Study and Development of Feature Discovery Model (Cancer cell Analysis)

Fall 2016

Final Report

Student ID:

Gupta, Arunit - 7

Patel, Marmik - 19

Objective

Nowadays, machine learning and big data is becoming the integral part of intelligent application. There are lot of open source repositories and tools available for machine learning. But there are very few architectures which combines machine learning with big data. The objective of this directed reading is to design the architecture for integration of machine learning and big data which uses available open source tools. The objective includes creation of application for detection of features of the streaming video using the big data framework to achieve high performance. Features of video can be defined as change in scene, frame detection, audio detection, face detection, object tracking. This will help user to analyze the video in terms of features and can help them to search another related videos using features identified, compare video with another, for analyzing the current trends among people. The directed reading involves features extraction of video using libraries, creating distributed framework which includes open source big data software like Apache Storm, Apache Kafka, Apache Spark, using machine learning algorithm for training and testing the model and annotate the video with extracted feature. Various Machine Learning algorithm involves Decision tree, Random Forest and Naïve Bayes algorithm.

The overall objective is to build a distributed model which is capable of detecting the features and functionality present in the streaming video and annotate the video based on the extracted features. Input provided is the streaming video of cancer cells taken as stream of MR Images of Brain, Breast, Lung and Prostate.

Expected Outcomes:

The primary objective of this course is to develop an application which is capable of detecting the features of the videos based on the define model. Expected output is to detect the type of cancer and stages of the cancer.

Progress Plan

1. Proposing and Building workflow model for the project.
2. Analysis of Health domain and search for related streaming input videos.
3. Implementing mechanism to fetch the live streaming or static videos
4. Study of Spark, Storm and Kafka frameworks.
5. Extraction of various parameter from videos using OpenIMAJ library.
6. Extracting Image features for processing: SIFT, HOG and Color.
7. Study and Implementation of Video Extraction techniques: Frame slicing and shot transition.
8. Audio Extraction and Feature vector generation/selection/filtering/compression.
9. Implementing storm with feature vector.
10. Integrating and connecting Spark, Storm and Kafka.
11. Base64 implementation for Video conversion from Kafka to Spark.
12. Implementing machine learning algorithm for testing and training video streams.
13. Study and implementation of Annotating videos based on feature extraction
14. Combining all modules and techniques and apply to live streaming video and annotate video based on extracted features

Project Objectives

Significance

This Application is capable of detecting cancer cells of various cancer cells. Application is trained with multiple images of cancer and tested on stream of videos which helps to test the type of cancer.

We have tried multiple machine learning algorithm like decision tree and random forest algorithm and compared accuracy results of algorithms.

Technologies include Big data technologies where we can provide large input on an Enterprise level by providing streaming data on a large scale and provide real time results. Apache Storm is stream processing engine where we can process streaming data and provide results. Kafka is chosen as a messaging system which is a pub sub system which has high throughput. For machine learning we have used different algorithm for which we have used Spark.

Features: Use Case/Scenario

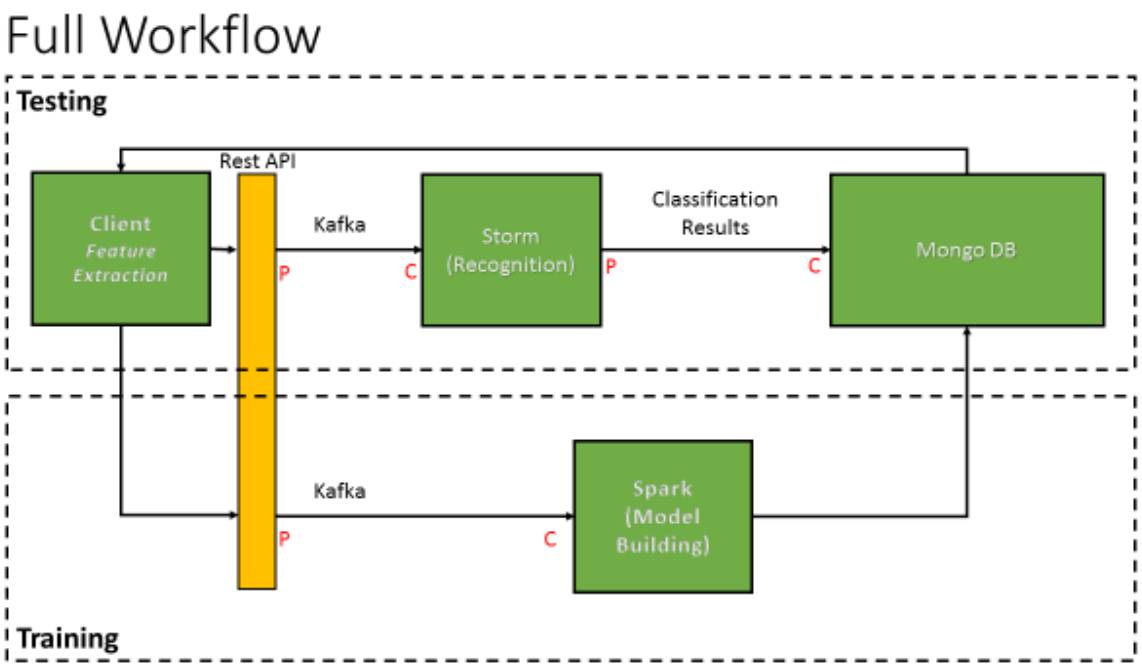
Real time processing of Cancer Cells, there are four types of cancer cells which are Brain, breast, lung and prostate. Training and testing data were considered which are in different ratio. Ratios considered were 50:50, 60:40; 70:30; 80:20; 90:10.

Testing and training sets consist of images of Brain, breast, lung and prostate. Input was provided as Video of streaming MRI's of Brain, breast, lung and prostate, where we have change in cells of cancer. Expected output consists of annotation of the videos where we train the machine itself and use real life scenarios to test what is the type of cancer and geometry shapes to predict the stage of cancer. Various machine learning algorithm should be tested to check the accuracy of the algorithm.

Approach

Full Work Flow

Application approach follow the below Workflow. It consists of Testing and Training data. Where Client interacts with Storm and stores in MongoDB. Spark is used to train the model using machine learning and feature vectors are stored in MongoDB. Rest API is used for interaction.



Training Model

Kafka is used for creating a topic for sending training data, War file of servlet code for REST API and it was deployed on tomcat.

Kafka Consumer is tested for kafka Topic. Spark model builder is tested to generate the model and save the model to MongoDB.

Testing

Kafka Topic was created for sending training data, we need to make sure that REST API is running. Topology Generator was running to create storm topology and storm topology was deployed on storm server. Results were received back to Client from MongoDB.

Research Covered

- Storm and Kafka Integration
- Spark and Kafka Integration
- Video Processing and Feature Vector Implementation

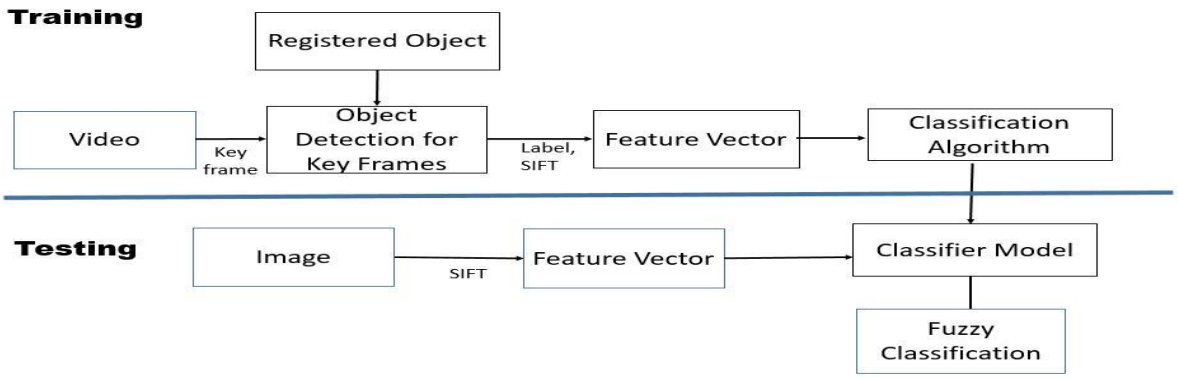
Terminology

- **Apache Spark:** It is an open source cluster computing framework. Originally developed at the University of California, Berkeley's AMP Lab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since. Spark provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance.

- **Apache Storm** is a distributed stream processing computation framework written predominantly in the Clojure programming language. Originally created by Nathan Marz and team at BackType, the project was open sourced after being acquired by Twitter. It uses custom created "spouts" and "bolts" to define information sources and manipulations to allow batch, distributed processing of streaming data. The initial release was on 17 September 2011.

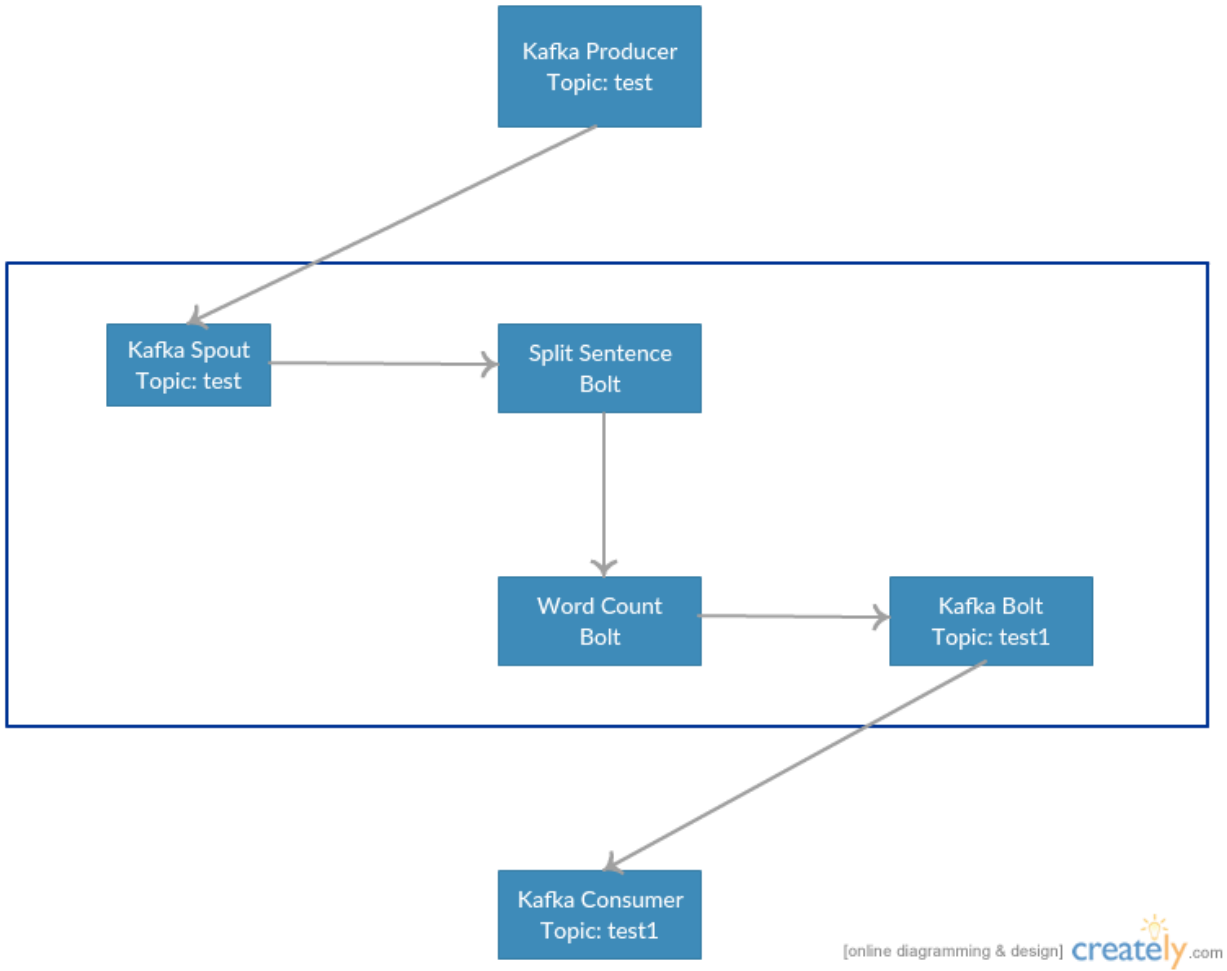
- **Apache Kafka**: Apache Kafka is an open-source message broker project developed by the Apache Software Foundation written in Scala. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds

Old Work Flow Model



Storm -Kafka Connection

Kafka and Storm naturally complement each other, and their powerful cooperation enables real-time streaming analytics for fast-moving big data. Kafka and Storm integration is to make easier for developers to ingest and publish data streams from Storm topologies.



Workflow Explanation:

1. Kafka Producer (Kafka) -> Kafka Consumer (Kafka Spout)
   Kafka spout which works in the storm framework, consumes data from Kafka producer on some topic i.e. test
2. Kafka Spout -> Split sentence bolt
   Spout transfer data to split sentence bolt to split the sentences into words.
3. Split Sentence Bolt -> Word Count Bolt
   Split sentence bolt sent words to Word count bolt to count the words.
4. Word Count Bolt -> Kafka Bolt (Kafka Producer)
   Counts the words and sent it to Kafka Bolt.
5. Kafka Bolt -> Kafka Consumer
   Kafka Bolt is kafka producer in storm framework which produces messages on some topic i.e. test1 and sent it to consumer.

Status: Completed till word count bolt, facing some issue with Word Count bolt to Kafka bolt. It is the last step which needs to be solved for completion of storm – Kafka workflow.

Screenshots

- Word Count Bolt Program

```
/**
 * Created by Mayanka on 17-Sep-15.
 */
public class WordCountBolt extends BaseBasicBolt {
    Map<String, Integer> counts = new HashMap<~>();
    static Logger log;
    @Override
    public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector) {
        String word = tuple.getStringByField("words");
        log = LoggerFactory.getLogger(BaseBasicBolt.class);
        Integer count = counts.get(word);
        if (count == null)
            count = 0;
        count++;
        counts.put(word, count);
        try {
            BufferedWriter br = new BufferedWriter(new FileWriter(new File("output"), true));
            br.append(word + ":" + count + "\n");
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        log.info("Word:" + word +" Count:" +count);
        // insertIntoMongoDB(word, count);
        log.info("WordStore:" + word +" CountStore:" +count);
        basicOutputCollector.emit(new Values(word, count));

    }

    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {
```

- Split Senetence Bolt Program

```
import ...

/**
 * Created by Mayanka on 16-Sep-15.
 */
public class SplitSentenceBolt extends BaseBasicBolt {

    public void execute(Tuple tuple, BasicOutputCollector basicOutputCollector) {
        String s= tuple.getString(0);
        String a[]=s.split(" ");
        for (int i=0;i<a.length;i++)
        {
            basicOutputCollector.emit(new Values(a[i]));
        }
    }

    public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {
        outputFieldsDeclarer.declare(new Fields("words"));
    }
}
```

- Storm Kafka Producer

```
 */
public class StormKafkaProducer {
    static Logger log;
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        BasicConfigurator.configure();
      log = LoggerFactory.getLogger(StormKafkaProducer.class);

        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(
                    "Kafka_Bolt_Test",
                    createConfig(true),
                    createTopology());
            try {
                Thread.sleep(60000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            cluster.shutdown();
```

Spark and Kafka Workflow

Kafka is a (pub-sub) messaging and integration platform for Spark streaming. Kafka act as the central hub for real-time streams of data and are processed using complex algorithms in Spark Streaming. After data is processed, Spark Streaming could be publishing results into another Kafka topic or store in HDFS, databases or dashboards.

Working on Spark and Kafka integration.

