Hadoop

Introduction to Spark

- Introduction to MlLib
- Hands-On Approach

Ari Wibisono, M. Kom

Outline

- Spark MLLib
- Hands-on Spam Classification
- Hands-on Traffic Flow Prediction

MLlib

- MLlib is Spark's library of machine learning functions. Designed to run in parallel on clusters, MLlib contains a variety of learning algorithms and is accessible from all of Spark's programming languages.
- MLlib's design and philosophy are simple: it lets you invoke various algorithms on distributed datasets, representing all data as RDDs.
- MLlib introduces a few data types (e.g., labeled points and vectors), but at the end of the day, it is simply a set of functions to call on RDDs.
- For example, to use MLlib for a text classification task (e.g., identifying spammy emails), you might do the following:

MLlib

- MLlib is Spark's library of machine learning functions. Designed to run in parallel on clusters, MLlib contains a variety of learning algorithms and is accessible from all of Spark's programming languages.
- MLlib's design and philosophy are simple: it lets you invoke various algorithms on distributed datasets, representing all data as RDDs.
- MLlib introduces a few data types (e.g., labeled points and vectors), but at the end of the day, it is simply a set of functions to call on RDDs.
- For example, to use MLlib for a text classification task (e.g., identifying spammy emails), you might do the following:

MLlib

1. Start with an RDD of strings representing your messages.
2. Run one of MLlib's feature extraction algorithms to convert text into numerical features (suitable for learning algorithms); this will give back an RDD of vectors.
3. Call a classification algorithm (e.g., logistic regression) on the RDD of vectors; this will give back a model object that can be used to classify new points.
4. Evaluate the model on a test dataset using one of MLlib's evaluation functions.

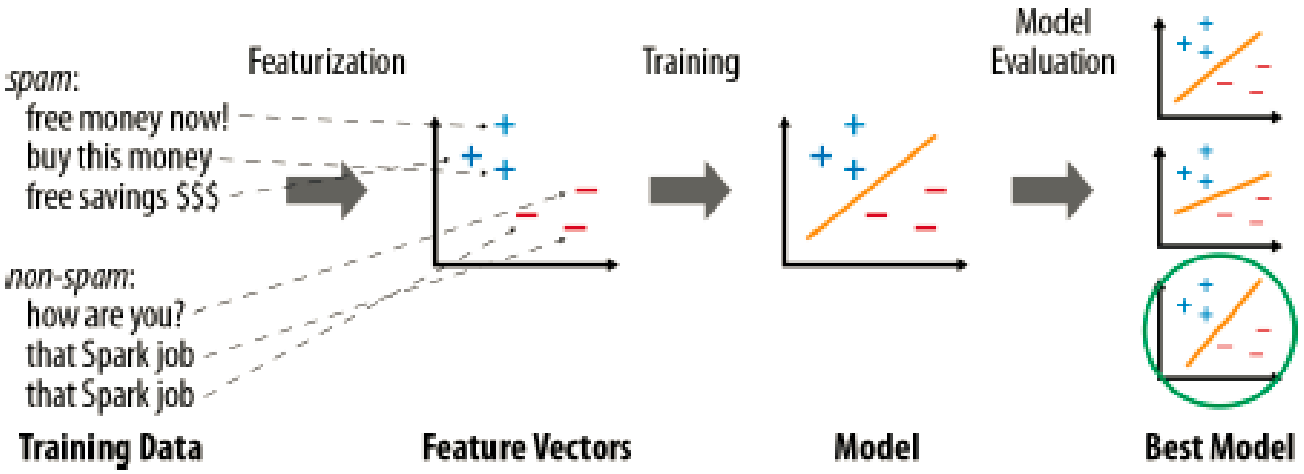
MLlib

1. Start with an RDD of strings representing your messages.
2. Run one of MLlib's feature extraction algorithms to convert text into numerical features (suitable for learning algorithms); this will give back an RDD of vectors.
3. Call a classification algorithm (e.g., logistic regression) on the RDD of vectors; this will give back a model object that can be used to classify new points.
4. Evaluate the model on a test dataset using one of MLlib's evaluation functions.

