**CS590RA Real-Time Big Data Analytics**

**Challenge #2: Hackathon**

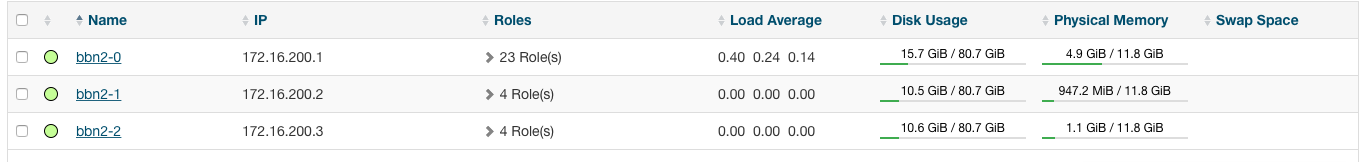
G9:

Mayanka Chandra Shekar

Shuai Jack Zhao

Supervisor: Prof. **Lee, Yugyung**

**Network Abnormity in Campus Network Netflow Traffic using Kafka/Storm/Weka**

1. Report:
2. KafKa/Storm explanation:
   1. We use three nodes configuration.
3. 
   1. Kafka
      1. Code Explaination
   2. Storm

Train Bolt This an extension to train topology

Receives the kafka data, and extracts the features. and add an ? to the feature to be classified using weka