Workflow includes:

Kafka Producer→ Spark Streaming → Kafka Consumer

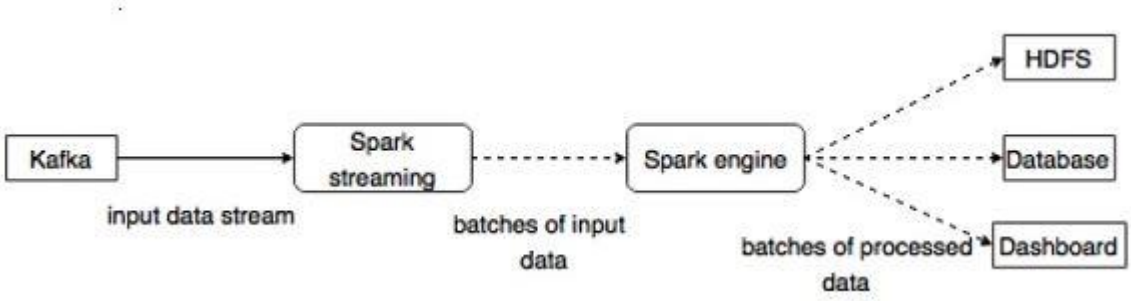


Fig1: Kafka Producer Integration with spark engine

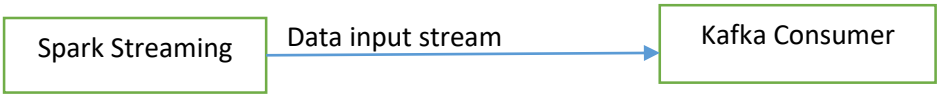


Fig 2: Spark Streaming data to Kafka Consumer

Spark and Kafka Integration

- Kafka Producer Code

```
// Produces some random words between 1 and 100.
object KafkaWordCountProducer {

  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: KafkaWordCountProducer <metadataBrokerList> <topic> " +
        "<messagesPerSec> <wordsPerMessage>")
      System.exit(1)
    }

    val Array(brokers, topic, messagesPerSec, wordsPerMessage) = args

    // Zookeeper connection properties
    val props = new HashMap[String, Object]()
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, brokers)
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
      "org.apache.kafka.common.serialization.StringSerializer")
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
      "org.apache.kafka.common.serialization.StringSerializer")

    val producer = new KafkaProducer[String, String](props)

    // Send some messages
    while(true) {
      (1 to messagesPerSec.toInt).foreach { messageNum =>
        val str = (1 to wordsPerMessage.toInt).map(x => scala.util.Random.nextInt(10).toString)
          .mkString(" ")

        val message = new ProducerRecord[String, String](topic, null, str)
        producer.send(message)
      }

      Thread.sleep(1000)
    }
  }

}
```

- Spark Kafka Word Count

```
 *      my-consumer-group topic1,topic2 1`
 */
object KafkaWordCount {
  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: KafkaWordCount <zkQuorum> <group> <topics> <numThreads>")
      System.exit(1)
    }

    StreamingExamples.setStreamingLogLevels()

    val Array(zkQuorum, group, topics, numThreads) = args
    val sparkConf = new SparkConf().setAppName("KafkaWordCount")
    val ssc = new StreamingContext(sparkConf, Seconds(2))
    ssc.checkpoint("checkpoint")

    val topicMap = topics.split(",").map((_, numThreads.toInt)).toMap
    val lines = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap).map(_._2)
    val words = lines.flatMap(_.split(" "))
    val wordCounts = words.map(x => (x, 1L))
      .reduceByKeyAndWindow(_ + _, _ - _, Minutes(10), Seconds(2), 2)
    wordCounts.print()

    ssc.start()
    ssc.awaitTermination()
  }
}

// Produces some random words between 1 and 100.
object KafkaWordCountProducer {

  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: KafkaWordCountProducer <metadataBrokerList> <topic> " +
        "<messagesPerSec> <wordsPerMessage>")
      System.exit(1)
    }
```

- Kafka Consumer

```
// Produces some random words between 1 and 100.
object KafkaWordCountProducer {

  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: KafkaWordCountProducer <metadataBrokerList> <topic> " +
        "<messagesPerSec> <wordsPerMessage>")
      System.exit(1)
    }

    val Array(brokers, topic, messagesPerSec, wordsPerMessage) = args

    // Zookeeper connection properties
    val props = new HashMap[String, Object]()
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, brokers)
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
      "org.apache.kafka.common.serialization.StringSerializer")
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
      "org.apache.kafka.common.serialization.StringSerializer")

    val producer = new KafkaProducer[String, String](props)

    // Send some messages
    while(true) {
      (1 to messagesPerSec.toInt).foreach { messageNum =>
        val str = (1 to wordsPerMessage.toInt).map(x => scala.util.Random.nextInt(10).toString)
          .mkString(" ")

        val message = new ProducerRecord[String, String](topic, null, str)
        producer.send(message)
      }

      Thread.sleep(1000)
    }
  }

}
```

- Kafka Output

```
bytes result sent to driver
15/10/08 09:21:50 INFO TaskSetManager: Finished task 0.0 in stage 137.0 (TID 181
in 41 ms on localhost (1/1)
15/10/08 09:21:50 INFO TaskSchedulerImpl: Removed TaskSet 137.0, whose tasks hav
all completed, from pool
15/10/08 09:21:50 INFO DAGScheduler: Stage 137 (print at KafkaWordCount.scala:27
finished in 0.037 s
-------------------------------------------
Time: 1444321310000 ms
-------------------------------------------
(4,307359)
(8,307165)
(6,307435)
(0,306762)
(2,305377)
(7,306598)
(5,306727)
(9,307420)
(3,306981)
(1,306376)

15/10/08 09:21:50 INFO DAGScheduler: Job 35 finished: print at KafkaWordCount.sc
a:27, took 0.054576 s
15/10/08 09:21:50 INFO JobScheduler: Finished job streaming job 1444321310000 ms
```

Data Sources

Cancer Imaging Archive

http://www.cancerimagingarchive.net/

TCIA Collections

The image data in The Cancer Imaging Archive (TCIA) is organized into purpose-built collections of subjects. The subjects typically have a cancer type and/or anatomical site (lung, brain, etc.) in common. Each link in the table below contains information concerning the scientific value of a collection, information about how to obtain any supporting non-image data which may be available, and links to view or download the imaging data.

RIDER NEURO MRI

RIDER Neuro MRI contains imaging data on 19 patients with recurrent glioblastoma who underwent repeat imaging sets. These images were obtained approximately 2 days apart. All 19 patients had repeat dynamic contrast-enhanced MRI (DCE-MRI) datasets on the same 1.5T imaging magnet. On the basis of T2-weighted images, technologists chose 16 image locations using 5mm thick contiguous slices for the imaging. For T1 mapping, multi-flip 3D FLASH images were obtained using flip angles of 5, 10, 15, 20, 25 and 30 degrees, TR of 4.43 ms, TE of 2.1 ms, 2 signal averages. Dynamic images were obtained during the intravenous injection of 0.1mmol/kg of Magnevist intravenous at 3ccs/second, started 24 seconds after the scan had begun.

RIDER Lung CT

The RIDER Lung CT collection was constructed as part of a study to evaluate the variability of tumor unidimensional, bidimensional, and volumetric measurements on same-day repeat computed tomographic (CT) scans in patients with non–small cell lung cancer.

Thirty-two patients with non–small cell lung cancer, each of whom underwent two CT scans of the chest within 15 minutes by using the same imaging protocol, were included in this study. Three radiologists independently measured the two greatest diameters of each lesion on both scans and, during another session, measured the same tumors on the first scan.

Lung Phantom

The FDA anthropomorphic thorax phantom with 12 phantom lesions of different sizes (10 and 20 mm in effective diameter), shapes (spherical, elliptical, lobulated, and spiculated), and densities (−630,−10, and +100 HU) was scanned at Columbia University Medical Center on a 64-detector row scanner (LightSpeed VCT, GE Healthcare, Milwaukee, WI). The CT scanning parameters were 120 kVp, 100 mAs, 64x0.625 collimation, and pitch of 1.375. The images were reconstructed with the lung kernel using 1.25 mm slice thickness.

Analytical Tools

- Apache Storm
- Apache Kafka
- Apache Spark
- Machine Learning
- REST API
- WEKA

Expected Input: Streaming MR images in .mkv format

Expected Output: Application displaying results with Annotation

Related Work

Open Source Projects

Brain Tumor Image Segmentation benchmark

Results of Brain Tumor Segmentation Benchmark were studies. MR Scan data of high and low grade glioma patients were studied and compared up to 65 scans using Tumor Image simulation software. BRATS image data and manual annotations are available publically. Below are the results and algorithms which were followed.

Algorithms Followed

Algorithm	MRI modalities	Approach	Perform. score	Tumor type	trainining/testing (tt/cv)
Fletcher 2001	T_1 T_2 PD	Fuzzy clustering w/ image retrieval	Match (53-91%)	na	2/4 tt
Kaus 2001	T_1	Template-moderated classification	Accuracy (95%)	LG, M	10/10 tt
Ho 2002	T_1 T_1c	Level-sets w/ region competition	Jaccard (85-93%)	G, M	na/5 tt
Prastawa 2004	T_2	Generative model w/ outlier detection	Jaccard (59-89%)	G, M	na/3 tt
Corso 2008	T_1 T_1c T_2 FLAIR	Weighted aggregation	Jaccard (62-69%)	HG	10/10 tt
Verma 2008	T_1 T_1c T_2 FLAIR DTI	SVM	Accuracy (34-93%)	HG	14/14 cv
Wels 2008	T_1 T_{1c} T_2	Discriminative model w/ CRF	Jaccard (78%)	G	6/6 cv
Cobzas 2009	T_{1c} FLAIR	Level-set w/ CRF	Jaccard (50-75%)	G	6/6 tt
Wang 2009	T_1	Fluid vector flow	Tanimoto (60%)	na	0/10 tt
Menze 2010	T_1 T_1c T_2 FLAIR	Generative model w/ lesion class	Dice (40-70%)	G	25/25 cv
Bauer 2011	T_1 T_{1c} T_2 FLAIR	Hierarchical SVM w/ CRF	Dice (77-84%)	G	10/10 cv

Table: Represents Algorithm, MRI modalities, approach followed with Performance score

BRATS (Brain Tumor Image Segmentation benchmark)

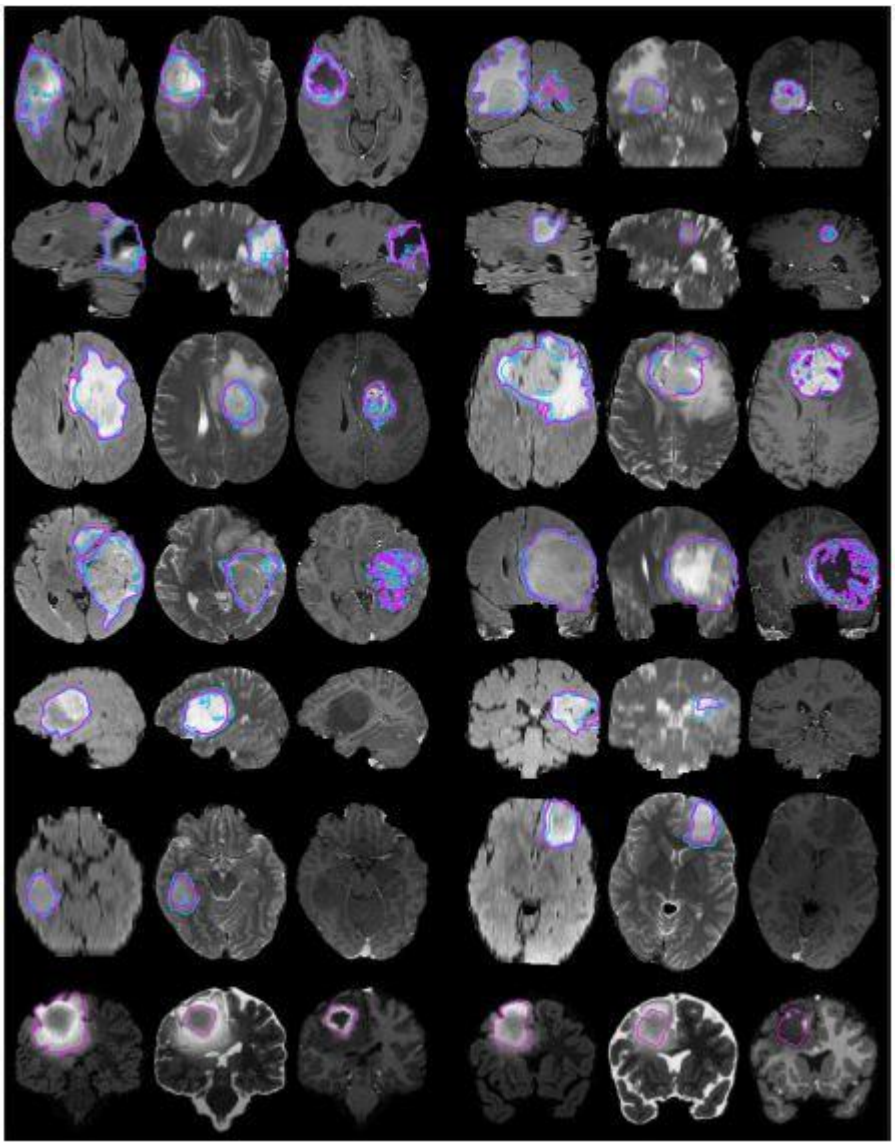


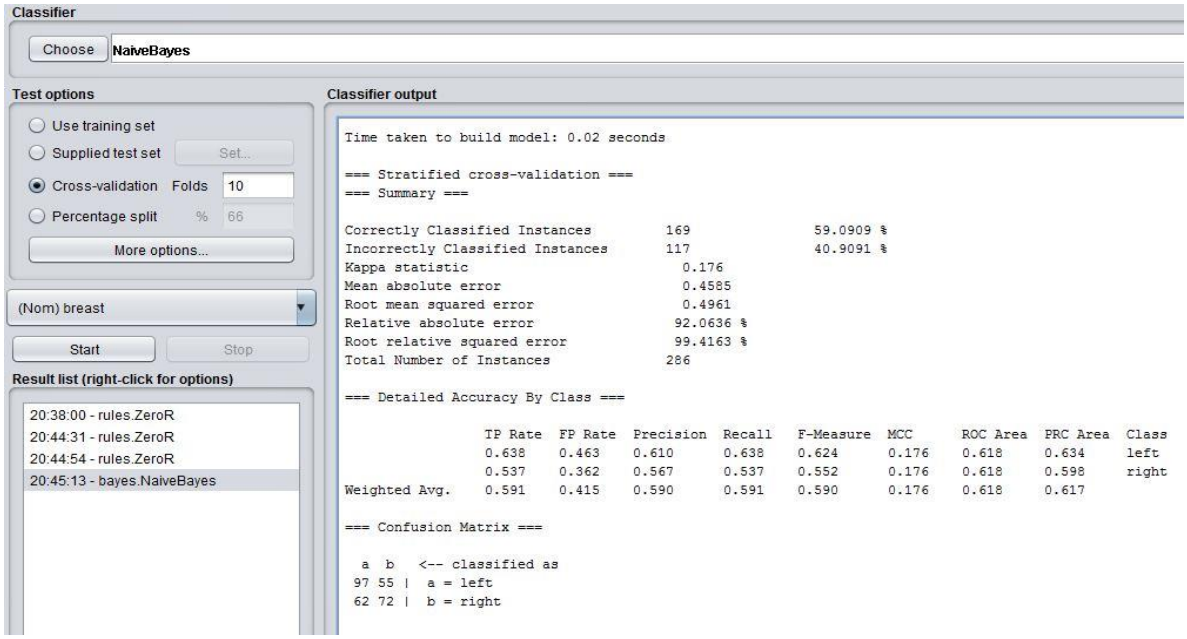
Figure: Examples from BRATS training data with Tumor Region and annotations of Individual Experts and Consensus segmentation

Related Literatures

Breast Cancer data was analyzed using Weka tool. Clustering and knowledge flow and classification data was based on different Machine Learning Algorithms.