MLlib

- Machine learning algorithms attempt to make predictions or decisions based on training data, often maximizing a mathematical objective about how the algorithm should behave.
- All learning algorithms require defining a set of features for each item, which will be fed into the learning function.
- Most algorithms are defined only for numerical features (specifically, a vector of numbers representing the value for each feature), so often an important step is feature extraction and transformation to produce these feature vectors.

MLlib



- Most learning algorithms have multiple parameters that can affect results, so real-world pipelines will train multiple versions of a model and evaluate each one.
- To do this, it is common to separate the input data into “training” and “test” sets, and train only on the former, so that the test set can be used to see whether the model overfit the training data.
- MLlib provides several algorithms for model evaluation

Example : Spam Classification

- This program uses two MLlib algorithms: HashingTF, which builds term frequency feature vectors from text data, and Logistic
- RegressionWithSGD, which implements the logistic regression procedure using stochastic gradient descent (SGD).
- We assume that we start with two files, spam.txt and normal.txt, each of which contains examples of spam and non-spam emails, one per line.
- We then turn the text in each file into a feature vector with TF, and train a logistic regression model to separate the two types of messages.

Example : Spam Classification

```
#Initialization
from pyspark import SparkContext
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.classification import LogisticRegressionWithSGD
from pyspark.mllib.feature import HashingTF

if __name__ == "__main__":
    sc = SparkContext(appName="PythonBookExample")

    # Load 2 types of emails from text files: spam and ham (non-spam).
    # Each line has text from one email.
    spam = sc.textFile("http://hdp-00:8020/<your file path>")
    ham = sc.textFile("http://hdp-00:8020/<your file path>")

    # Create a HashingTF instance to map email text to vectors of 100 features.
    tf = HashingTF(numFeatures = 100)
    # Each email is split into words, and each word is mapped to one feature.
    spamFeatures = spam.map(lambda email: tf.transform(email.split(" ")))
    hamFeatures = ham.map(lambda email: tf.transform(email.split(" ")))
```

Example : Spam Classification

```
# Create LabeledPoint datasets for positive (spam) and negative (ham) examples.
    positiveExamples = spamFeatures.map(lambda features: LabeledPoint(1, features))
    negativeExamples = hamFeatures.map(lambda features: LabeledPoint(0, features))
    training_data = positiveExamples.union(negativeExamples)
    training_data.cache() # Cache data since Logistic Regression is an iterative algorithm.

    # Run Logistic Regression using the SGD optimizer.
    # regParam is model regularization, which can make models more robust.
    model = LogisticRegressionWithSGD.train(training_data)

     # Test on a positive example (spam) and a negative one (ham).
    # First apply the same HashingTF feature transformation used on the training data.
    posTestExample = tf.transform("O M G GET cheap stuff by sending money to ...".split(" "))
    negTestExample = tf.transform("Hi Dad, I started studying Spark the other ...".split(" "))

    # Now use the learned model to predict spam/ham for new emails.
    print "Prediction for positive test example: %g" % model.predict(posTestExample)
    print "Prediction for negative test example: %g" % model.predict(negTestExample)

    sc.stop()
```

Traffic Flow Prediction

- We carry out the processing of traffic data during the 5-year in Motorway. Those data are processed and analyzed using FIMT-DD algorithm (a decision tree based algorithm
- The knowledge of traffic condition is developed based on huge training data .
- We have developed the visualization animation based on the knowledge which has been built.

Ari Wibisono, et. al, Traffic big data prediction and visualization using Fast Incremental Model Trees-Drift Detection (FIMT-DD), Knowledge-Based Systems, Volume 93, 1 February 2016,