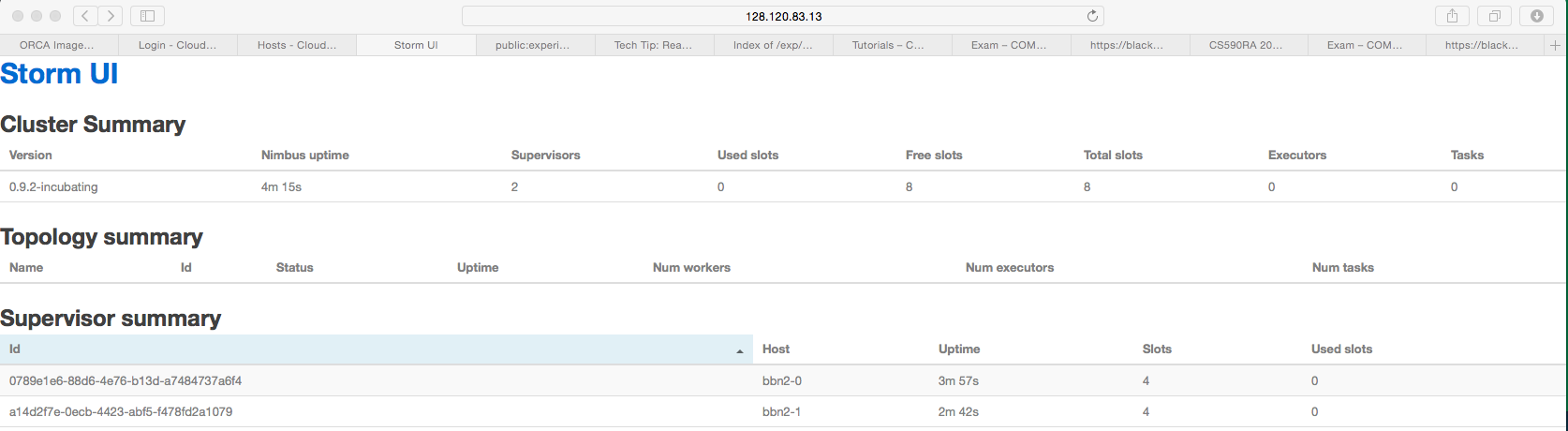
Test bolt

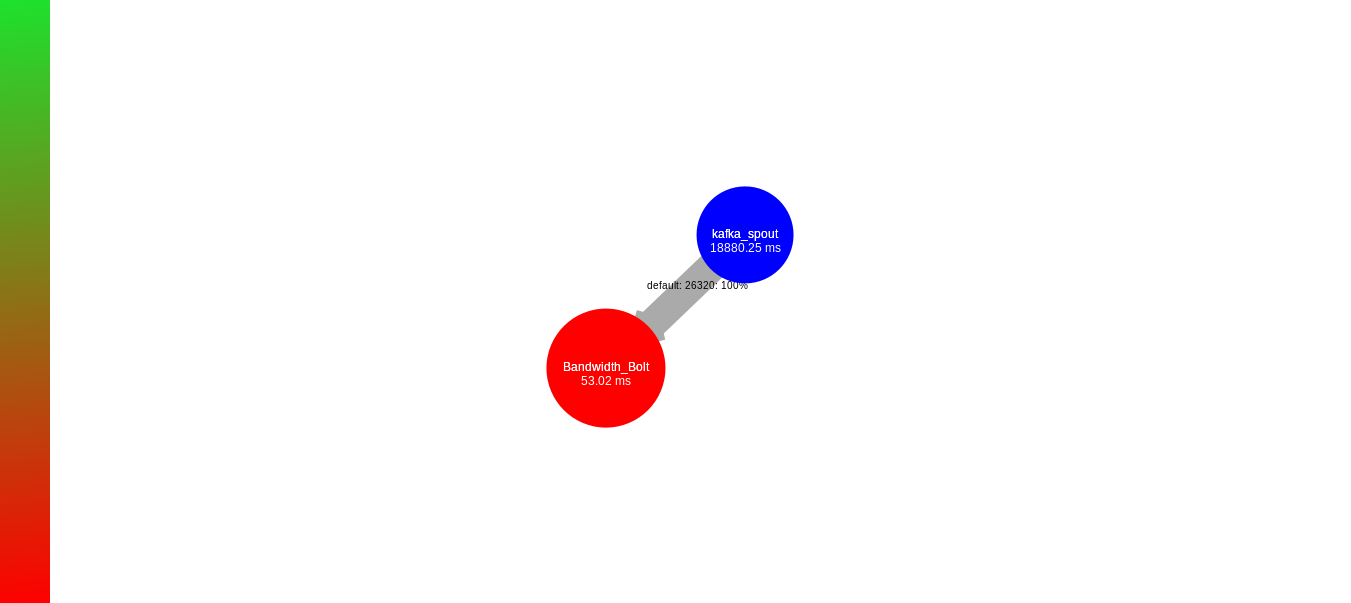
Reads the train file and create an classifier model (here we use Naive Bayes)

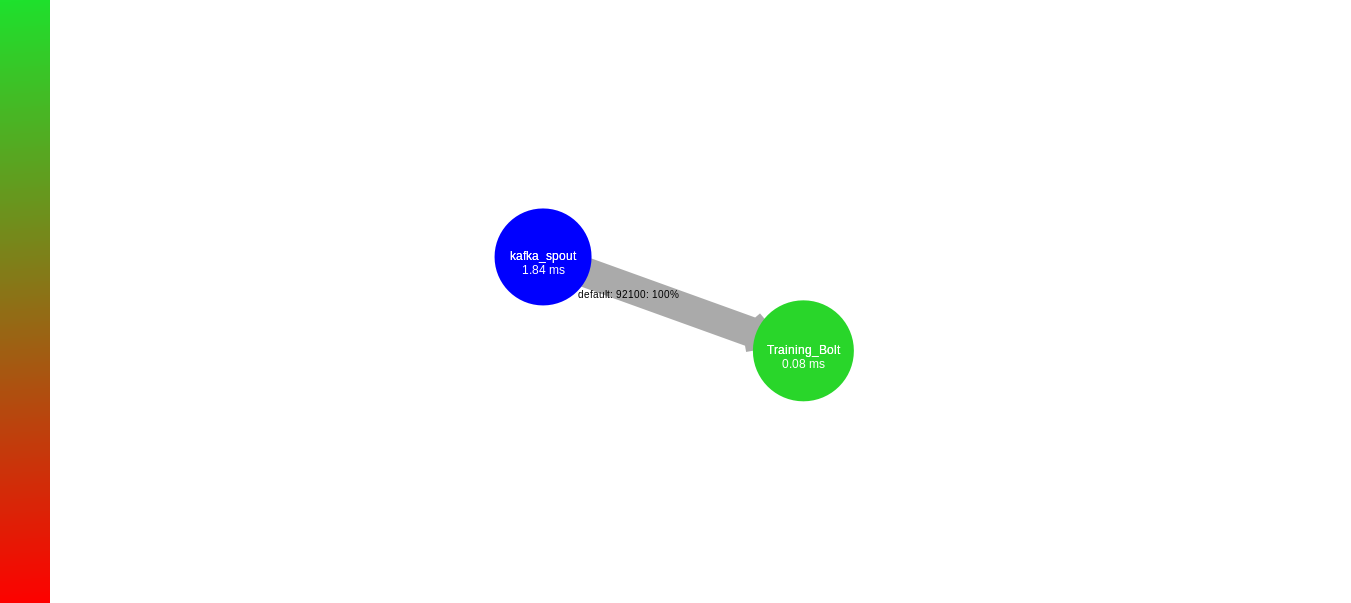
Then using the model, tests the data stred test.csv and displyas the output.

Kafka spout emits the network data.

Train Bolt retrives it and extracts the neccesary features.