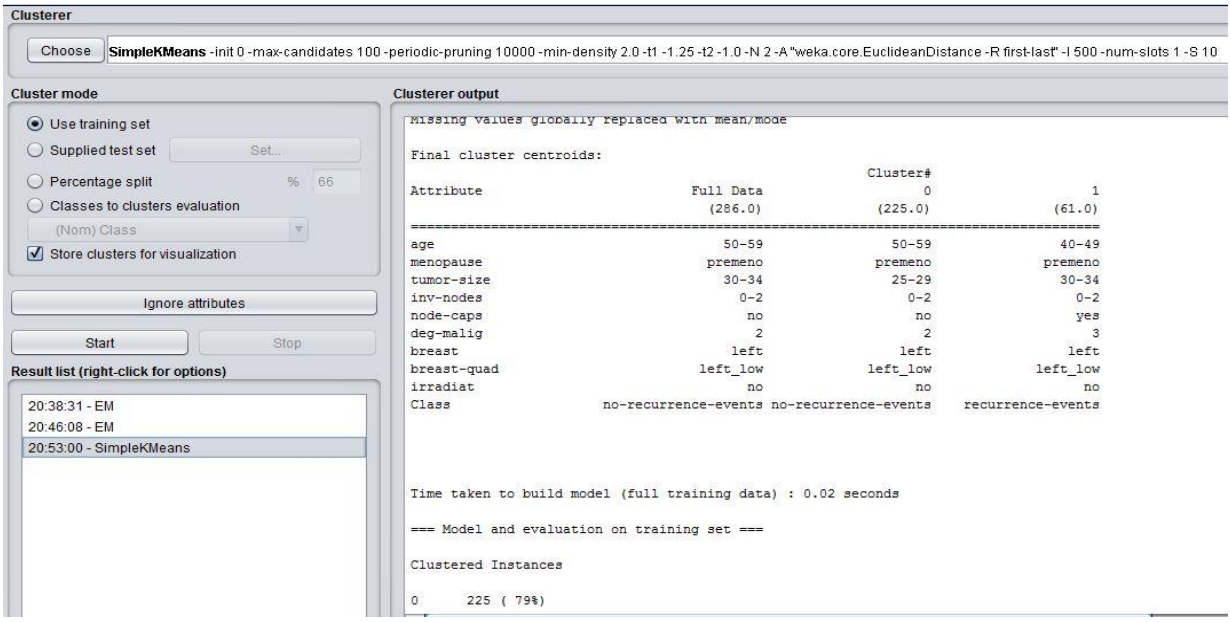
- ### Naïve Bayes

  Accuracy: 59%



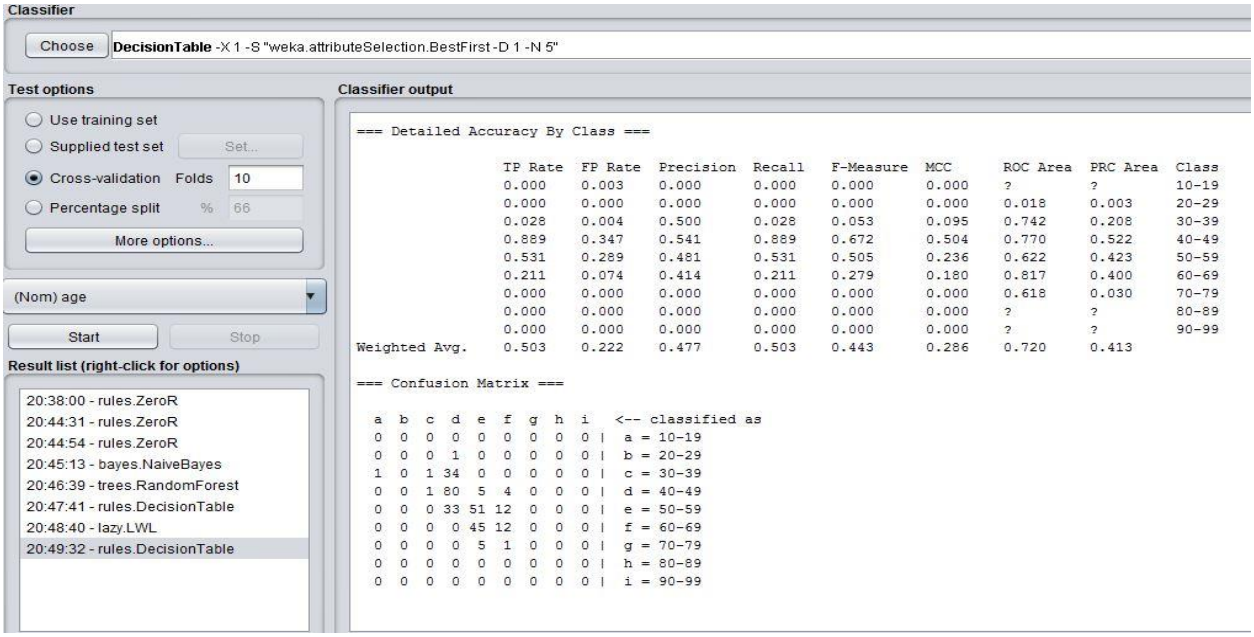
- ### K-Means

  Accuracy: 63%



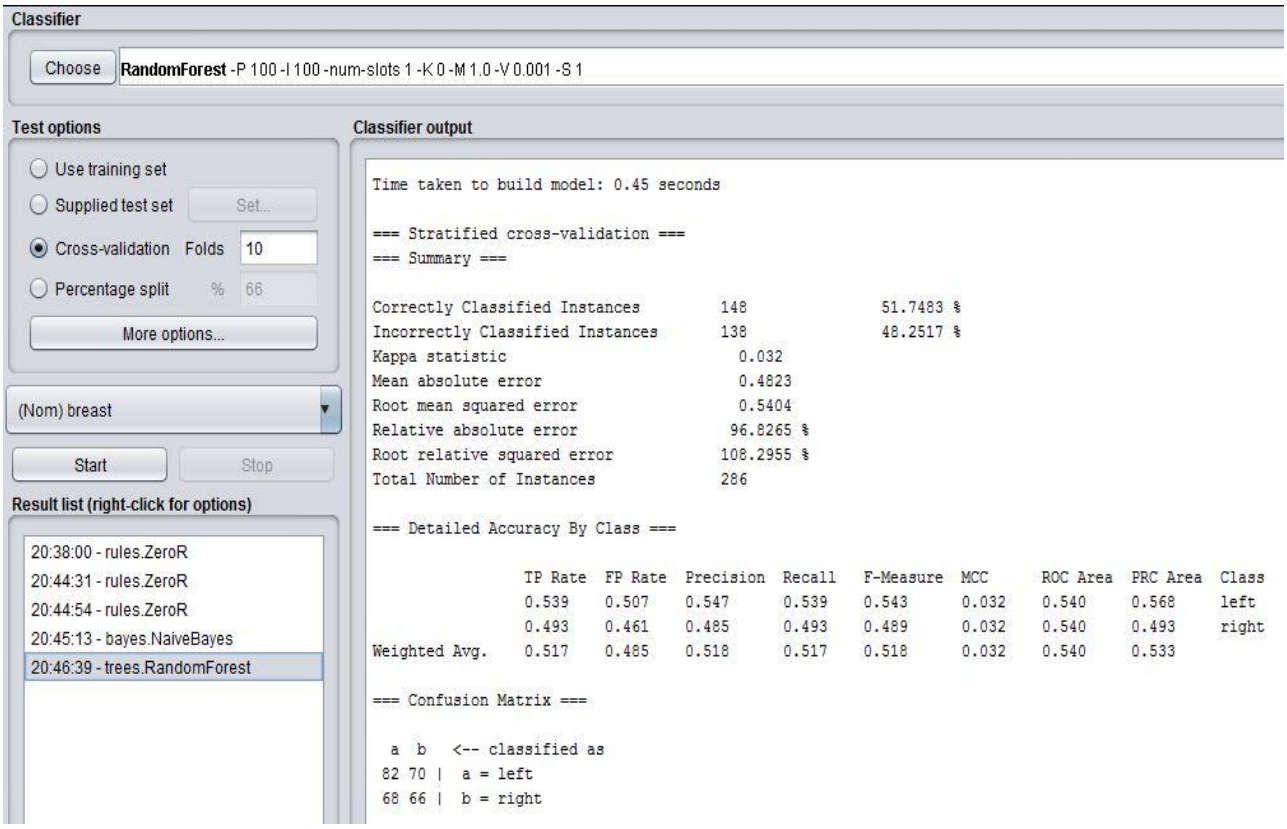
- ## Decision Tree

  Accuracy: 88%



- ## Random Forest

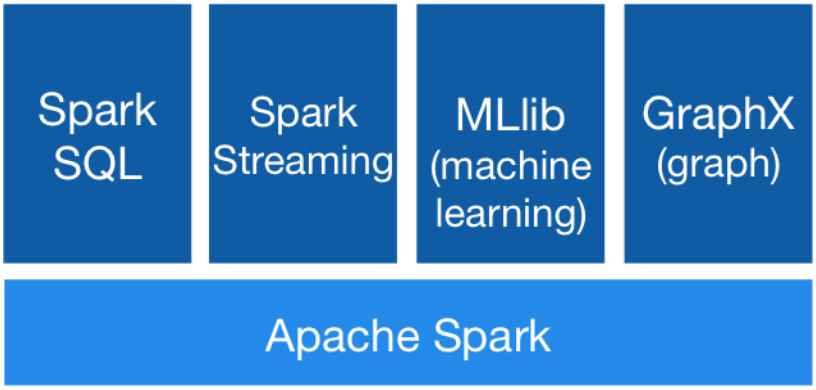
  Accuracy: 52%



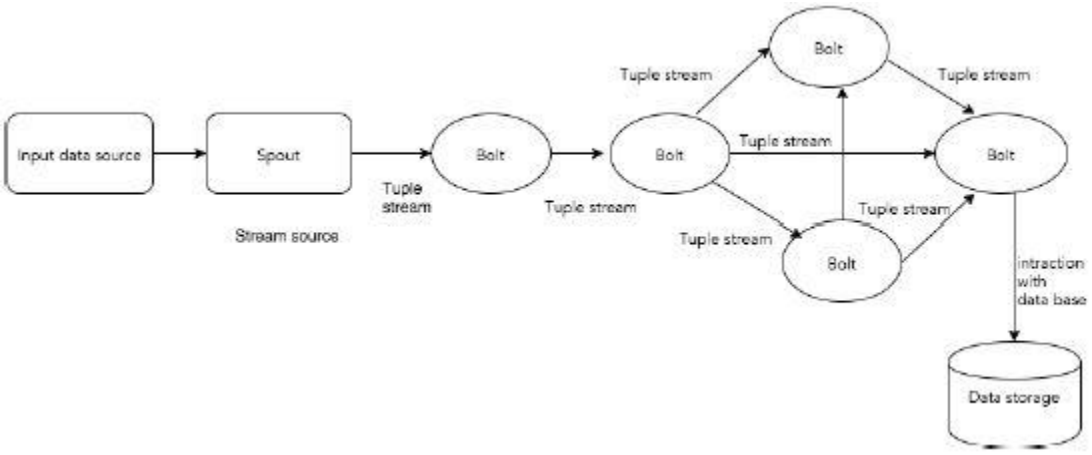
Application Specifications

System Specifications