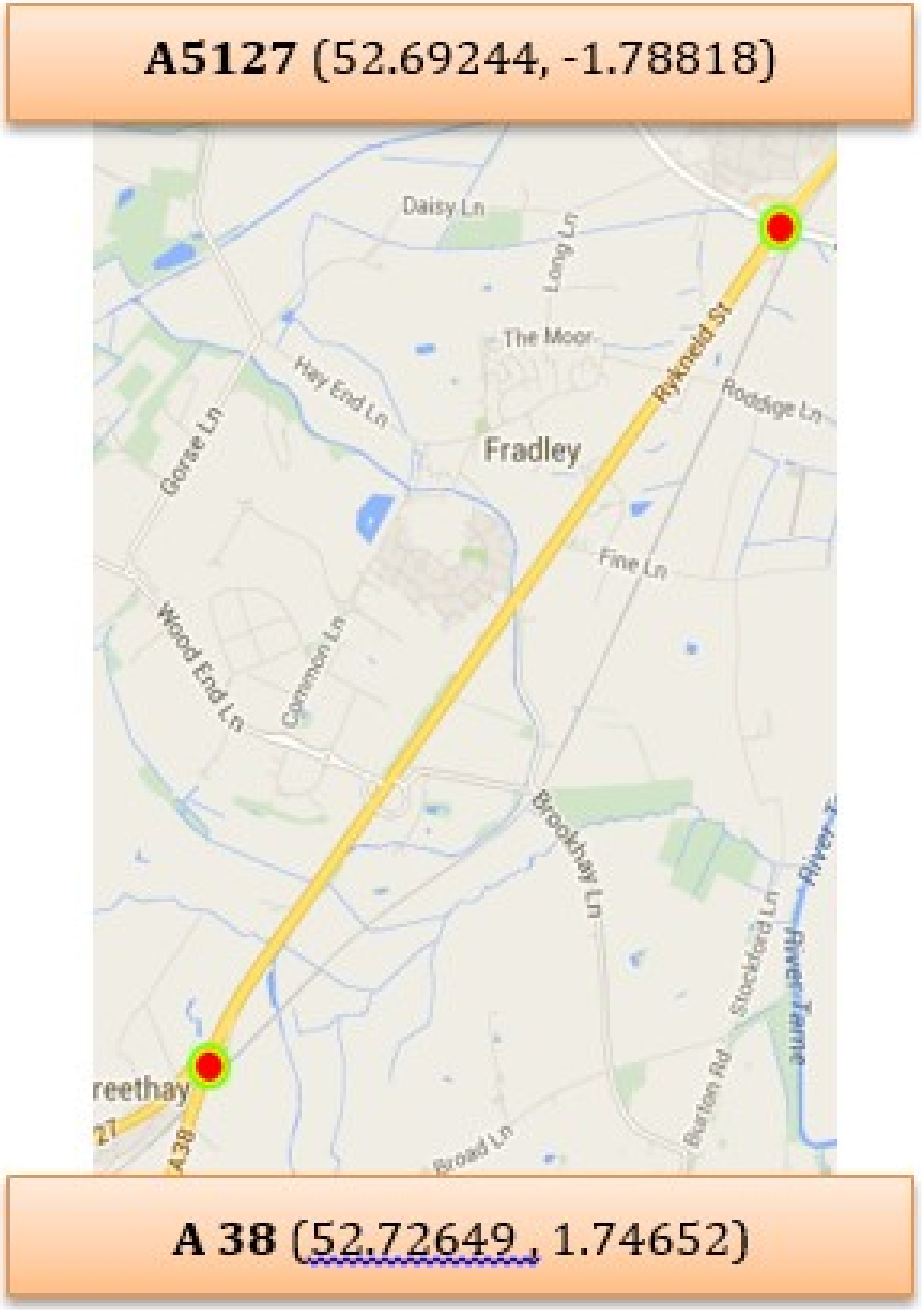
Traffic Big Data Prediction

- This data have been gathered by the UK Highway Agency for one and a half years, from 2008 to 2009. In total, the data contains 100,000,000 instances.
- The data were gathered by combining several sources, such as Automatic Number Plate Recognition cameras, in-vehicle Global Positioning Systems, and several inductive loops sensors built inside the road.
- The data were gathered for 24 hours straight, for every day of the week.
- For a single month, the size of the data is approximately 800 Megabytes.
- That means, in a single year, the data will be 800 x 12 = 9600 Megabytes of data, and for 1,5 years, the data will be 9600 x 1,5 = 14000 Megabytes or 14 Gigabytes