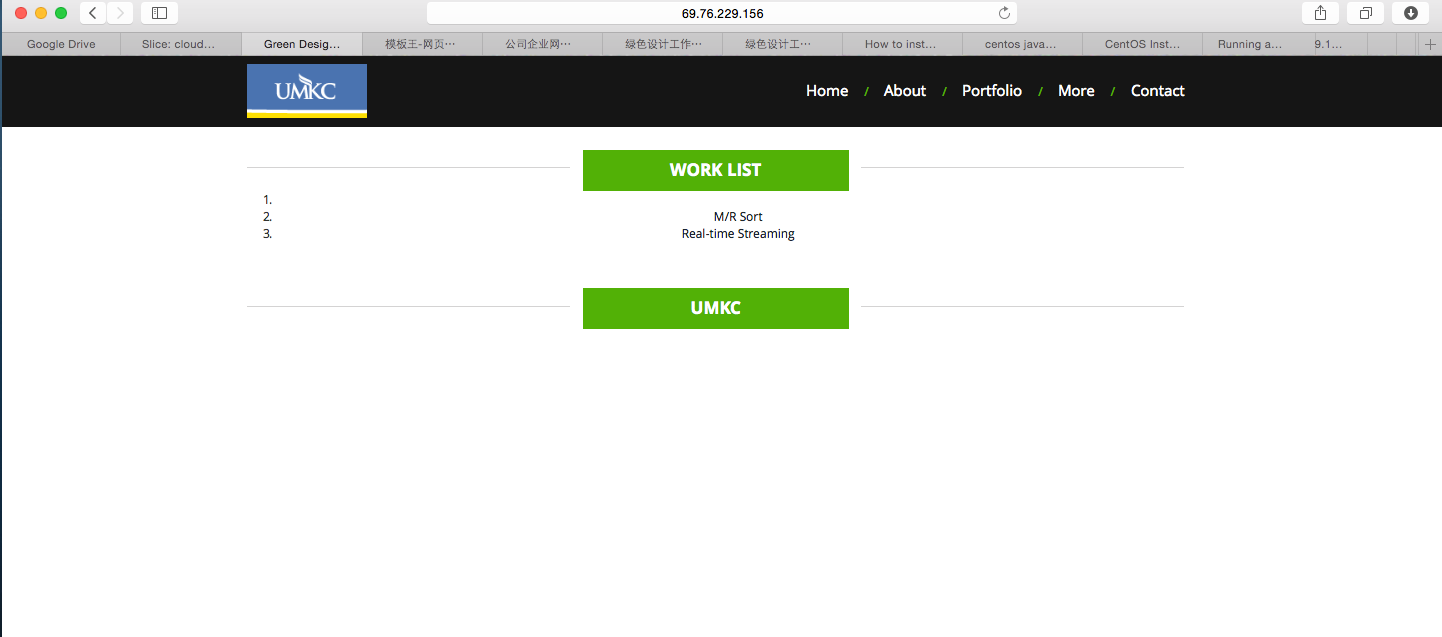


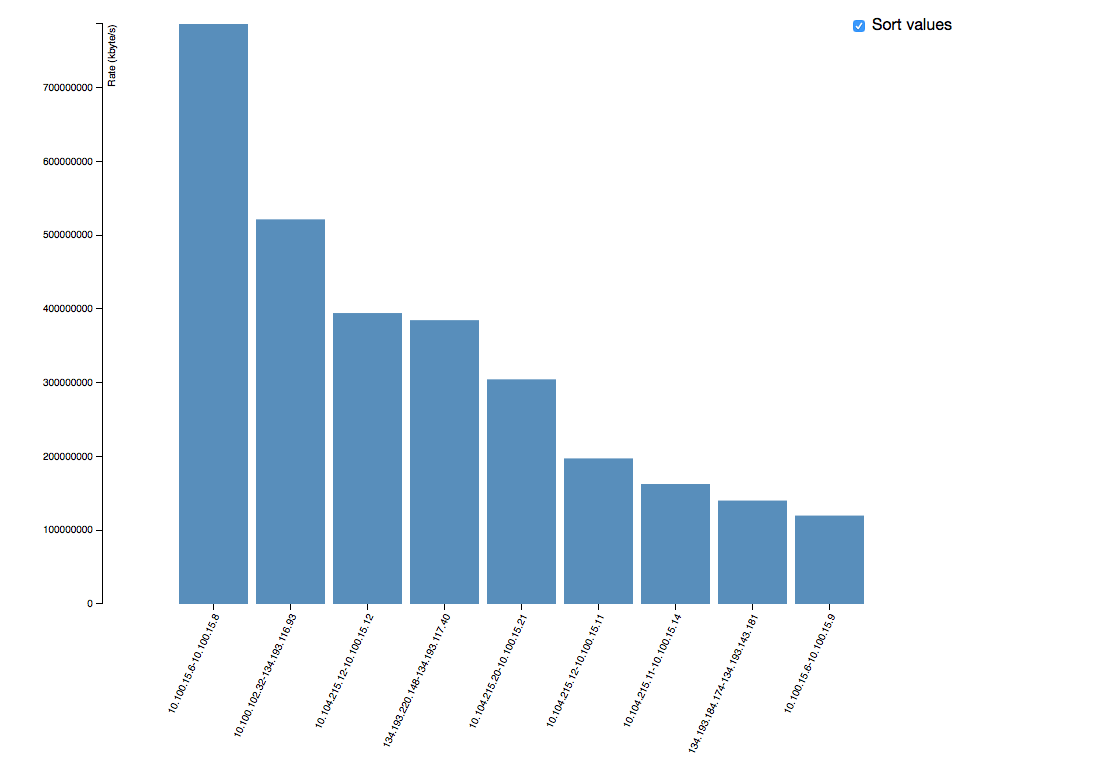




1. Web Server
   1. ScreenShot

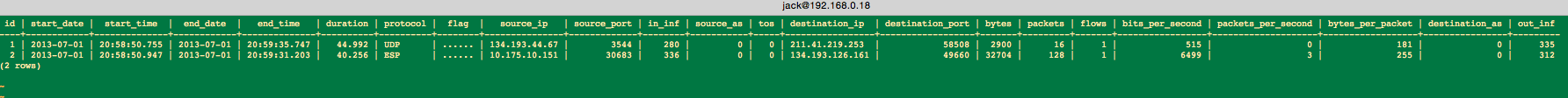






Notes: for the real time plot, we still have issues combing javascript to our webserver

1. Data
   1. ScreenShot



This is one sample dataset that is saved at postgresql database. It contains network information such sourceIP, destinationIP, protocol, bandwidth and etc.

1. Analysis
   1. DataCollection

Attached

b. FeatureExtraction: Features extracted from the data where Source ip address , destination ip address , protocol and bandwidth

From these features status of the network connection is being classified.

The classification of the status are as follows

       statusClassList.add("TCP Connection");

       statusClassList.add("TCP Data Transfer");

       statusClassList.add("UDP Connection");

       statusClassList.add("UDP  Data Transfer");

       statusClassList.add("Unknown");

       statusClassList.add("Backup");

1. statusClassList.add("Abnormal");

|  |  |
| --- | --- |
| @RELATION netflow\_test  @ATTRIBUTE SrcIP STRING (x.x.x.x)  @ATTRIBUTE DstIP STRING (x.x.x.x)  @ATTRIBUTE Proto STRING {TCP, UDP}  @ATTRIBUTE BandWidth Numeric  @ATTRIBUTE class {C1, C2. C3, C4, C5}  @DATA |  |