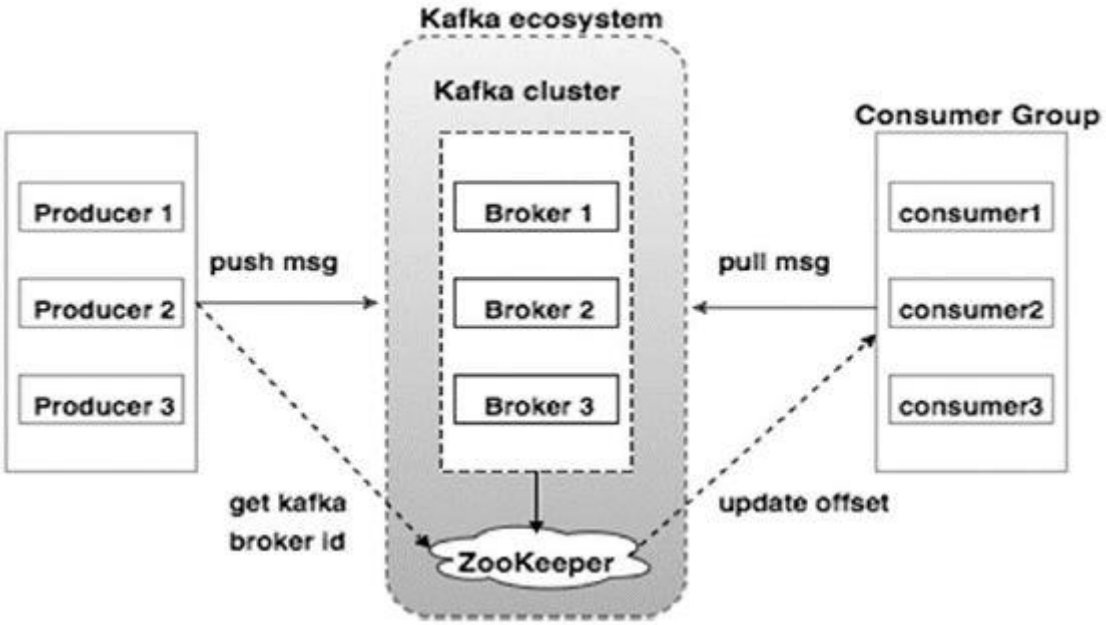
Spark Architecture



Storm Architecture



Kafka Architecture



Design

Architecture Diagram

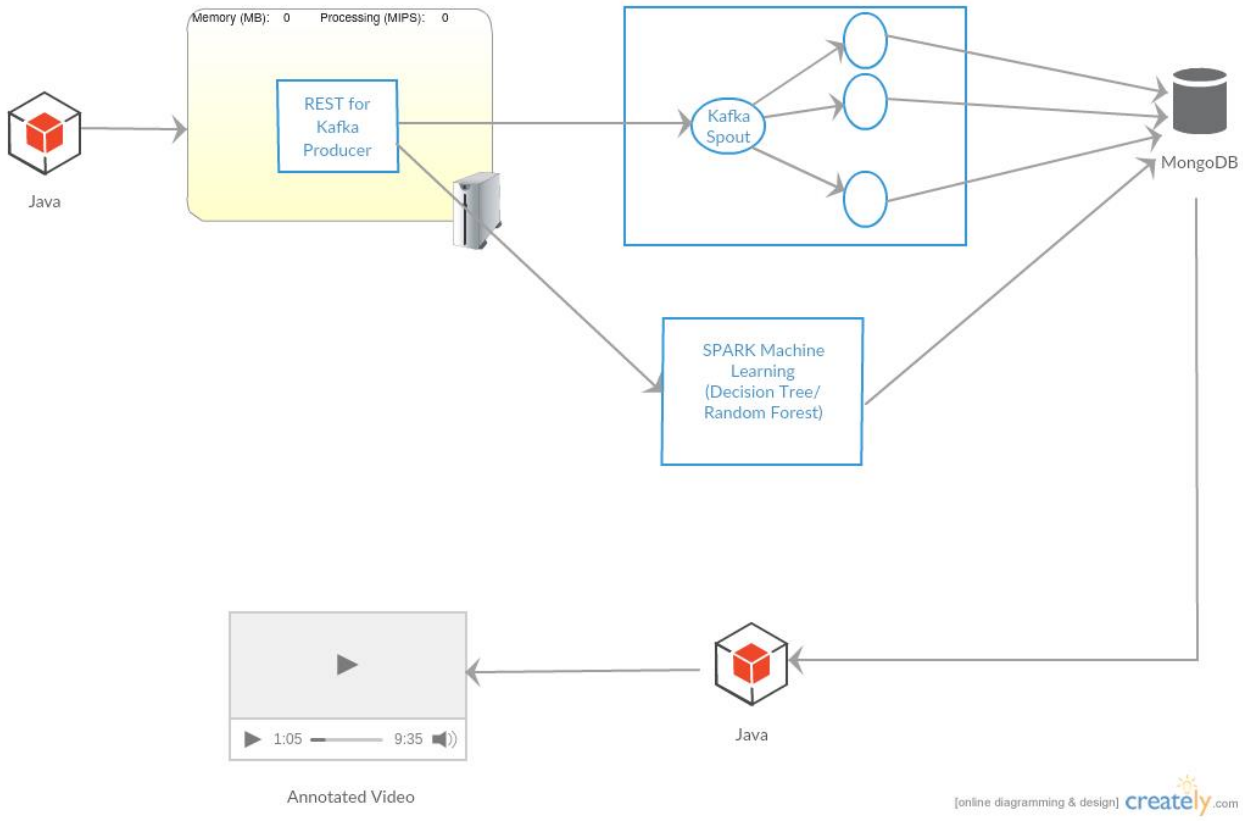


Figure: Architecture Diagram

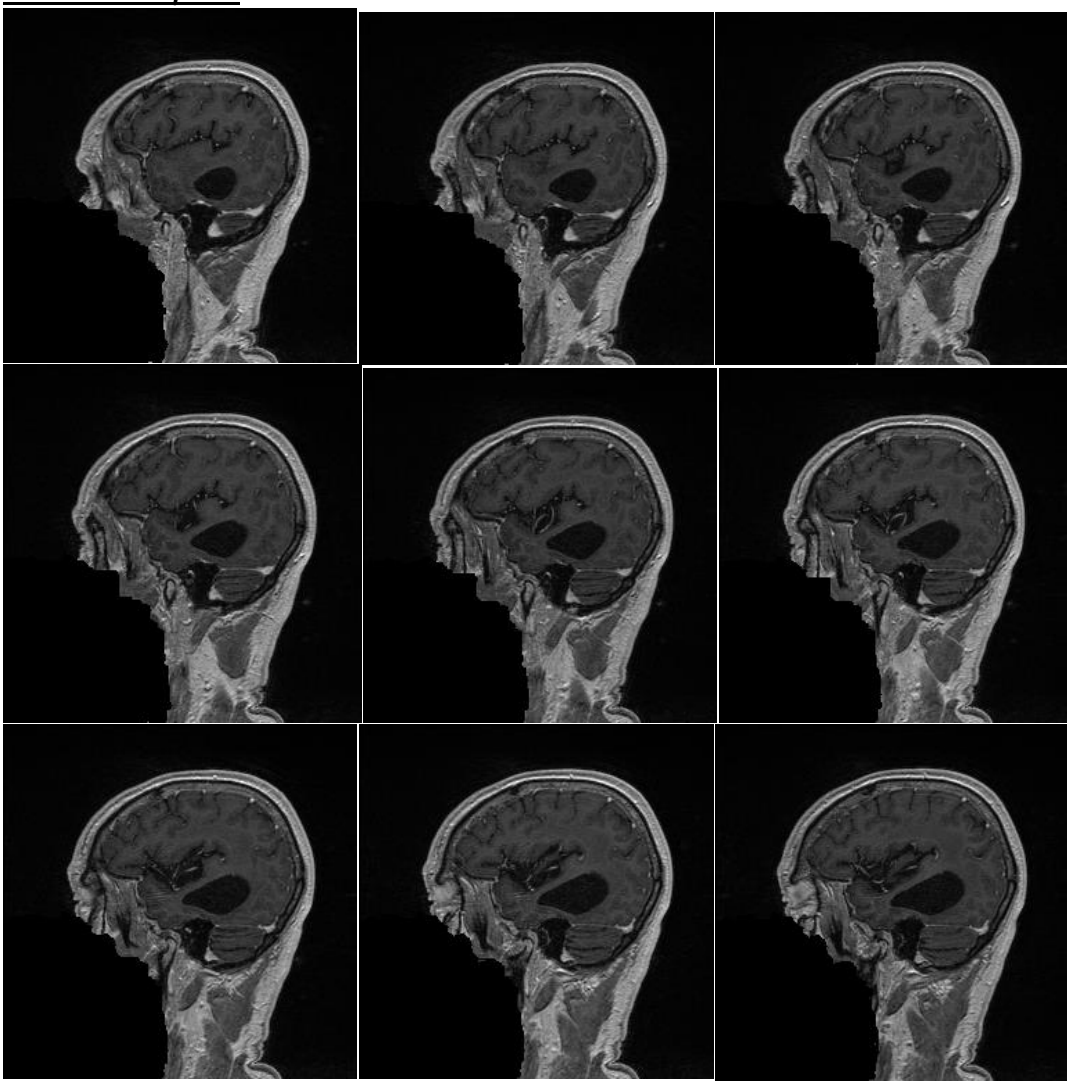
Data Samples

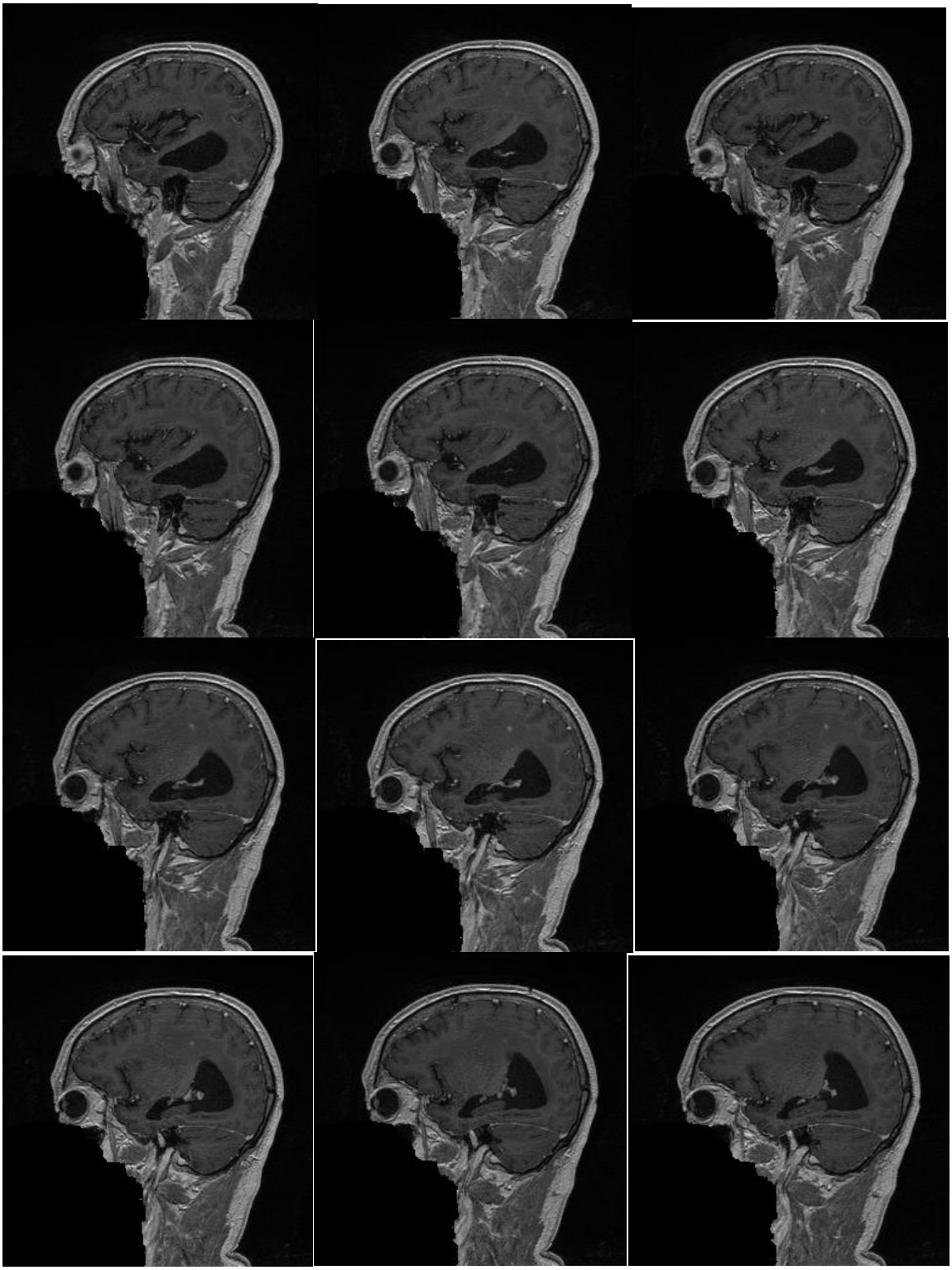
Domain: Cancer

Types:

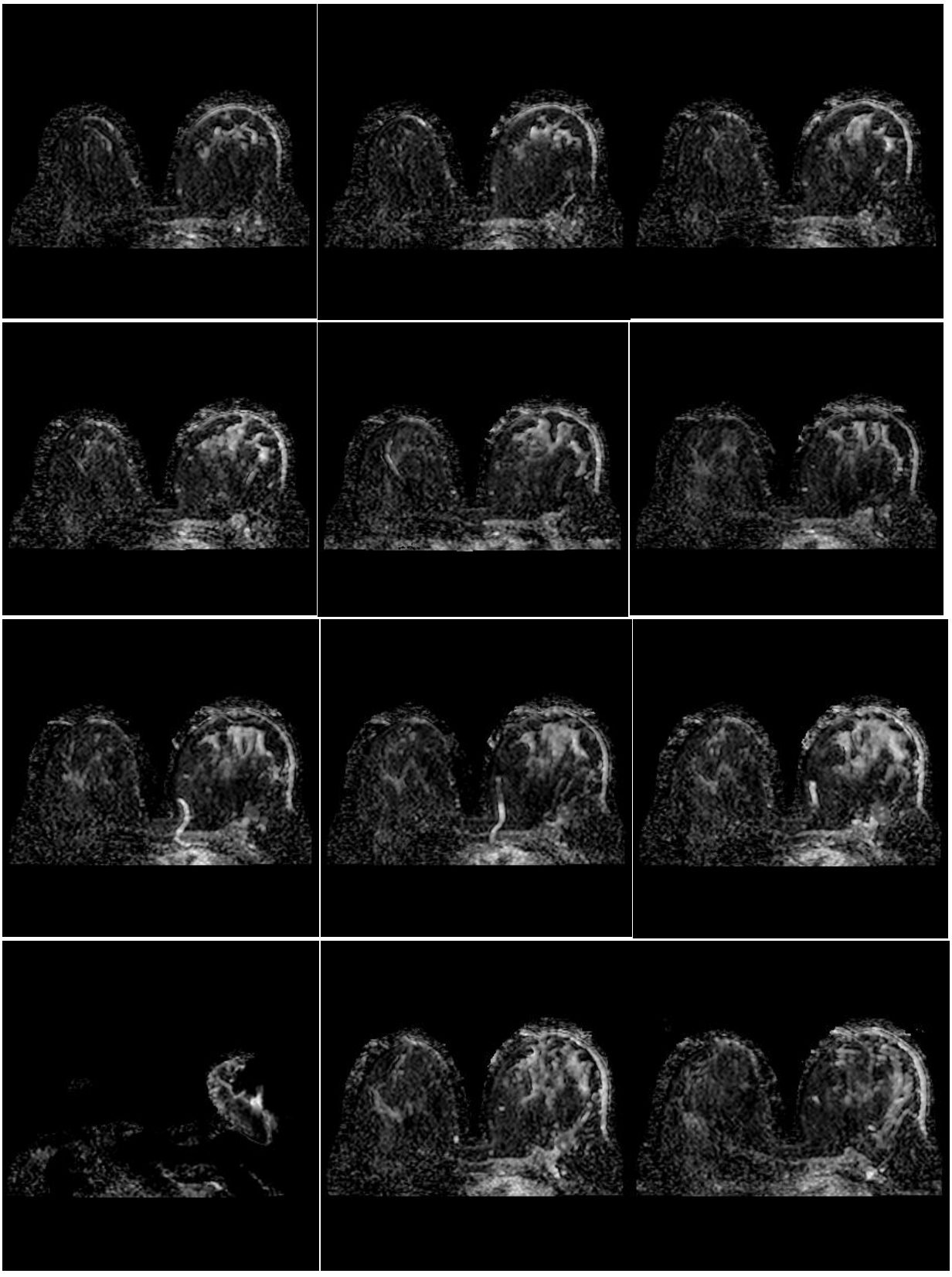
- Brain Cancer
- Lung Cancer
- Breast Cancer
- Prostate Cancer