No	Attribute Name	Description of the Attributes
1	LinkRef	This attribute is an alphanumeric link ID that is unique. This attribute represents the link from a junction to junction on the roads that the Highway Agency currently manages.
2	LinkDescription	The description of the link.
3	Date	The date of travel.
4	TimePeriod	Each time period is represented as 15 minute-intervals in this attribute, therefore, there are 96 intervals in the day that the data refers to. The data is represented from 0 to 95, where 0 indicates 00:00 to 00:15.
5	AverageJT	This attribute is the average Journey Time that it takes to travel across each 'LinkRef'. In other words, this is the journey time from one junction to the other, within a given 15-minute time period. This time is represented in seconds.
6	AverageSpeed	The average speed of the vehicle entering a junction to junction link within the 15 minute-intervals. This attribute is represented in kilometers per hour (km/h).
7	DataQuality	An indicator that shows the quality of the the quality of the journey time data for the link and time period. This attribute is represented from 1 to 5, where 1 indicates the highest quality data and 5 the lowest.
8	LinkLength	The length from one junction to junction in kilometers.
9	Flow	The average of the flow of the link, with time period, and day type.

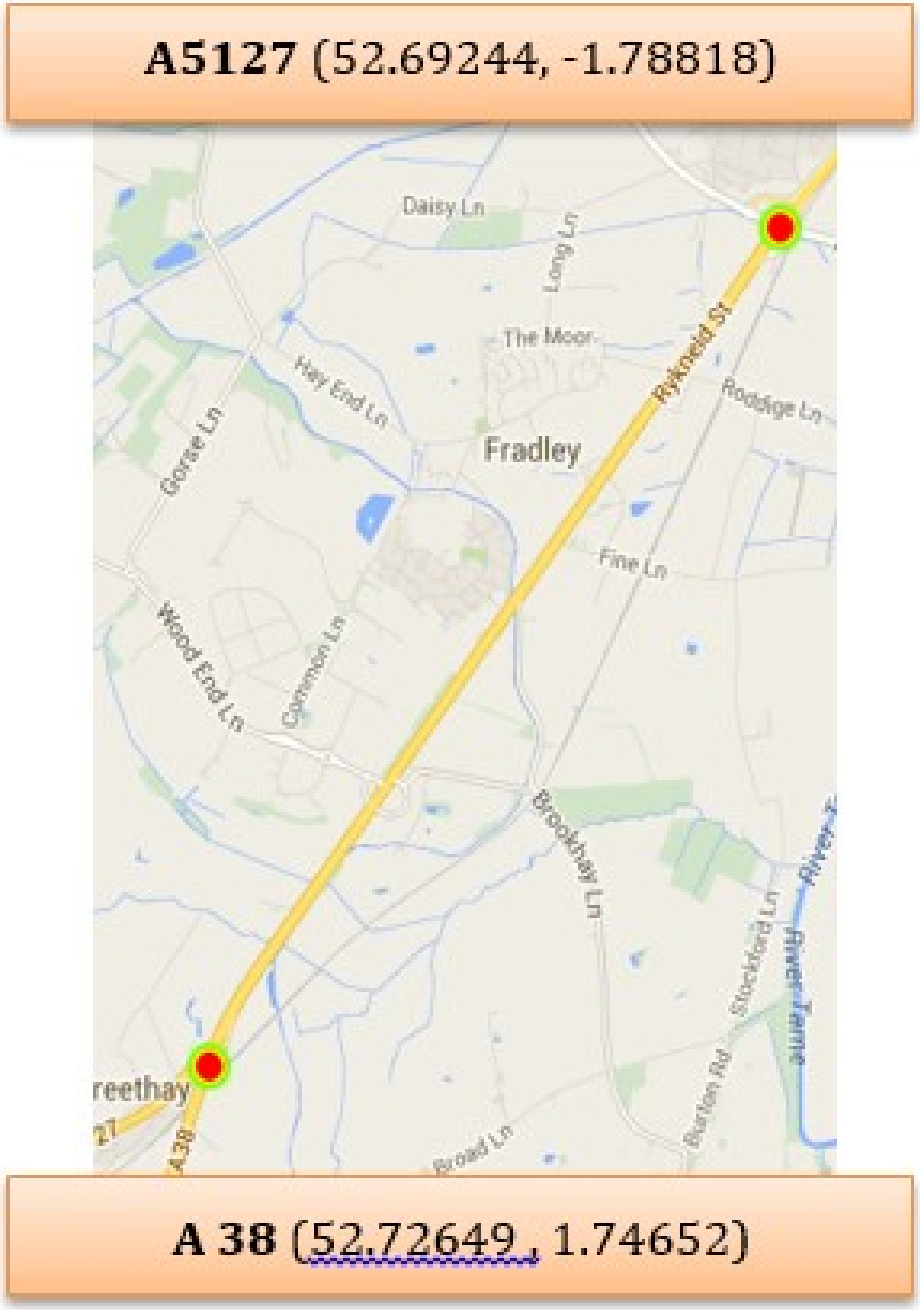
Attribute Extraction

- The unique id represents location of the sensor. We have modified it to represent the latitude and longitude positions of the sensor location.
- For the attributes that contains the date, we have modified the data so that it represents the days in the week. The days are represented in numbers, where 1 is Monday and 7 is Sunday.
- For the specific time data, the time is split from a 24 hours format to a 15 minutes interval format. So, there are 96 different time intervals, where 0 represents the time interval 00:00 to 00:15.
- The metrics used in this dataset is kilometers, so distance is represented as kilometers and the speed is kilometers per hour.