|  |

1. Classification
   * 1. UDP Connection
     2. TCP Connection
     3. UDP Data Transfer
     4. Unknown
     5. Abnormal
     6. UDP Data Transfer
2. Evaluation

UDP 10.192.10.18 134.193.44.67 0.0 UDP Connection

UDP 10.205.207.4 134.193.1.2 0.0 UDP Connection

TCP 10.139.59.48 134.193.235.3 0.0 TCP Connection

TCP 10.193.3.147 10.139.20.1 0.0 TCP Connection

TCP 10.193.3.147 10.139.20.31 0.0 TCP Connection

TCP 10.193.3.147 10.139.20.9 0.0 TCP Connection

UDP 10.182.0.231 134.193.83.4 0.0 UDP Connection

UDP 134.193.160.143 10.100.100.18 0.0 UDP Connection

TCP 10.193.3.147 10.139.20.74 0.0 TCP Connection

TCP 10.193.3.147 10.139.20.80 0.0 TCP Connection

TCP 10.193.3.147 10.139.20.41 0.0 TCP Connection

UDP 134.193.176.227 134.193.1.2 236.0 UDP Data Transfer

ESP 10.139.40.26 10.193.3.129 1194.0 Unknown

TCP 134.193.126.153 10.245.0.181 134.0 Abnormal

UDP 134.193.44.67 134.193.185.39 278.0 UDP Data Transfer

ESP 134.193.55.168 134.193.44.67 149.0 Unknown

ESP 134.193.156.5 134.193.44.67 148.0 Unknown

ESP 134.193.221.29 134.193.44.63 149.0 Unknown

ESP 134.193.29.4 134.193.44.67 149.0 Unknown

1. Validation
2. Accuracy
3. Input data
   1. Training.csv, Testing.txt,Training.
4. Results
5. Reference
   1. Umkc blackboard