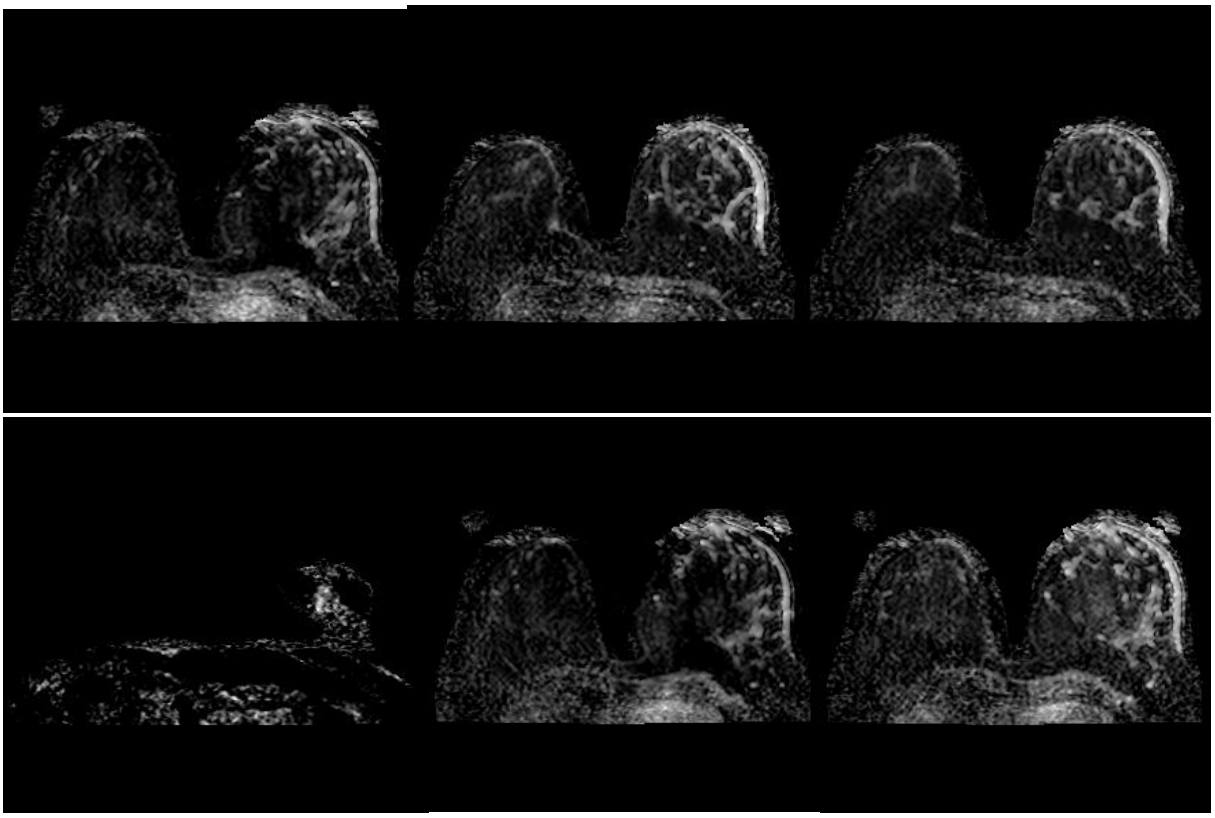
Brain Samples



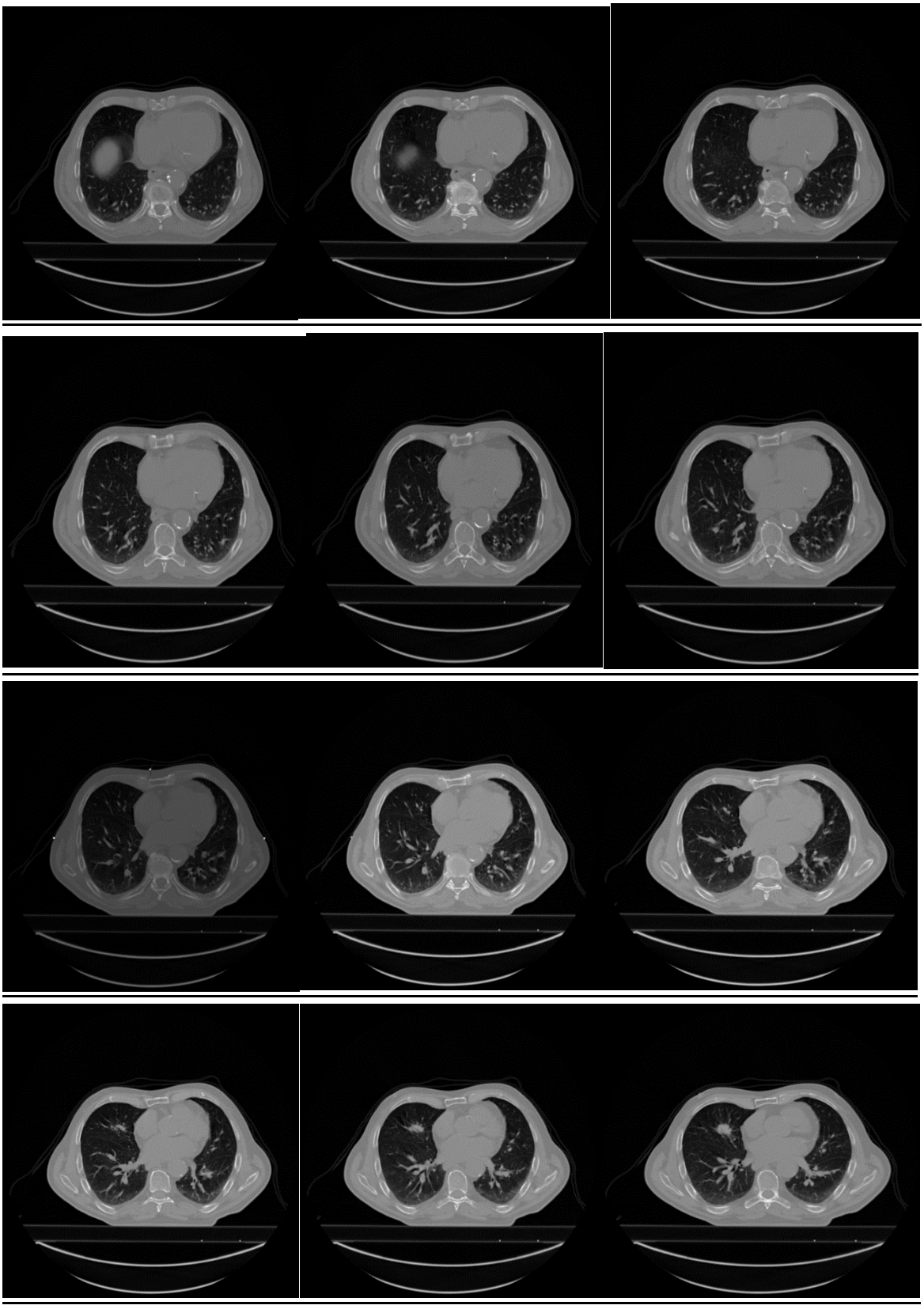


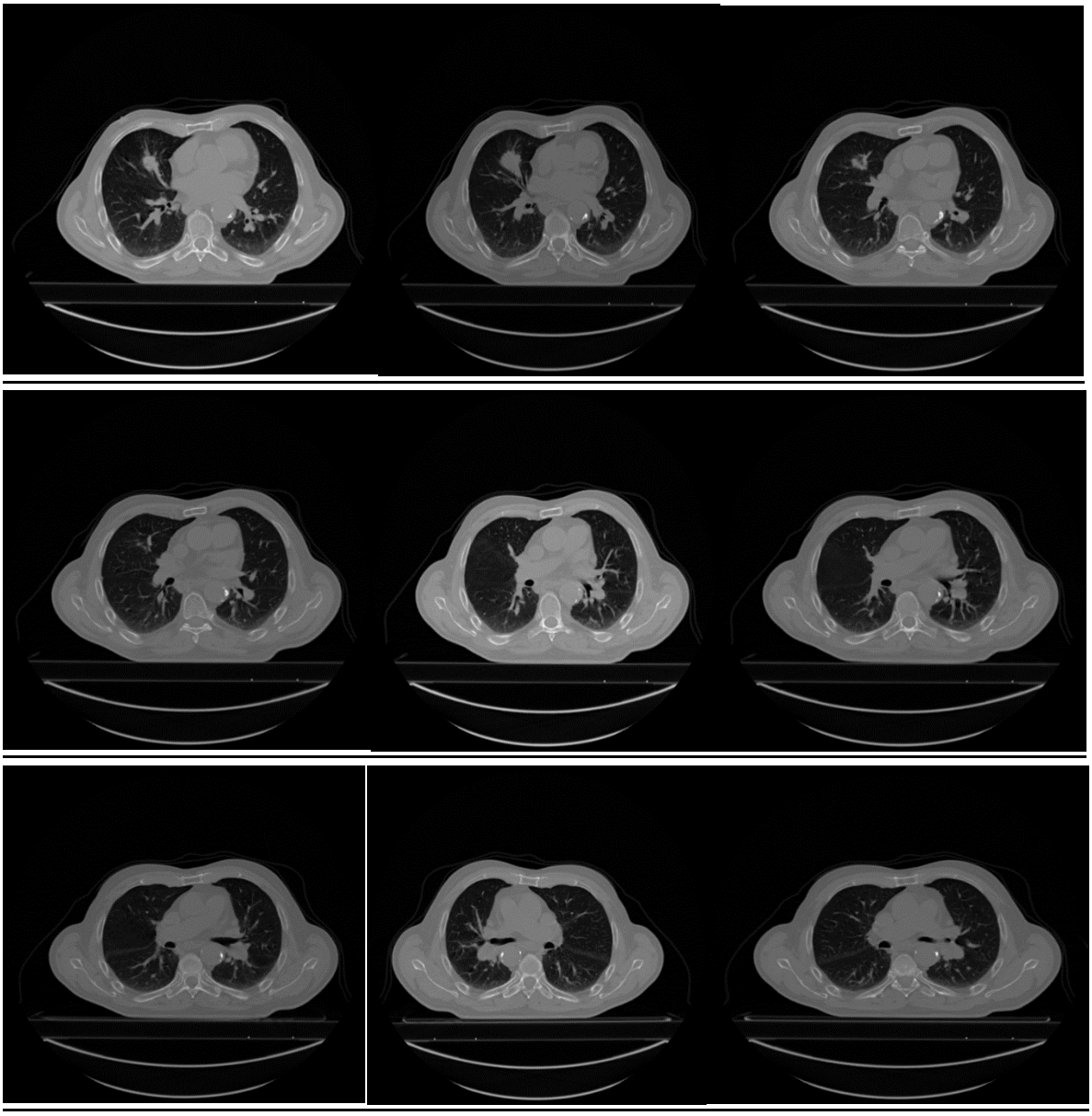
Breast Cancer





Lung Cancer





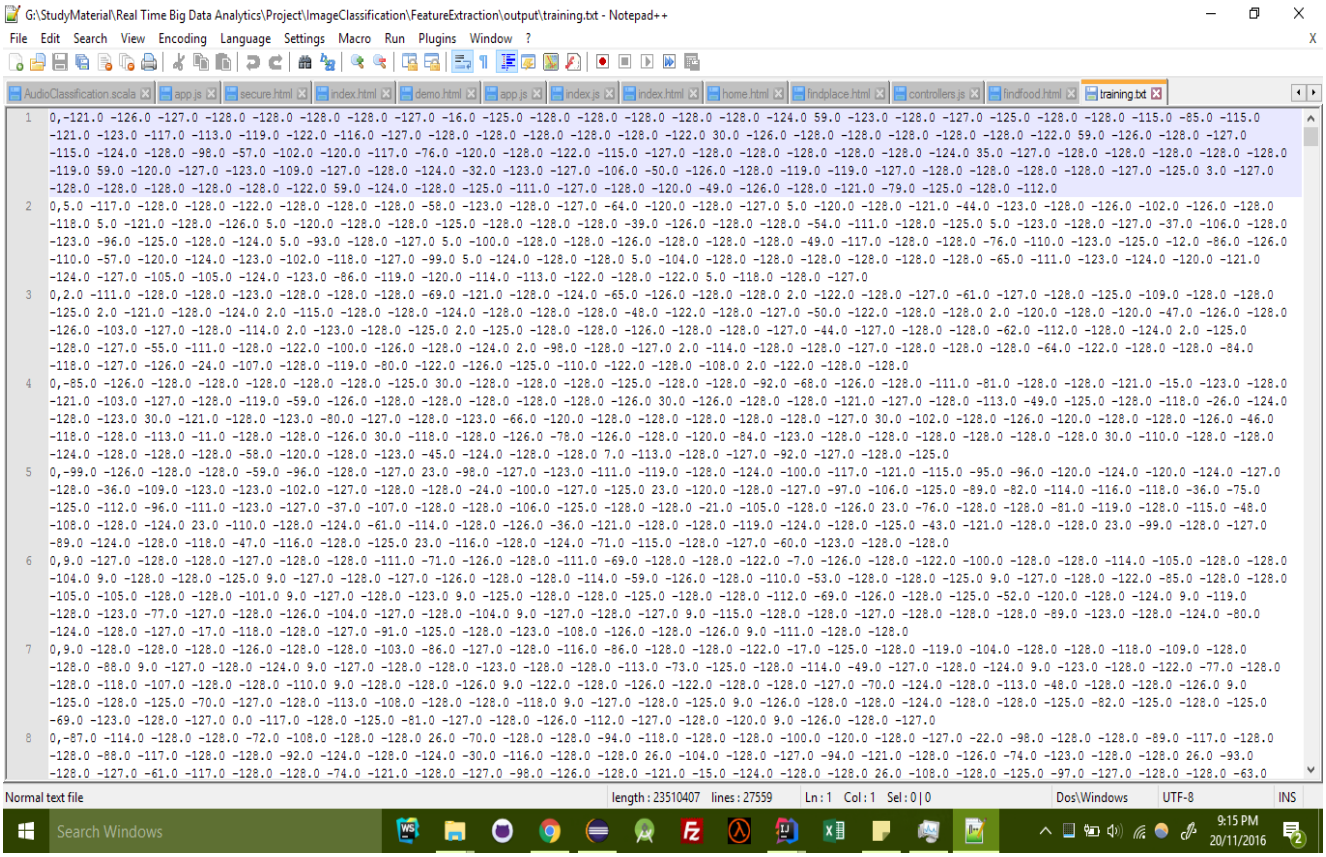
Project Structure

Training and testing data contains Brain, breast and lung cancer samples. There were 20 samples, training data contains 16 samples and testing contains 4 samples.

Increasing Training and testing data to increase accuracy of the data.

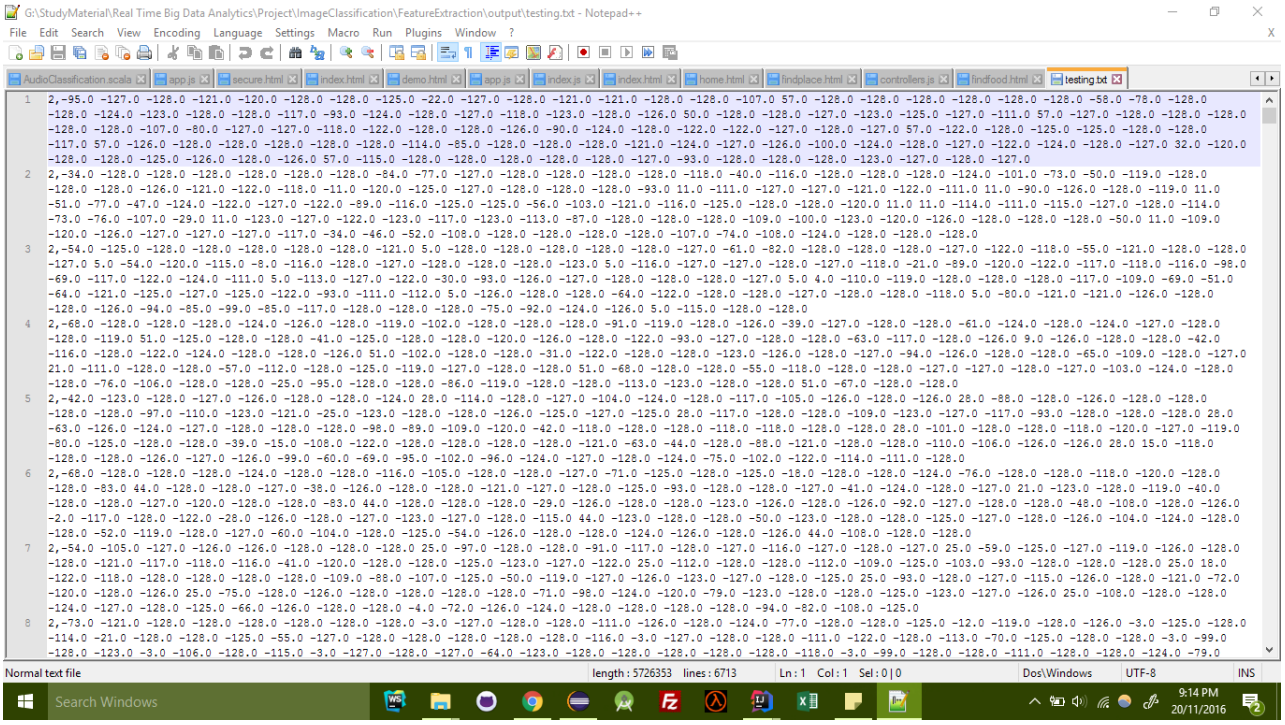
Training feature Extraction

Below is the feature extraction for the training data.