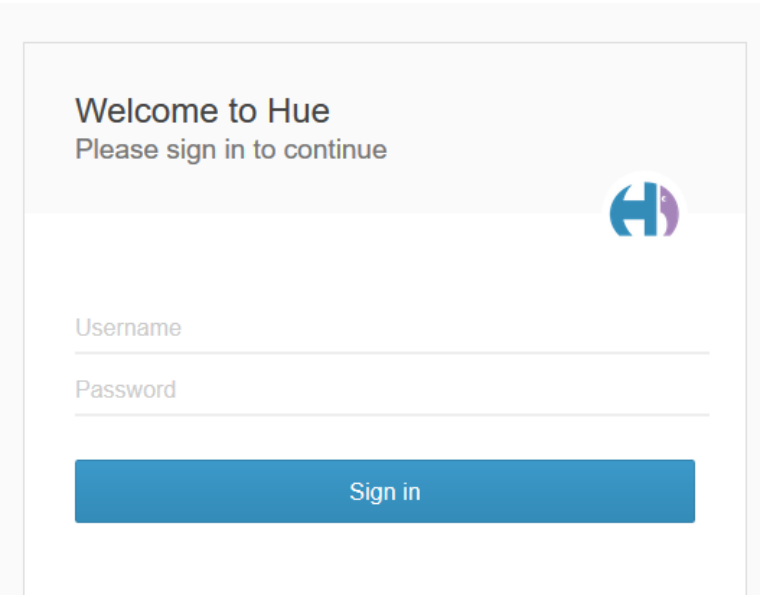
Attribute Extraction

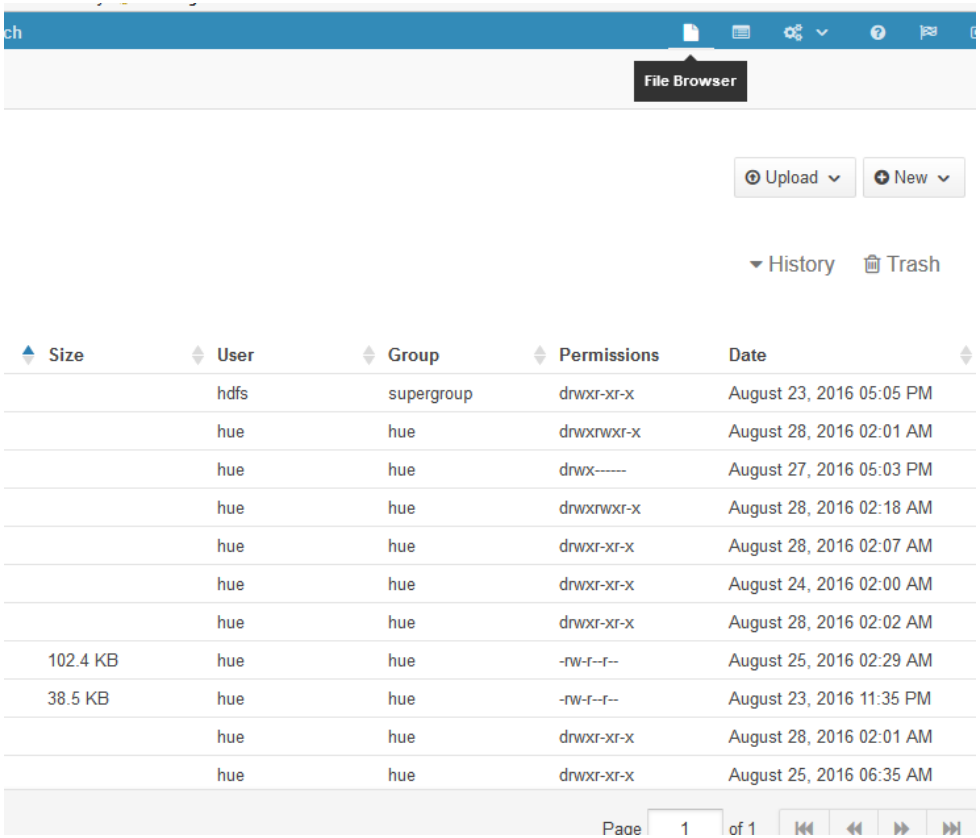
- The unique id represents location of the sensor. We have modified it to represent the latitude and longitude positions of the sensor location.
- For the attributes that contains the date, we have modified the data so that it represents the days in the week. The days are represented in numbers, where 1 is Monday and 7 is Sunday.
- For the specific time data, the time is split from a 24 hours format to a 15 minutes interval format. So, there are 96 different time intervals, where 0 represents the time interval 00:00 to 00:15.
- The metrics used in this dataset is kilometers, so distance is represented as kilometers and the speed is kilometers per hour.



Move your File to HDFS

- Login to Hue With your Account
- Click File Browser in the right side of the page
- Upload your file to your folder





Traffic Big Data Prediction

```
from __future__ import print_function

from pyspark import SparkContext
# $example on$
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
from pyspark.mllib.util import MLUtils
# $example off$

if __name__ == "__main__":

    sc = SparkContext(appName="TrafiicDataProcessing")

    # $example on$
    # Load and parse the data file into an RDD of LabeledPoint.
    data = MLUtils.loadLibSVMFile(sc, 'hdfs://hdp-00:8020/<your file path>')
    # Split the data into training and test sets (30% held out for testing)
    (trainingData, testData) = data.randomSplit([0.7, 0.3])
```