**Submission:**

Code Subbmission:

1. blackboard:
2. GitHub:

Report sbumisson:

1. blackboard:
2. DataSets
   1. Training data
   2. Testing data
      1. 10-fold
3. Feature Extraction

|  |  |
| --- | --- |
| @RELATION netflow\_test  @ATTRIBUTE SrcIP STRING (x.x.x.x)  @ATTRIBUTE DstIP STRING (x.x.x.x)  @ATTRIBUTE Proto STRING {TCP, UDP}  @ATTRIBUTE BandWidth Numeric  @ATTRIBUTE class {C1, C2. C3, C4, C5}  @DATA |  |
|  |

1. Classification
2. Evaluation/Validation
3. WebApp : 69.76.229.156:8001

Train.code

Training data

Netmain:

package main.java.netks;

import org.apache.log4j.BasicConfigurator;

import org.apache.log4j.Logger;

import storm.kafka.KafkaSpout;

import storm.kafka.SpoutConfig;

import storm.kafka.StringScheme;

import storm.kafka.ZkHosts;

import backtype.storm.Config;

import backtype.storm.LocalCluster;

import backtype.storm.generated.StormTopology;

import backtype.storm.StormSubmitter;

import backtype.storm.spout.SchemeAsMultiScheme;

import backtype.storm.topology.TopologyBuilder;

public class Netmain

{

private final Logger LOGGER = Logger.getLogger(this.getClass());

private static final String KAFKA\_TOPIC ="new";

public static void main(String[] args) {

// TODO Auto-generated method stub

BasicConfigurator.configure();

if (args != null && args.length > 0)

{

try {

StormSubmitter.submitTopology(

args[0],

createConfig(false),

createTopology());

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

else

{

LocalCluster cluster = new LocalCluster();

cluster.submitTopology(

"network-analysis",

createConfig(true),

createTopology());

try {

Thread.sleep(60000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

cluster.shutdown();

}

}

private static StormTopology createTopology()

{

SpoutConfig kafkaConf = new SpoutConfig(

new ZkHosts("localhost:2181"),

KAFKA\_TOPIC,

"/kafka",

"KafkaSpout");

kafkaConf.scheme = new SchemeAsMultiScheme(new StringScheme());

TopologyBuilder topology = new TopologyBuilder();

// topology.setSpout("k\_spout", new Kspout(), 4);

topology.setSpout("kafka\_spout", new KafkaSpout(kafkaConf), 4);

topology.setBolt("Training\_Bolt", new TrainingBolt(), 4)

.shuffleGrouping("kafka\_spout");

// topology.setBolt("Bandwidth\_Bolt", new BandwidthBolt(), 4)

// .shuffleGrouping("kafka\_spout");

// topology.setBolt("CounterBolt", new CounterBolt(), 4)

//.shuffleGrouping("Bandwidth\_Bolt");

/\* topology.setBolt("TopBolt", new TopBolt(), 4)

.shuffleGrouping("CounterBolt");

topology.setBolt("PrintBolt", new PrintBolt(), 4)

.shuffleGrouping("TopBolt");\*/

return topology.createTopology();

}

private static Config createConfig(boolean local)

{

int workers = 1;

Config conf = new Config();

conf.setDebug(true);

if (local)

conf.setMaxTaskParallelism(workers);

else

conf.setNumWorkers(workers);

return conf;

}

}

Training Bolt

package main.java.netks;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.io.Serializable;

import java.util.HashMap;

import java.util.Map;

import org.apache.log4j.Logger;

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

public class TrainingBolt extends BaseBasicBolt implements Serializable{

private static final long serialVersionUID = 42L;

private static final Logger LOGGER =

Logger.getLogger(TrainingBolt.class);

// private static final ObjectMapper mapper = new ObjectMapper();

File file = new File("/root/Downloads/documents/train.csv");

String text,status;

String[] str;

String sip;

FileWriter f=null,f1=null;

BufferedWriter b=null,b1=null;

/\*public TrainingBolt()

{

text="PROTOCOL \t SOURCE\_IP \t DESTINATION\_IP \t STATUS \n";

try

{

if(!file.exists())

{

file.createNewFile();

}

f = new FileWriter(file.getAbsoluteFile(),true);

b = new BufferedWriter(f);

}

catch(Exception ex)

{

ex.printStackTrace();

}

try

{

b.write(text);

b.close();

} catch (IOException e)