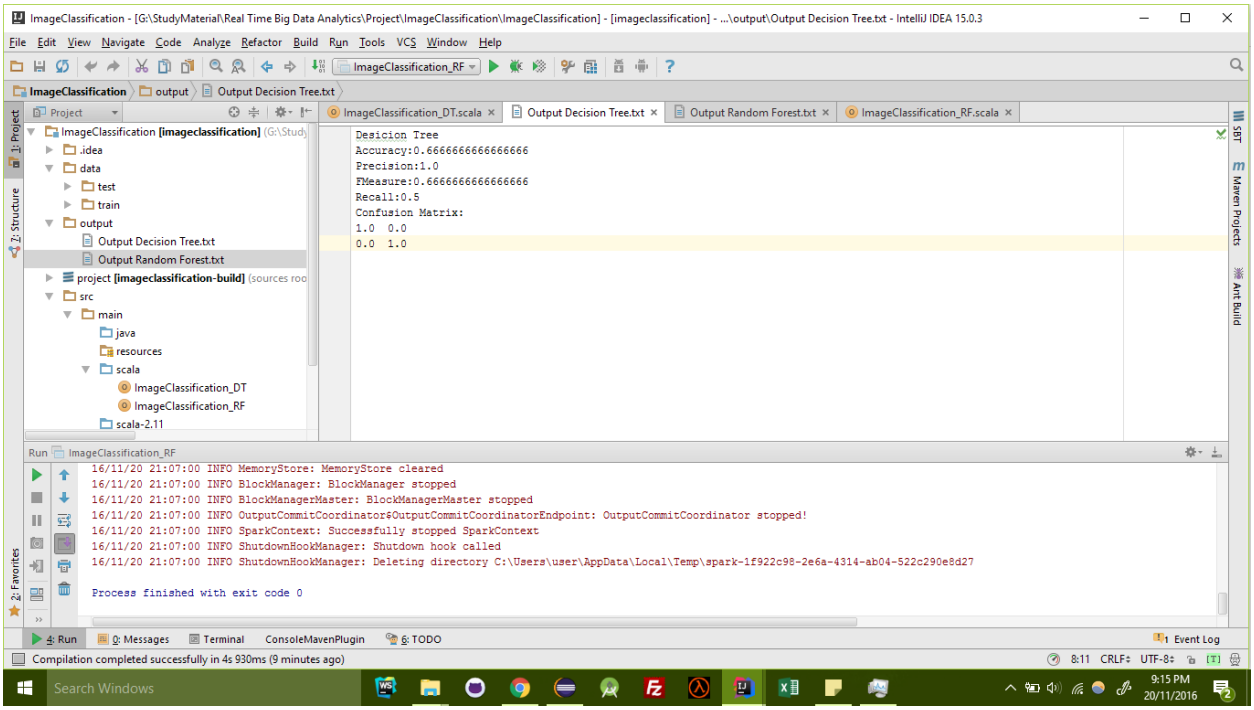
Testing Feature Extraction

Below is the feature extraction for the testing data



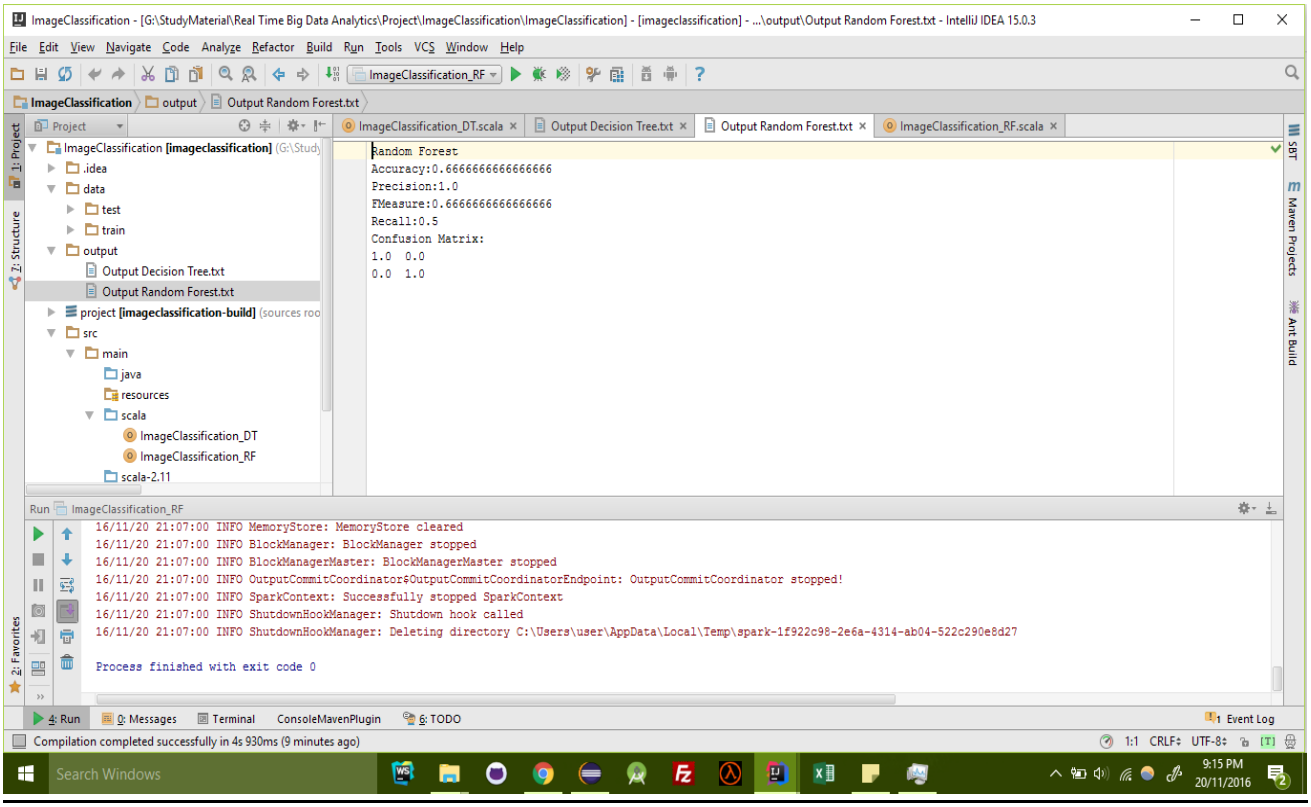
Decision Tree Accuracy

Accuracy Achieved: 66%



Random Forest Accuracy

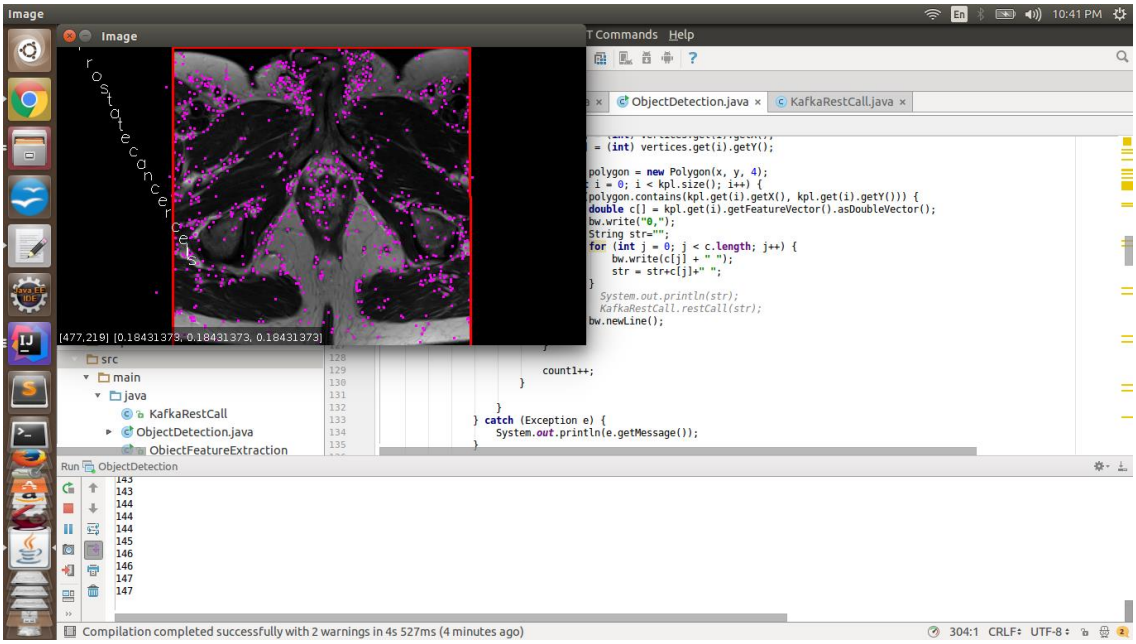
Accuracy Achieved: 66%



Implementation

Screenshots

Prostate Cancer, Brain, Breast and Brain Cancer with Storm/Spark/Kafka Connections



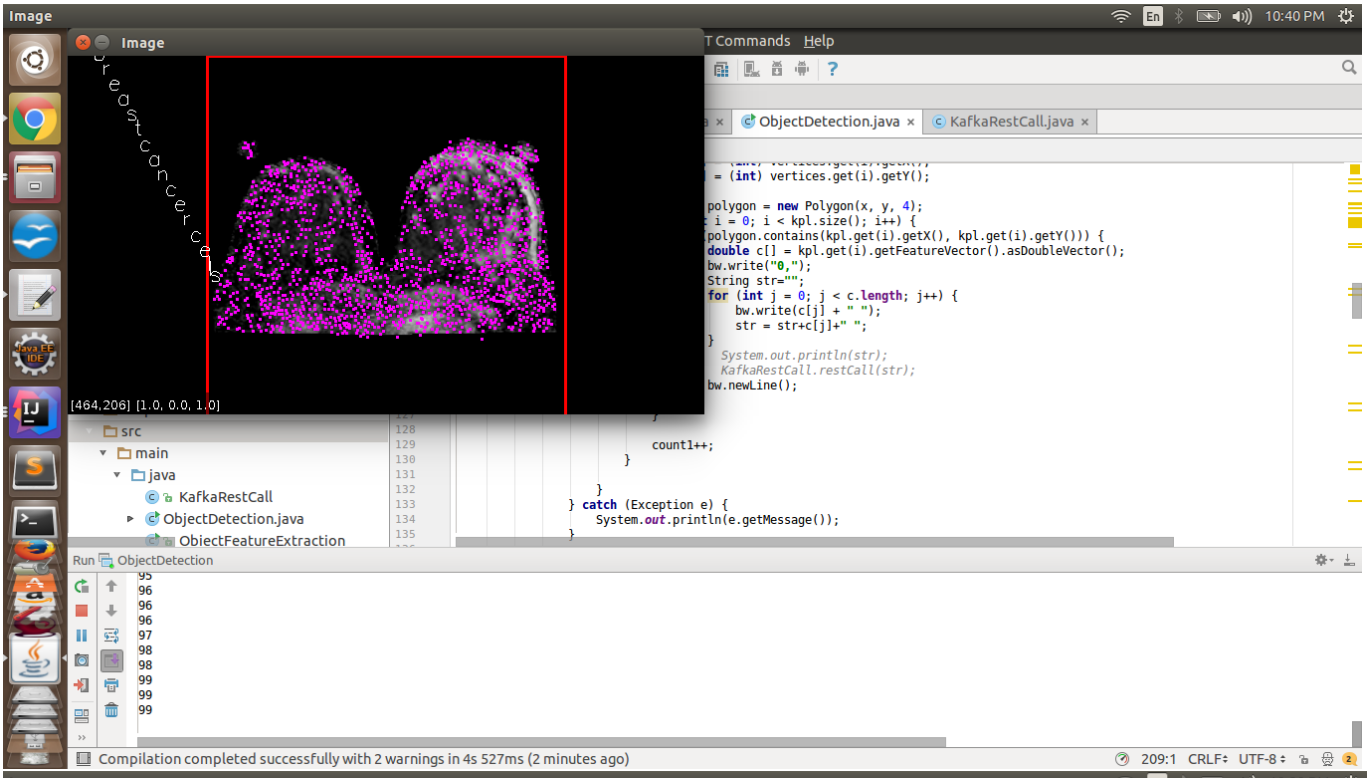


Image
10:40 PM
Image
Breast cancer cells
[464,206] [1.0, 0.0, 1.0]
T Commands Help
ObjectDetection.java
KafkaRestCall.java
= (int) vertices.get(i).getY();
polygon = new Polygon(x, y, 4);
i = 0; i < kpl.size(); i++) {
polygon.contains(kpl.get(i).getX(), kpl.get(i).getY())) {
double c[] = kpl.get(i).getFeatureVector().asDoubleVector();
bw.write("0,");
String str="";
for (int j = 0; j < c.length; j++) {
bw.write(c[j] + " ");
str = str+c[j]+" ";
}
System.out.println(str);
KafkaRestCall.restCall(str);
bw.newLine();
count1++;
} catch (Exception e) {
System.out.println(e.getMessage());
src
main
java
KafkaRestCall
ObjectDetection.java
ObjectFeatureExtraction
Run ObjectDetection
Compilation completed successfully with 2 warnings in 4s 527ms (2 minutes ago)
209:1 CRLF UTF-8

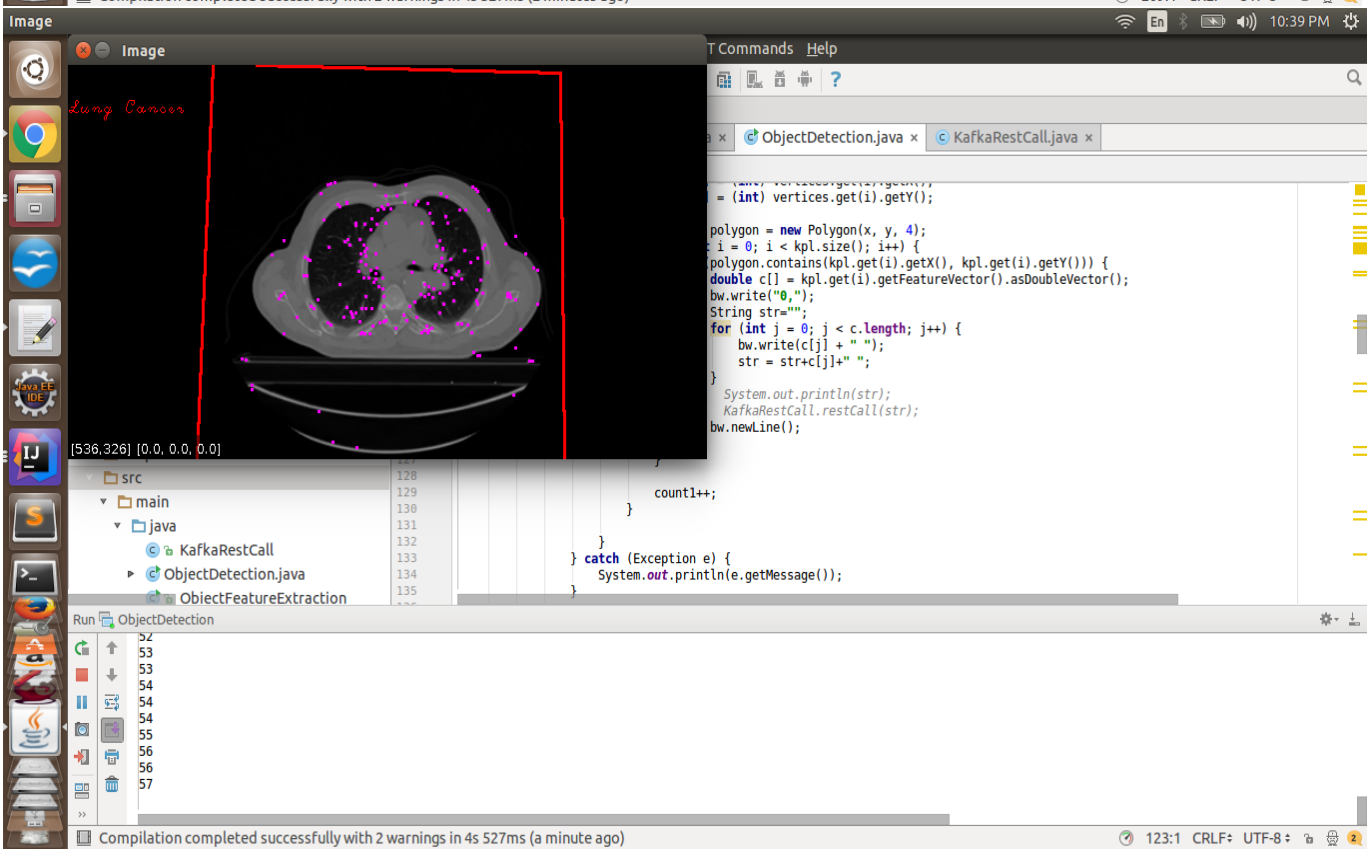


Image
10:39 PM
Image
Lung Cancer
[536,326] [0.0, 0.0, 0.0]
T Commands Help
ObjectDetection.java
KafkaRestCall.java
= (int) vertices.get(i).getY();
polygon = new Polygon(x, y, 4);
i = 0; i < kpl.size(); i++) {
polygon.contains(kpl.get(i).getX(), kpl.get(i).getY())) {
double c[] = kpl.get(i).getFeatureVector().asDoubleVector();
bw.write("0,");
String str="";
for (int j = 0; j < c.length; j++) {
bw.write(c[j] + " ");
str = str+c[j]+" ";
}
System.out.println(str);
KafkaRestCall.restCall(str);
bw.newLine();
count1++;
} catch (Exception e) {
System.out.println(e.getMessage());
src
main
java
KafkaRestCall
ObjectDetection.java
ObjectFeatureExtraction
Run ObjectDetection
Compilation completed successfully with 2 warnings in 4s 527ms (a minute ago)
123:1 CRLF UTF-8

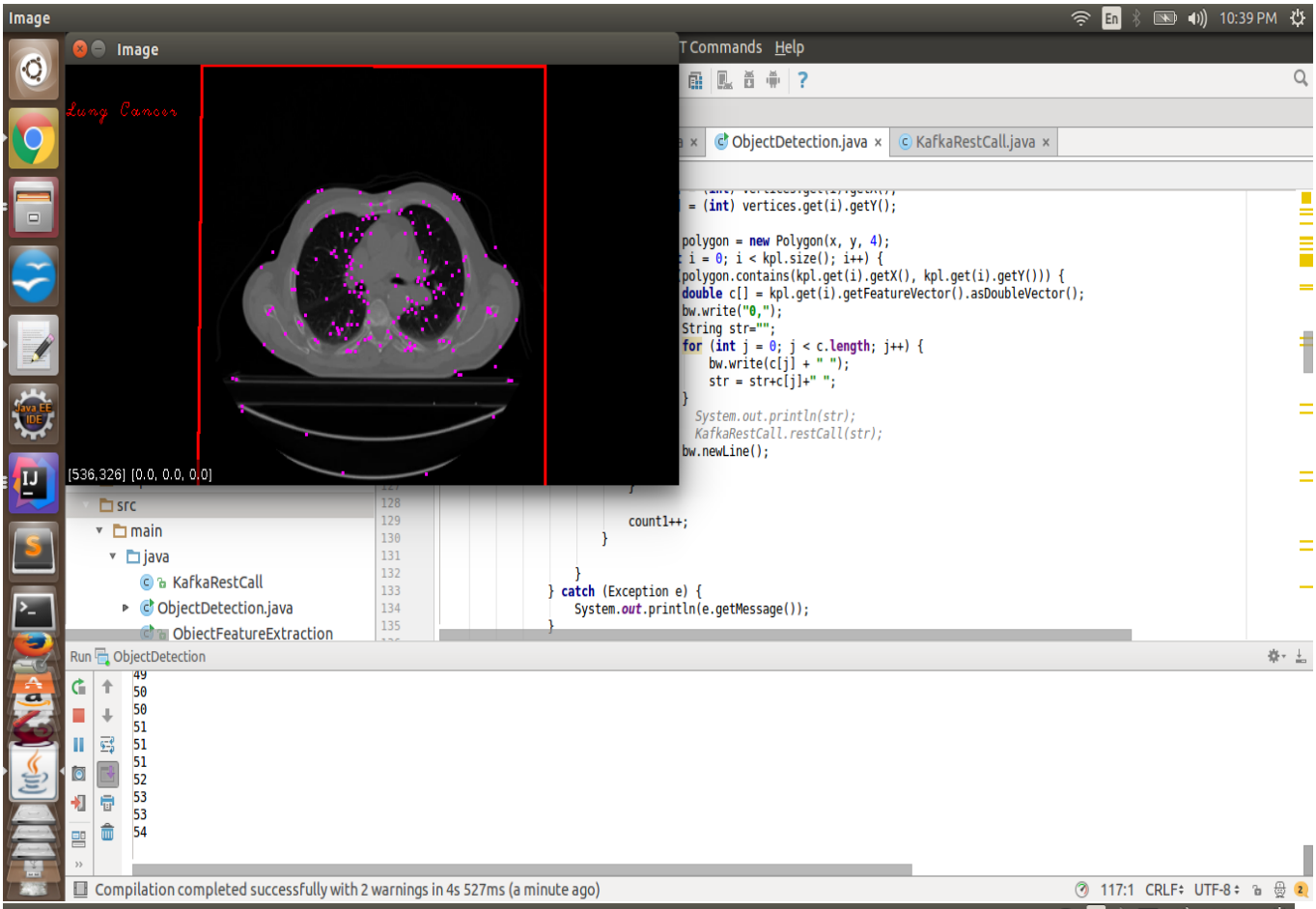


Image
Lung Cancer
[536,326] [0.0, 0.0, 0.0]
ObjectDetection.java
KafkaRestCall.java
Compilation completed successfully with 2 warnings in 4s 527ms (a minute ago)

