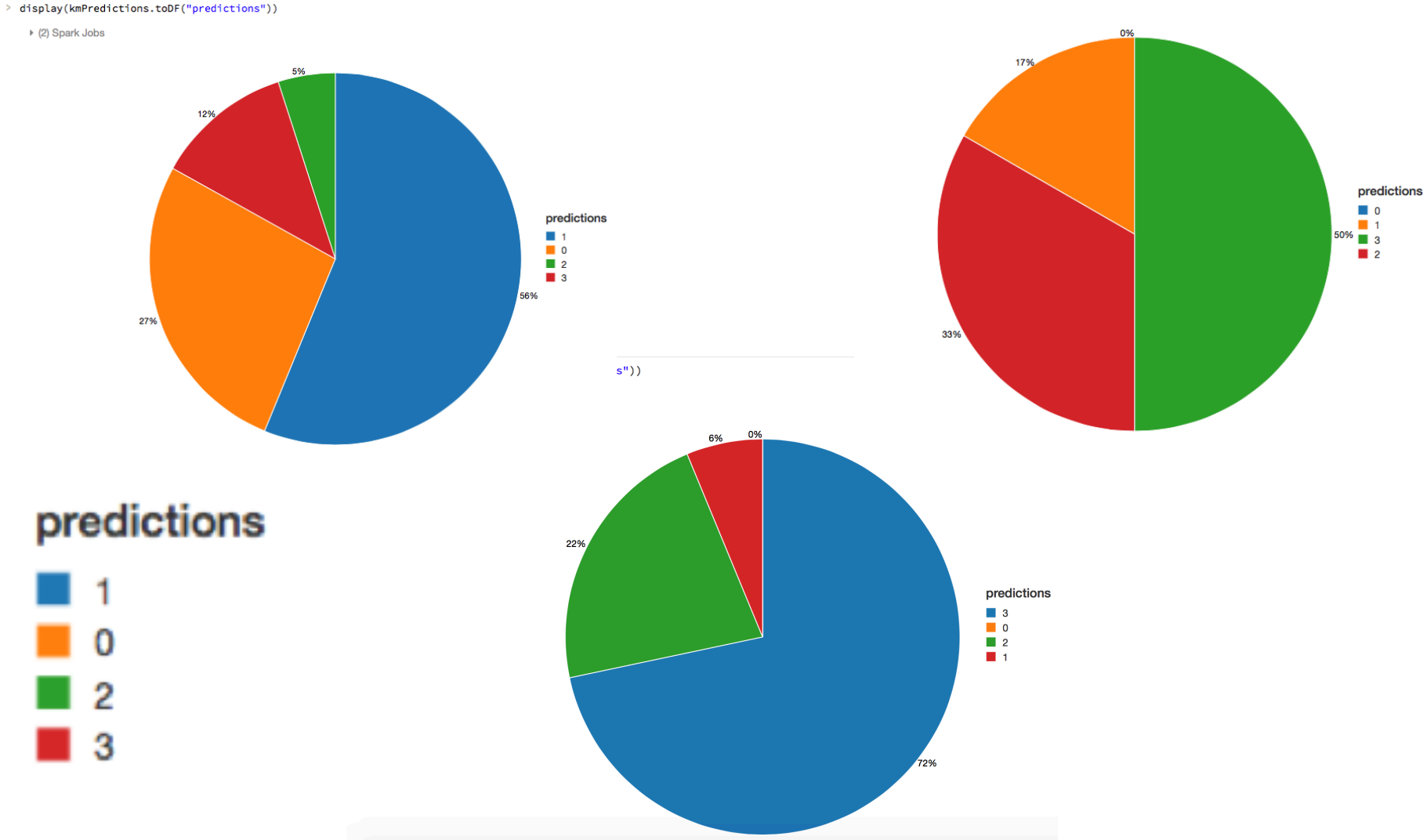
```
// We are taking the new tweet text and matching it with the new tweet prediction (group in KMeans)
val textAndPrediction = newTweetText.zip(newTweetsPrediction)
println(textAndPrediction.collect().mkString("\n"))
```

▸ (11) Spark Jobs Cancel

```
(Text= How to begin with Hadoop http://t.co/vaCZIUcTBc #BigData,0)
(Text= RT @SELFmagazine: These secrets of SELF Made women will help spark big ideas and
(Text= RT BMCSoftware ".EMA_Research says TrueSight Intelligence is bringing versatility
(Text= My favourite bit of #TheBeauxStratagem: they had nasty men at sword point, charac
(Text= RT @craigbrownphd: Accenture aims to make big data analytics immersive, accessibl
(Text= RT @ClearGrip: METAMORF_US: #Analyticship:#BusinessIntelligence, #BigData, #Analy
(Text= RT @SELFmagazine: These secrets of SELF Made women will help spark big ideas and
(Text= RT @craigbrownphd: Accenture aims to make big data analytics immersive, accessibl
(Text= RT @SQLServer: .@MWinkle on how to compose Spark & R, or do scale-out querying on
(Text= #bigdata #SaaS 10 Real #DataScientist Interview Questions http://t.co/UE0n0FCXyy
(Text= RT @smeyer831: BMC is bringing "versatility & ease of use to Big Data Analytics"
```

Display Predictions :: Pie Charts

- `display(kmPredictions.toDF("prediction"))`



Hands On

Switch back to your hands on notebook, and look at the section entitled **Twitter_Streaming_Lab01_Scala**.

Do the third lab exercise there, which will add **`KMeans`** clustering analysis to the **`DataFrame`**.

```
// Questions?

val instructor= Databricks.getInstructor()
    .setInputCol(student.getOutputCol) //Link
    .setOutputCol("answer")
```