{

// TODO Auto-generated catch block

e.printStackTrace();

}

}

\*/

public void execute(Tuple t1, BasicOutputCollector arg1) {

// TODO Auto-generated method stub

String d=t1.getString(0);

str=d.split(" "); //Split the data

if(str[6].equals("TCP"))

{

if(str[18].equals("0.0"))

{

status="TCP Connection";

}

else

{

status="TCP Data Transfer";

}

}

else if(str[6].equals("UDP"))

{

if(str[18].equals("0.0"))

{

status="UDP Connection";

}

else

{

status="UDP Data Transfer";

}

}

else

{

status="Unknown";

}

if(str[8].contains("10.100.15")||str[13].contains("10.100.15"))

{

status="Backup";

}

if(str[8].equals("134.193.126.153")||str[13].equals("134.193.126.153"))

{

status="Abnormal";

}

text=str[6]+"\t"+str[8]+"\t"+str[13]+"\t"+str[18]+"\t"+status+"\n";

try

{

if(!file.exists())

{

file.createNewFile();

}

f = new FileWriter(file.getAbsoluteFile(),true);

b = new BufferedWriter(f);

}

catch(Exception ex)

{

ex.printStackTrace();

}

try

{

b.write(text);

b.close();

} catch (IOException e)

{

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public void declareOutputFields(OutputFieldsDeclarer arg0) {

// TODO Auto-generated method stub

}

}

Test Code

Testing data

Netmain

package main.java.netks;

import org.apache.log4j.BasicConfigurator;

import org.apache.log4j.Logger;

import storm.kafka.KafkaSpout;

import storm.kafka.SpoutConfig;

import storm.kafka.StringScheme;

import storm.kafka.ZkHosts;

import backtype.storm.Config;

import backtype.storm.LocalCluster;

import backtype.storm.generated.StormTopology;

import backtype.storm.StormSubmitter;

import backtype.storm.spout.SchemeAsMultiScheme;

import backtype.storm.topology.TopologyBuilder;

public class Netmain

{

private final Logger LOGGER = Logger.getLogger(this.getClass());

private static final String KAFKA\_TOPIC ="new";

public static void main(String[] args) {

// TODO Auto-generated method stub

BasicConfigurator.configure();

if (args != null && args.length > 0)

{

try {

StormSubmitter.submitTopology(

args[0],

createConfig(false),

createTopology());

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

else

{

LocalCluster cluster = new LocalCluster();

cluster.submitTopology(

"network-analysis",

createConfig(true),

createTopology());

try {

Thread.sleep(60000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

cluster.shutdown();

}

}

private static StormTopology createTopology()

{

SpoutConfig kafkaConf = new SpoutConfig(

new ZkHosts("localhost:2181"),

KAFKA\_TOPIC,

"/kafka",

"KafkaSpout");

kafkaConf.scheme = new SchemeAsMultiScheme(new StringScheme());

TopologyBuilder topology = new TopologyBuilder();

// topology.setSpout("k\_spout", new Kspout(), 4);

topology.setSpout("kafka\_spout", new KafkaSpout(kafkaConf), 4);

topology.setBolt("Training\_Bolt", new TrainingBolt(), 4)

.shuffleGrouping("kafka\_spout");

topology.setBolt("Test\_Bolt", new TestBolt(), 4)

.shuffleGrouping("Training\_Bolt");

// topology.setBolt("Bandwidth\_Bolt", new BandwidthBolt(), 4)

// .shuffleGrouping("kafka\_spout");

// topology.setBolt("CounterBolt", new CounterBolt(), 4)

//.shuffleGrouping("Bandwidth\_Bolt");

/\* topology.setBolt("TopBolt", new TopBolt(), 4)

.shuffleGrouping("CounterBolt");

topology.setBolt("PrintBolt", new PrintBolt(), 4)

.shuffleGrouping("TopBolt");\*/

return topology.createTopology();

}

private static Config createConfig(boolean local)

{

int workers = 1;

Config conf = new Config();

conf.setDebug(true);

if (local)

conf.setMaxTaskParallelism(workers);

else

conf.setNumWorkers(workers);

return conf;

}

}

Train bolt

package main.java.netks;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.io.Serializable;

import java.util.HashMap;

import java.util.Map;

import org.apache.log4j.Logger;

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

import backtype.storm.tuple.Values;

public class TrainingBolt extends BaseBasicBolt implements Serializable{

private static final long serialVersionUID = 42L;

private static final Logger LOGGER =

Logger.getLogger(TrainingBolt.class);

// private static final ObjectMapper mapper = new ObjectMapper();