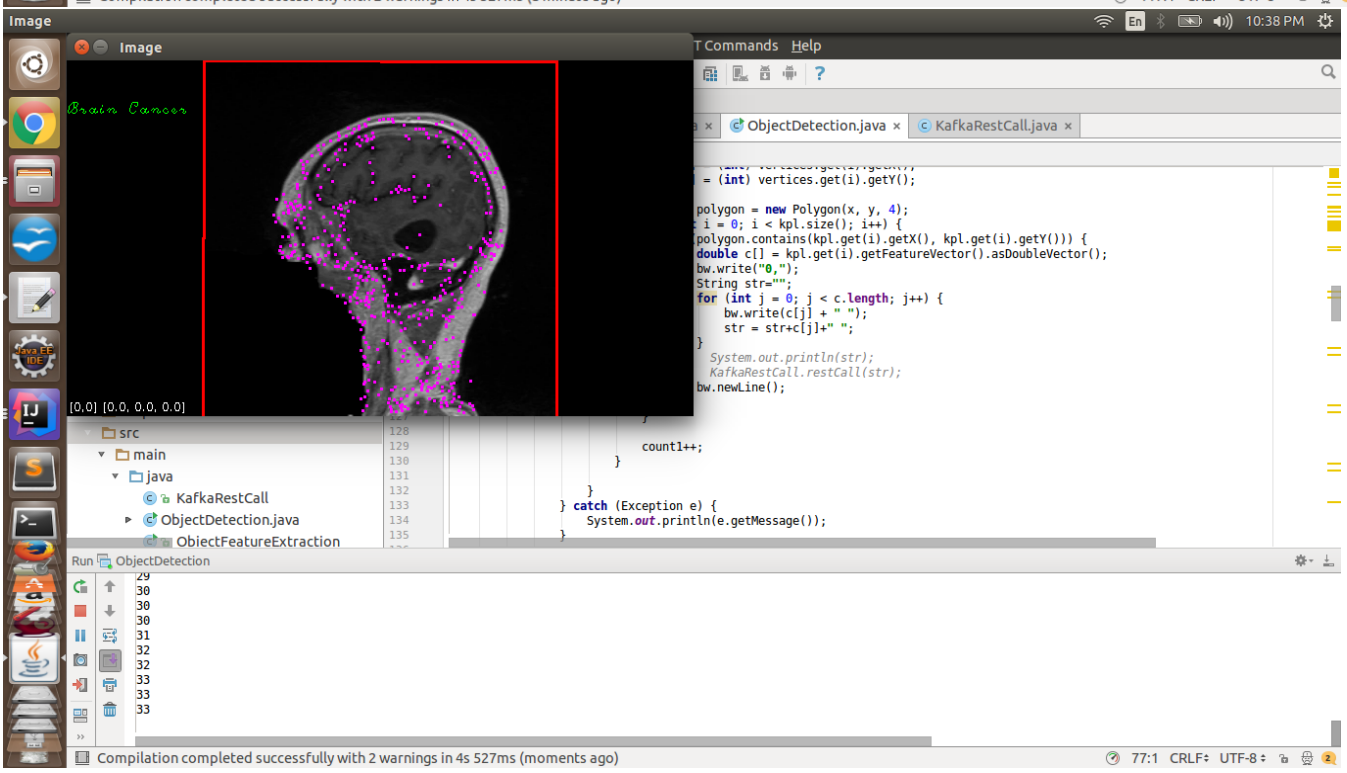
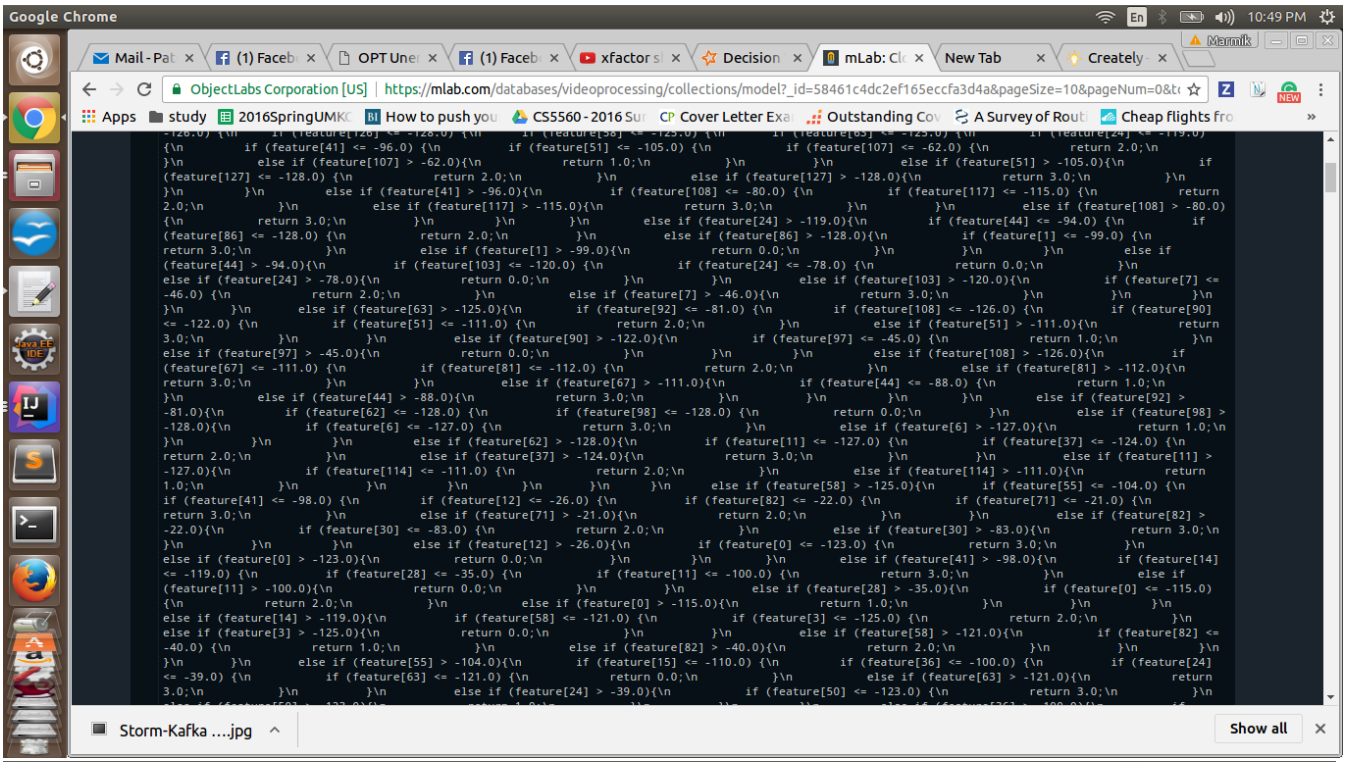
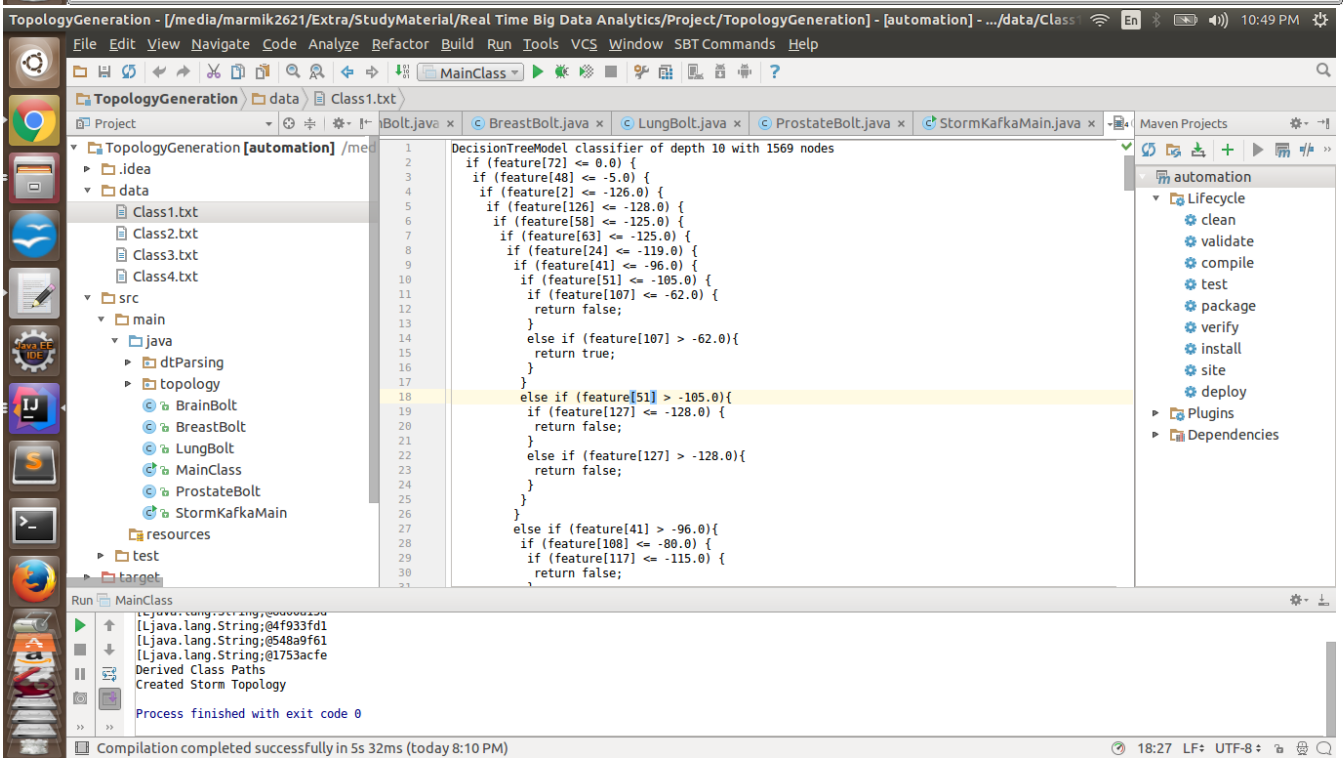
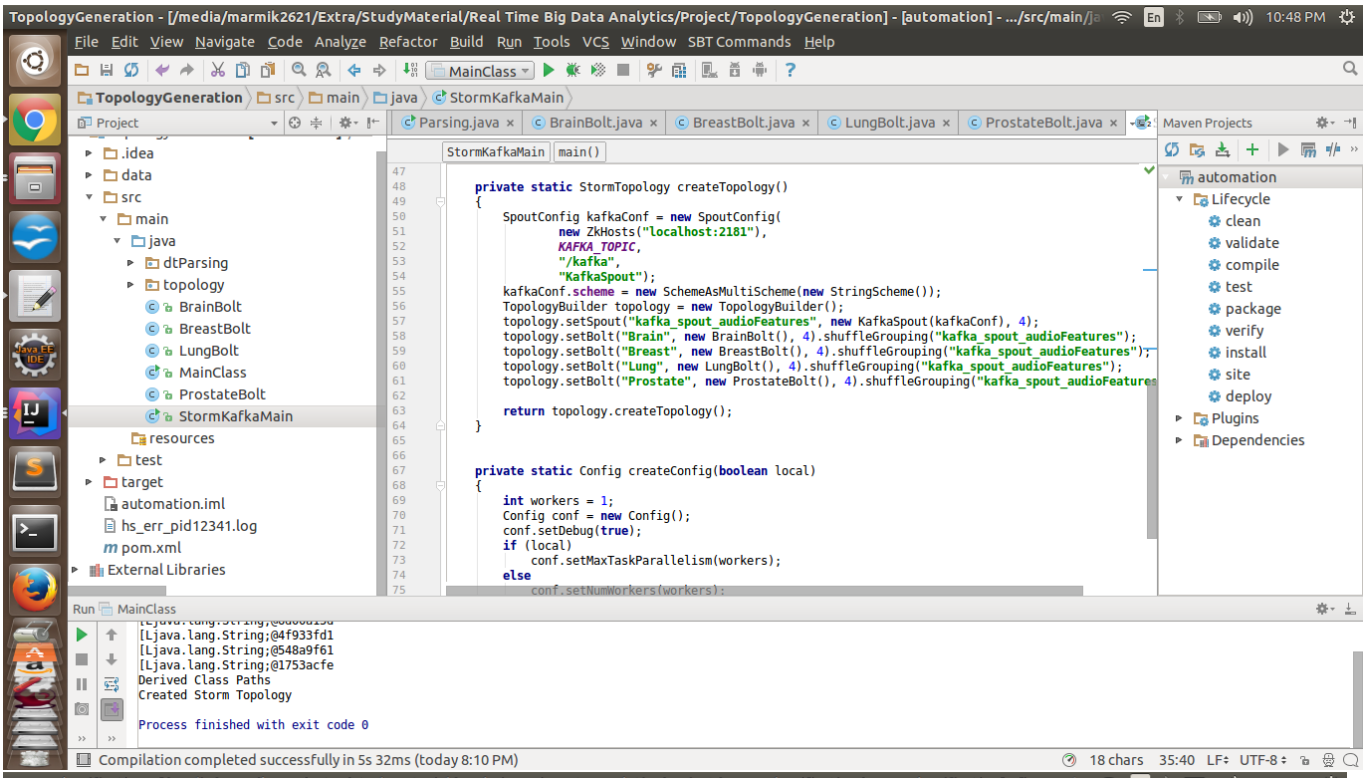
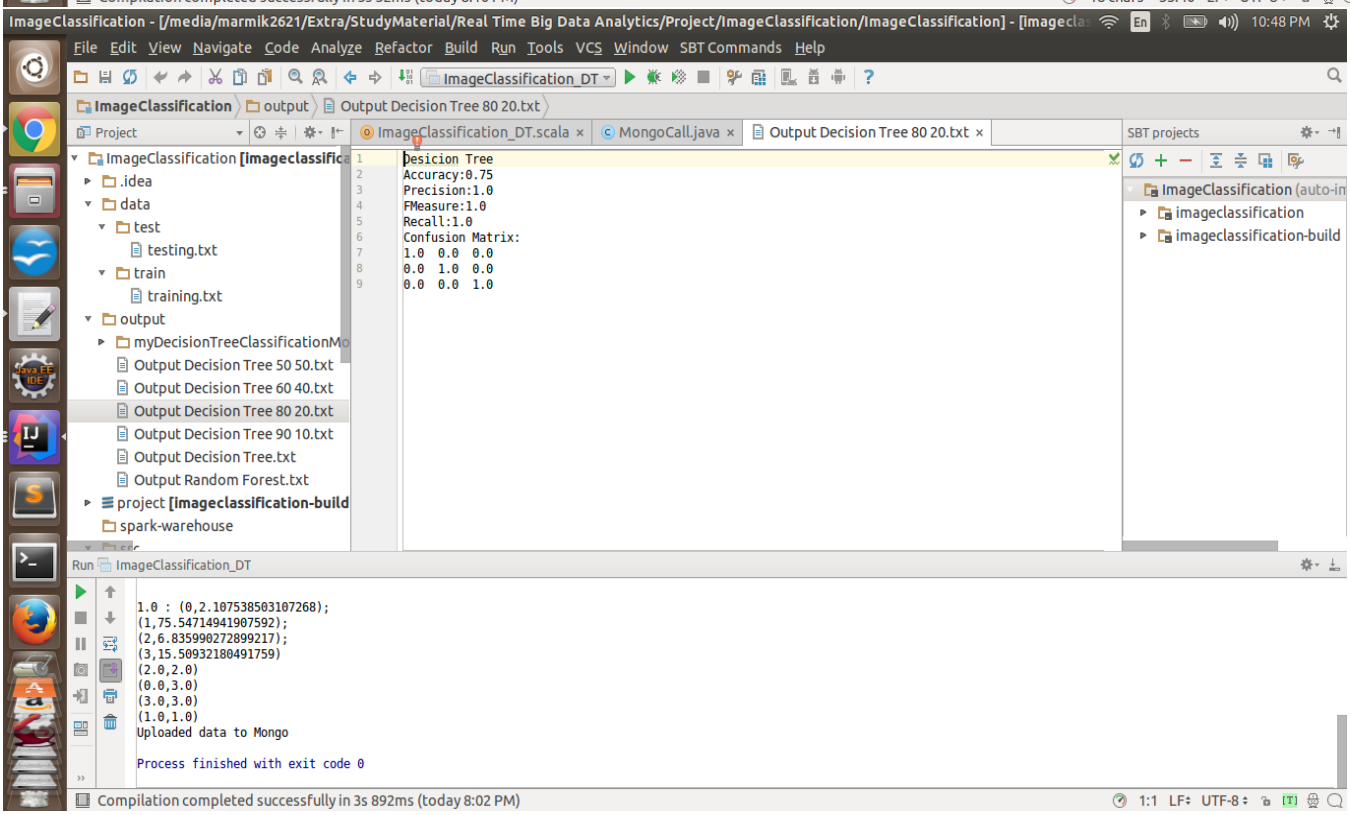