Traffic Big Data Prediction

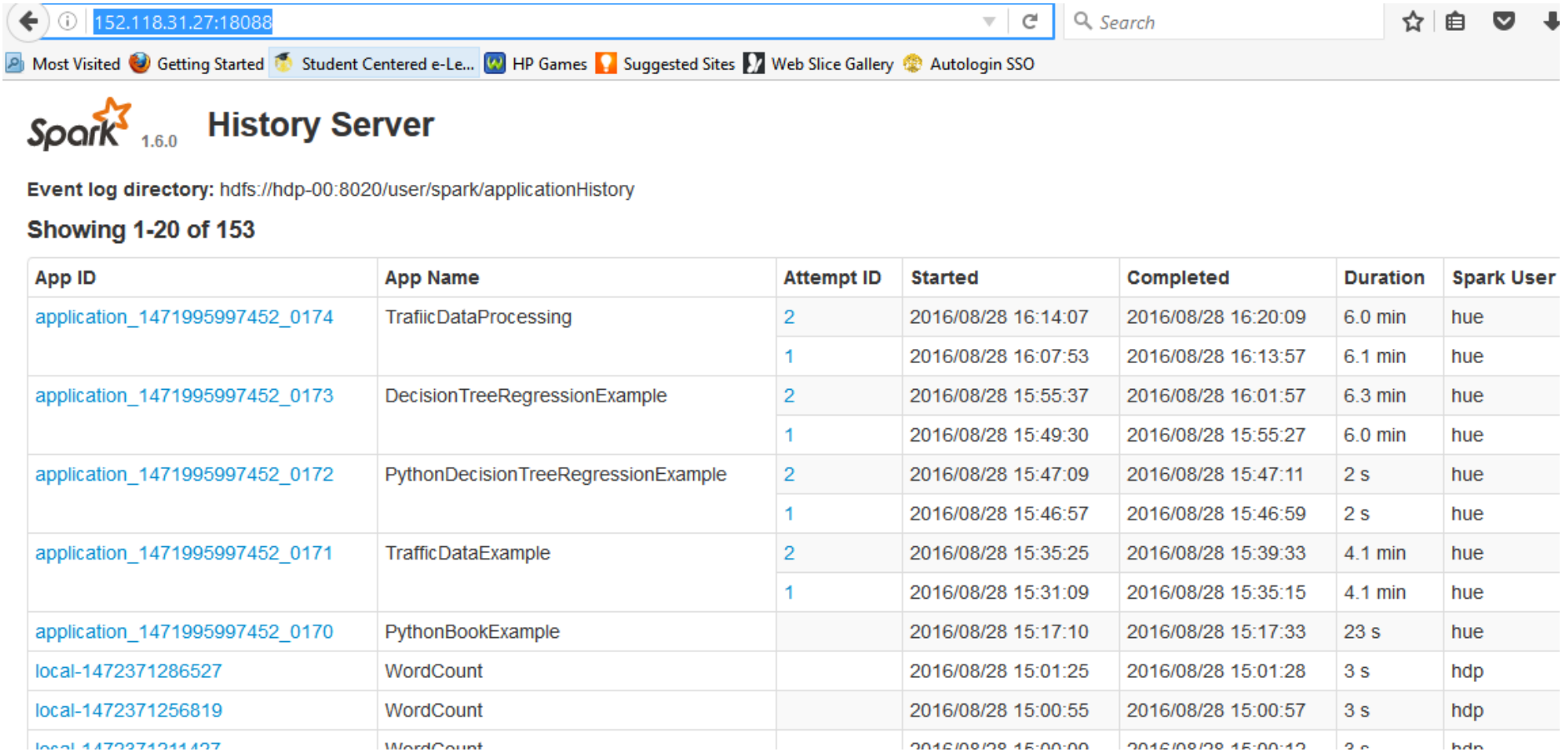
```
# Train a DecisionTree model.
  #  Empty categoricalFeaturesInfo indicates all features are continuous.
  model = DecisionTree.trainRegressor(trainingData, categoricalFeaturesInfo={},
                                      impurity='variance', maxDepth=5, maxBins=32)

  # Evaluate model on test instances and compute test error
  predictions = model.predict(testData.map(lambda x: x.features))
  labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictions)
  testMSE = labelsAndPredictions.map(lambda (v, p): (v - p) * (v - p)).sum() /\
     float(testData.count())
  print('Test Mean Squared Error = ' + str(testMSE))
  print('Learned regression tree model:')
  print(model.toDebugString())
```

Command Execution in Cluster

sudo -u hue spark-submit --name spam --driver-memory 1g --master yarn --deploy-mode cluster spamdetect.py