File file = new File("/root/Downloads/documents/test.csv");

String text,status;

String[] str;

String sip;

FileWriter f=null,f1=null;

BufferedWriter b=null,b1=null;

public void execute(Tuple t1, BasicOutputCollector boc) {

// TODO Auto-generated method stub

String d=t1.getString(0);

str=d.split(" "); //Split the data

/\* if(str[6].equals("TCP"))

{

if(str[18].equals("0.0"))

{

status="TCP Connection";

}

else

{

status="TCP Data Transfer";

}

}

else if(str[6].equals("UDP"))

{

if(str[18].equals("0.0"))

{

status="UDP Connection";

}

else

{

status="UDP Data Transfer";

}

}

else

{

status="Unknown";

}

if(str[8].contains("10.100.15")||str[13].contains("10.100.15"))

{

status="Backup";

}

if(str[8].equals("134.193.126.153")||str[13].equals("134.193.126.153"))

{

status="Abnormal";

}

\*/ text=str[6]+"\t"+str[8]+"\t"+str[13]+"\t"+str[18]+"\t ? \n";

try

{

if(!file.exists())

{

file.createNewFile();

}

f = new FileWriter(file.getAbsoluteFile(),true);

b = new BufferedWriter(f);

}

catch(Exception ex)

{

ex.printStackTrace();

}

try

{

b.write(text);

b.close();

} catch (IOException e)

{

// TODO Auto-generated catch block

e.printStackTrace();

}

boc.emit(new Values("test","test"));

}

public void declareOutputFields(OutputFieldsDeclarer d1) {

// TODO Auto-generated method stub

d1.declare(new Fields("tweet\_id", "tweet\_text"));

}

}

Test bolt

package main.java.netks;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.ObjectInputStream;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import java.util.Random;

import java.util.SortedMap;

import java.util.StringTokenizer;

import java.util.TreeMap;

import org.apache.commons.lang.StringEscapeUtils;

import org.apache.http.HttpResponse;

import org.apache.http.client.ClientProtocolException;

import org.apache.http.client.methods.HttpGet;

import org.apache.http.impl.client.DefaultHttpClient;

import org.apache.log4j.Logger;

import weka.classifiers.Classifier;

import weka.classifiers.Evaluation;

import weka.classifiers.bayes.NaiveBayes;

import weka.core.Attribute;

import weka.core.Instance;

import weka.core.Instances;

import weka.core.SparseInstance;

import weka.core.converters.CSVLoader;

import backtype.storm.topology.BasicOutputCollector;

import backtype.storm.topology.OutputFieldsDeclarer;

import backtype.storm.topology.base.BaseBasicBolt;

import backtype.storm.tuple.Fields;

import backtype.storm.tuple.Tuple;

public class TestBolt extends BaseBasicBolt{

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private ArrayList<String> featureWords;

private ArrayList<Attribute> attributeList;

private Instances inputDataset;

//! private POSTagger posTagger;

private Classifier classifier;

private ArrayList<String> statusClassList;

// private Instances testingInstances0;

public Instances testingInstances ;// private Instances testingInstances;

File file = new File("/root/Downloads/documents/Toutput.txt");

File file2 = new File("/root/Downloads/documents/Toutput2.txt");

Fields \_outFields;

private static final Logger LOGGER =

Logger.getLogger(TestBolt.class);

//private static final ObjectMapper mapper = new ObjectMapper();

BufferedReader br = null;

SortedMap<Instance,Instance> map = new TreeMap<Instance,Instance>();

List<String> documents = new ArrayList<String>();

public TestBolt()

{

attributeList = new ArrayList<Attribute>();

initialize();

}

private void initialize()

{

ObjectInputStream ois = null;

try {

//reads the feature words list to a hashset

ois = new ObjectInputStream(new FileInputStream("/root/Downloads/documents/FeatureWordsList.dat"));

featureWords = (ArrayList<String>) ois.readObject();

} catch (Exception ex) {

System.out.println("Exception in Deserialization");

} finally {

try {

ois.close();

} catch (IOException ex) {

System.out.println("Exception while closing file after Deserialization");

}

}

//creating an attribute list from the list of feature words

statusClassList = new ArrayList<String>();

statusClassList.add("TCP Connection");

statusClassList.add("TCP Data Transfer");

statusClassList.add("UDP Connection");

statusClassList.add("UDP Data Transfer");

statusClassList.add("Unknown");

statusClassList.add("Backup");

statusClassList.add("Abnormal");

for(String featureWord : featureWords)

{

attributeList.add(new Attribute(featureWord));

}

//the last attribute reprsents ths CLASS (status) of the tweet

attributeList.add(new Attribute("Status",statusClassList));

}

public void trainClassifier(final String INPUT\_FILENAME)

{

getTrainingDataset(INPUT\_FILENAME);

//trainingInstances consists of feature vector of every input

Instances trainingInstances = createInstances("TRAINING\_INSTANCES");

for(Instance currentInstance : inputDataset)

{

//extractFeature method returns the feature vector for the current input

// Instance currentFeatureVector = extractFeature(currentInstance);

//Make the currentFeatureVector to be added to the trainingInstances

// currentFeatureVector.setDataset(trainingInstances);

trainingInstances.add(currentInstance);

}

//You can create the classifier that you want. In this tutorial we use NaiveBayes Classifier

//For instance classifier = new SMO;

classifier = new NaiveBayes();

try {

//classifier training code

classifier.buildClassifier(trainingInstances);

//storing the trained classifier to a file for future use

weka.core.SerializationHelper.write("NaiveBayes.model",classifier);

} catch (Exception ex) {

System.out.println("Exception in training the classifier.");

}

}

public void testClassifier(final String INPUT\_FILENAME)

{

Random ran = new Random();

FileWriter fw = null;

BufferedWriter bw = null;

try {

if (!file.exists()) {

file.createNewFile();

}

fw = new FileWriter(file.getAbsoluteFile(),true);

bw = new BufferedWriter(fw);

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

getTrainingDataset(INPUT\_FILENAME);

//trainingInstances consists of feature vector of every input

testingInstances = createInstances("TESTING\_INSTANCES");

for(Instance currentInstance : inputDataset)

{

//extractFeature method returns the feature vector for the current input

//Instance currentFeatureVector = extractFeature(currentInstance);

//Make the currentFeatureVector to be added to the trainingInstances

//currentFeatureVector.setDataset(testingInstances);

testingInstances.add(currentInstance);

}

try {

classifier = (Classifier) weka.core.SerializationHelper.read("NaiveBayes.model");

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

try {

//Classifier deserialization

// String row = "row1";

// String columnFamily="Status";

// String column="col";

//classifier testing code

for(Instance testInstance : testingInstances)

{

//int count =ran.nextInt(10000);

double score = classifier.classifyInstance(testInstance);

System.out.println(testingInstances.attribute("Status").value((int)score));

bw.write(testInstance.stringValue(1)+","+testingInstances.attribute("Status").value((int)score)+"\n");

// bw.write(testingInstances.attribute("Status").value((int)score));

// String value = testingInstances.attribute("Status").value((int)score);

bw.write(testingInstances.attribute("Status").value((int)score)+"\n");

bw.close();

}

} catch (Exception ex) {

System.out.println("Exception in testing the classifier.");

}

try {

Evaluation eval = new Evaluation(inputDataset);

eval.evaluateModel(classifier, testingInstances);

System.out.println(eval.toSummaryString("\nResults\n======\n", false));

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

private void getTrainingDataset(final String INPUT\_FILENAME)

{

try{

//reading the training dataset from CSV file

CSVLoader trainingLoader =new CSVLoader();

trainingLoader.setSource(new File(INPUT\_FILENAME));

inputDataset = trainingLoader.getDataSet();

}catch(IOException ex)

{

System.out.println("Exception in getTrainingDataset Method");

}

}

private Instances createInstances(final String INSTANCES\_NAME)

{

//create an Instances object with initial capacity as zero

Instances instances = new Instances(INSTANCES\_NAME,attributeList,0);

//sets the class index as the last attribute (positive or negative)

instances.setClassIndex(instances.numAttributes()-1);

return instances;

}

/\*

private Instance extractFeature(Instance inputInstance)

{

Map<Integer,Double> featureMap = new TreeMap<>();

int indices[] = new int[featureMap.size()+1];

double values[] = new double[featureMap.size()+1];

int i=0;

for(Map.Entry<Integer,Double> entry : featureMap.entrySet())

{

indices[i] = entry.getKey();

values[i] = entry.getValue();

i++;

}

indices[i] = featureWords.size();

values[i] = (double)statusClassList.indexOf(inputInstance.stringValue(1));

return new SparseInstance(1.0,values,indices,featureWords.size());

}

\*/ public void execute(Tuple arg0, BasicOutputCollector arg1) {

// TODO Auto-generated method stub

File csv = new File ("/root/Downloads/documents/test.csv");

if(csv.length()>0)

{

trainClassifier("/root/Downloads/documents/train.csv");

testClassifier("/root/Downloads/documents/test.csv");

}

}

public void declareOutputFields(OutputFieldsDeclarer arg0) {

// TODO Auto-generated method stub

}

}