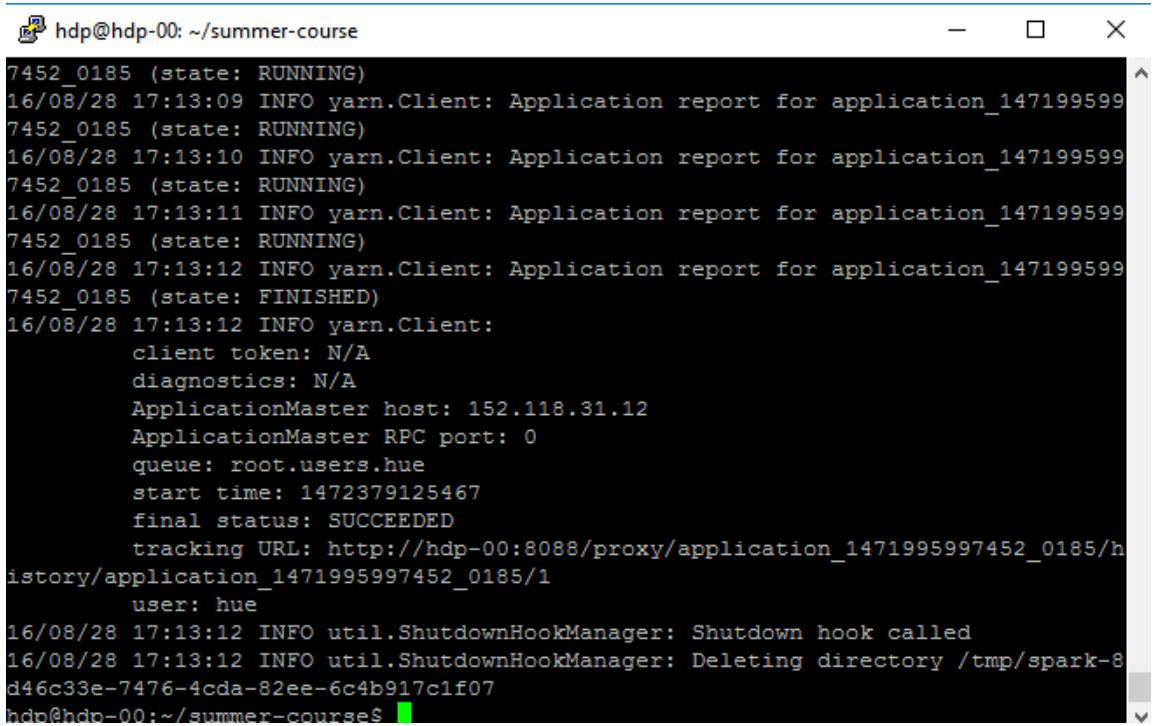
Monitor your Spark Job

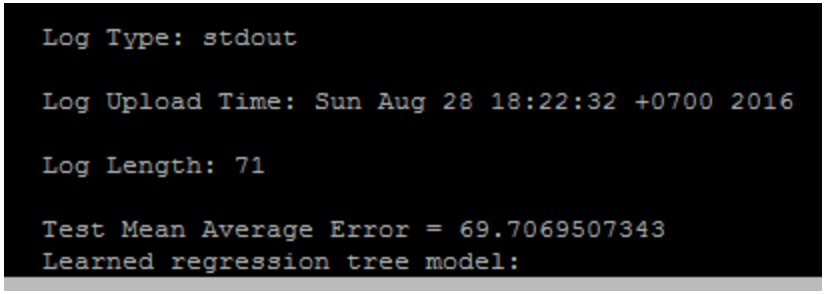
- Go to : http://<ip:address>:18088/ with your browser



App ID	App Name	Attempt ID	Started	Completed	Duration	Spark User
application_1471995997452_0174	TrafiicDataProcessing	2	2016/08/28 16:14:07	2016/08/28 16:20:09	6.0 min	hue
		1	2016/08/28 16:07:53	2016/08/28 16:13:57	6.1 min	hue
application_1471995997452_0173	DecisionTreeRegressionExample	2	2016/08/28 15:55:37	2016/08/28 16:01:57	6.3 min	hue
		1	2016/08/28 15:49:30	2016/08/28 15:55:27	6.0 min	hue
application_1471995997452_0172	PythonDecisionTreeRegressionExample	2	2016/08/28 15:47:09	2016/08/28 15:47:11	2 s	hue
		1	2016/08/28 15:46:57	2016/08/28 15:46:59	2 s	hue
application_1471995997452_0171	TrafficDataExample	2	2016/08/28 15:35:25	2016/08/28 15:39:33	4.1 min	hue
		1	2016/08/28 15:31:09	2016/08/28 15:35:15	4.1 min	hue
application_1471995997452_0170	PythonBookExample		2016/08/28 15:17:10	2016/08/28 15:17:33	23 s	hue
local-1472371286527	WordCount		2016/08/28 15:01:25	2016/08/28 15:01:28	3 s	hdp
local-1472371256819	WordCount		2016/08/28 15:00:55	2016/08/28 15:00:57	3 s	hdp

Job : Succeeded





References

- Holden Karau, et al, Learning Spark, Lightning-Fast Data Analysis, February 2015, Databricks, O'Reilly Media
- Ari Wibisono, et. al, Traffic big data prediction and visualization using Fast Incremental Model Trees-Drift Detection (FIMT-DD), Knowledge-Based Systems, Volume 93, 1 February 2016,