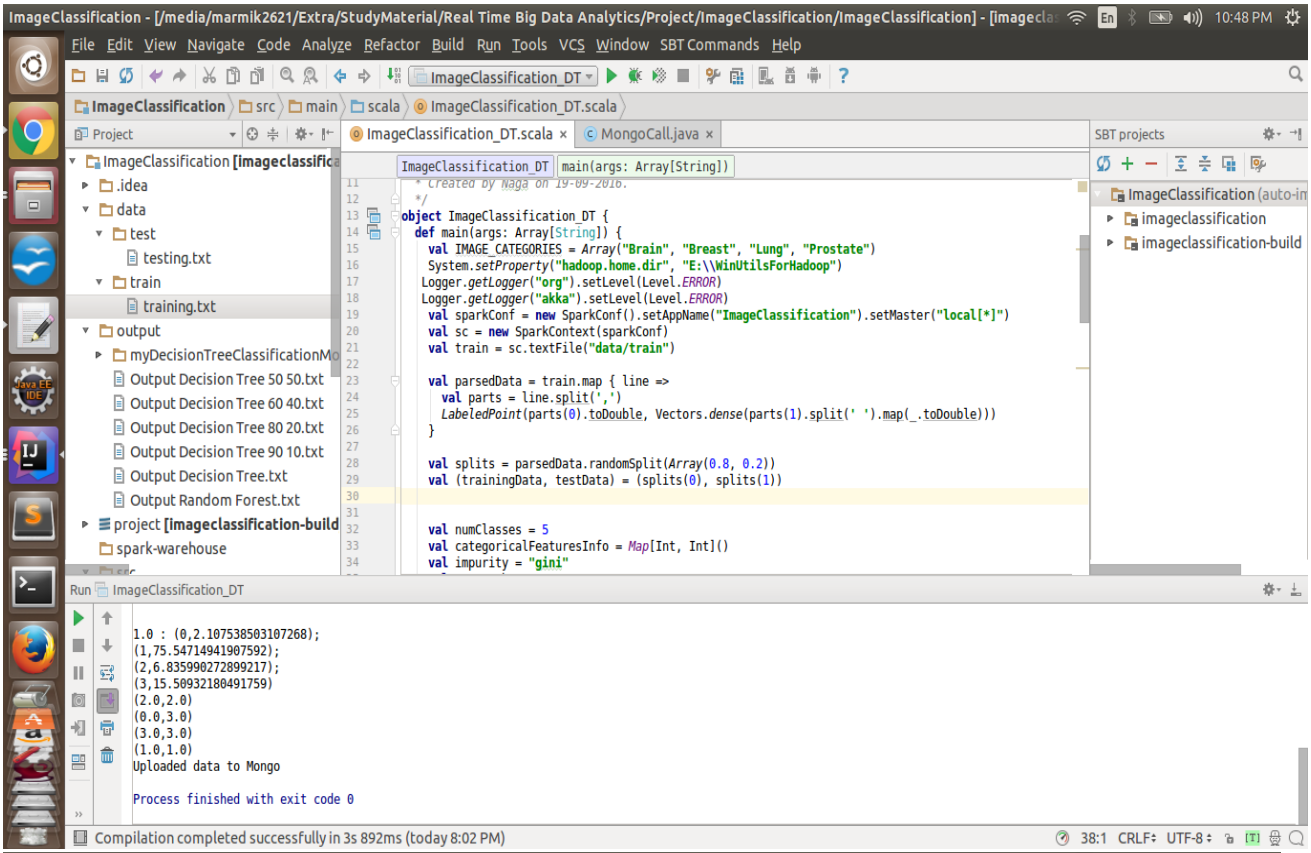
Image
Brain Cancer
[0,0] [0.0, 0.0, 0.0]
ObjectDetection.java
KafkaRestCall.java
Compilation completed successfully with 2 warnings in 4s 527ms (moments ago)

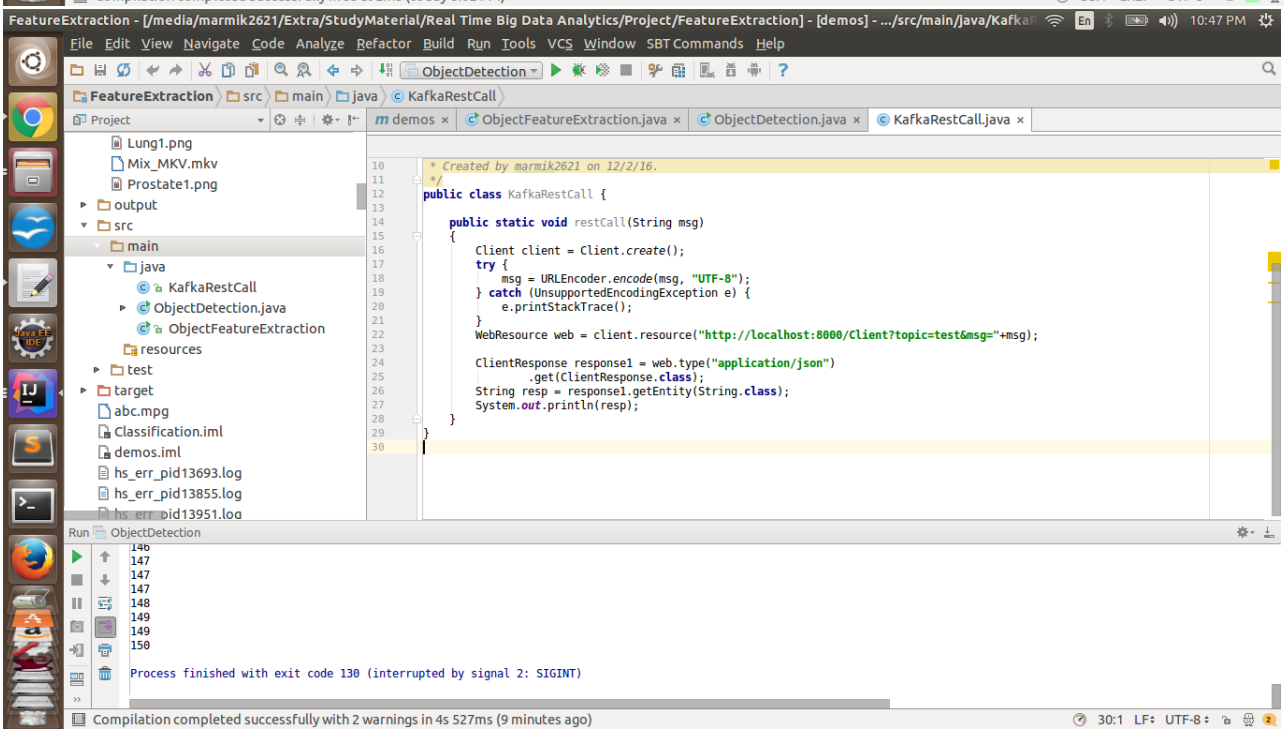

Google Chrome
10:49 PM
Mail - Pat
(1) Faceb
OPT Une
(1) Faceb
xfactor s
Decision
mLab: Cl
New Tab
Creately
ObjectLabs Corporation [US]
https://mlab.com/databases/videoprocessing/collections/model?_id=58461c4dc2ef165eccfa3d4a&pageSize=10&pageNum=0&t
Apps
study
2016SpringUMK
How to push you
CS5560 - 2016 Sur
Cover Letter Exa
Outstanding Cov
A Survey of Routi
Cheap flights fro
Storm-Kafka ....jpg
Show all

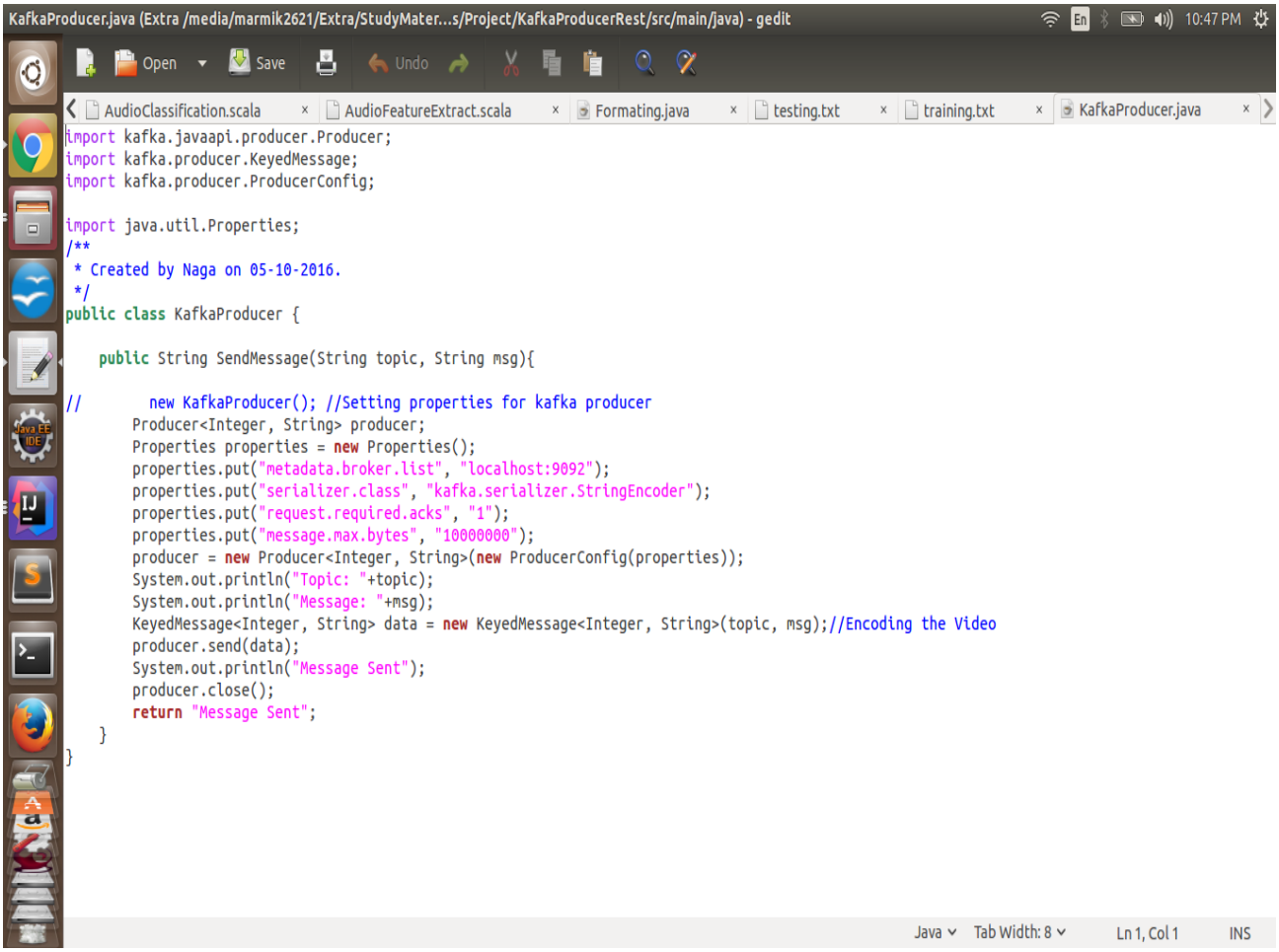

TopologyGeneration - [/media/marmik2621/Extra/StudyMaterial/Real Time Big Data Analytics/Project/TopologyGeneration] - [automation] - .../data/Class1.txt
10:49 PM
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window SBT Commands Help
MainClass
TopologyGeneration
data
Class1.txt
Project
TopologyGeneration [automation]
.idea
data
Class1.txt
Class2.txt
Class3.txt
Class4.txt
src
main
java
dtParsing
topology
BrainBolt
BreastBolt
LungBolt
MainClass
ProstateBolt
StormKafkaMain
resources
test
target
BreastBolt.java
LungBolt.java
ProstateBolt.java
StormKafkaMain.java
DecisionTreeModel classifier of depth 10 with 1569 nodes
Maven Projects
automation
Lifecycle
clean
validate
compile
test
package
verify
install
site
deploy
Plugins
Dependencies
Run MainClass
[Ljava.lang.String;@4f933fd1
[Ljava.lang.String;@548a9f61
[Ljava.lang.String;@1753acfe
Derived Class Paths
Created Storm Topology
Process finished with exit code 0
Compilation completed successfully in 5s 32ms (today 8:10 PM)
18:27 LF UTF-8


TopologyGeneration - [/media/marmik2621/Extra/StudyMaterial/Real Time Big Data Analytics/Project/TopologyGeneration] - [automation] - .../src/main/ja
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window SBT Commands Help
MainClass
TopologyGeneration src main java StormKafkaMain
Project
Parsing.java × BrainBolt.java × BreastBolt.java × LungBolt.java × ProstateBolt.java ×
StormKafkaMain main()
Maven Projects
automation
Lifecycle
clean
validate
compile
test
package
verify
install
site
deploy
Plugins
Dependencies
.idea
data
src
main
java
dtParsing
topology
BrainBolt
BreastBolt
LungBolt
MainClass
ProstateBolt
StormKafkaMain
resources
test
target
automation.iml
hs_err_pid12341.log
pom.xml
External Libraries
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("kafka_spout_audioFeatures", new KafkaSpout(kafkaConf), 4);
topology.setBolt("Brain", new BrainBolt(), 4).shuffleGrouping("kafka_spout_audioFeatures");
topology.setBolt("Breast", new BreastBolt(), 4).shuffleGrouping("kafka_spout_audioFeatures");
topology.setBolt("Lung", new LungBolt(), 4).shuffleGrouping("kafka_spout_audioFeatures");
topology.setBolt("Prostate", new ProstateBolt(), 4).shuffleGrouping("kafka_spout_audioFeatures
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);
Run MainClass
[Ljava.lang.String;@4f933fd1
[Ljava.lang.String;@548a9f61
[Ljava.lang.String;@1753acfe
Derived Class Paths
Created Storm Topology
Process finished with exit code 0
Compilation completed successfully in 5s 32ms (today 8:10 PM)
18 chars 35:40 LF UTF-8


ImageClassification - [/media/marmik2621/Extra/StudyMaterial/Real Time Big Data Analytics/Project/ImageClassification/ImageClassification] - [imageclas
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window SBT Commands Help
ImageClassification_DT
ImageClassification output Output Decision Tree 80 20.txt
Project
ImageClassification_DT.scala × MongoCall.java × Output Decision Tree 80 20.txt ×
SBT projects
ImageClassification (auto-in
imageclassification
imageclassification-build
ImageClassification [imageclassific
.idea
data
test
testing.txt
train
training.txt
output
myDecisionTreeClassificationMo
Output Decision Tree 50 50.txt
Output Decision Tree 60 40.txt
Output Decision Tree 80 20.txt
Output Decision Tree 90 10.txt
Output Decision Tree.txt
Output Random Forest.txt
project [imageclassification-build
spark-warehouse
Desicion Tree
Accuracy:0.75
Precision:1.0
FMeasure:1.0
Recall:1.0
Confusion Matrix:
1.0 0.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
Run ImageClassification_DT
1.0 : (0,2.107538503107268);
(1,75.54714941907592);
(2,6.835990272899217);
(3,15.50932180491759)
(2.0,2.0)
(0.0,3.0)
(3.0,3.0)
(1.0,1.0)
Uploaded data to Mongo
Process finished with exit code 0
Compilation completed successfully in 3s 892ms (today 8:02 PM)
1:1 LF UTF-8


ImageClassification - [/media/marmik2621/Extra/StudyMaterial/Real Time Big Data Analytics/Project/ImageClassification/ImageClassification] - [imageclas
10:48 PM
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window SBT Commands Help
ImageClassification_DT
ImageClassification src main scala ImageClassification_DT.scala
Project
ImageClassification [imageclassifica
.idea
data
test
testing.txt
train
training.txt
output
myDecisionTreeClassificationMo
Output Decision Tree 50 50.txt
Output Decision Tree 60 40.txt
Output Decision Tree 80 20.txt
Output Decision Tree 90 10.txt
Output Decision Tree.txt
Output Random Forest.txt
project [imageclassification-build
spark-warehouse
ImageClassification_DT.scala
MongoCall.java
ImageClassification_DT
main(args: Array[String])
* Created by Naga on 19-09-2016.
*/
object ImageClassification_DT {
def main(args: Array[String]) {
val IMAGE_CATEGORIES = Array("Brain", "Breast", "Lung", "Prostate")
System.setProperty("hadoop.home.dir", "E:\\WinUtilsForHadoop")
Logger.getLogger("org").setLevel(Level.ERROR)
Logger.getLogger("akka").setLevel(Level.ERROR)
val sparkConf = new SparkConf().setAppName("ImageClassification").setMaster("local[*]")
val sc = new SparkContext(sparkConf)
val train = sc.textFile("data/train")
val parsedData = train.map { line =>
val parts = line.split(',')
LabeledPoint(parts(0).toDouble, Vectors.dense(parts(1).split(' ').map(_.toDouble)))
}
val splits = parsedData.randomSplit(Array(0.8, 0.2))
val (trainingData, testData) = (splits(0), splits(1))
val numClasses = 5
val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "gini"
SBT projects
ImageClassification (auto-in
imageclassification
imageclassification-build
Run ImageClassification_DT
1.0 : (0,2.107538503107268);
(1,75.54714941907592);
(2,6.835990272899217);
(3,15.509321804917 59)
(2.0,2.0)
(0.0,3.0)
(3.0,3.0)
(1.0,1.0)
Uploaded data to Mongo
Process finished with exit code 0
Compilation completed successfully in 3s 892ms (today 8:02 PM)
38:1 CRLF: UTF-8:


FeatureExtraction - [/media/marmik2621/Extra/StudyMaterial/Real Time Big Data Analytics/Project/FeatureExtraction] - [demos] - .../src/main/java/Kafka
10:47 PM
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window SBT Commands Help
ObjectDetection
FeatureExtraction src main java KafkaRestCall
Project
Lung1.png
Mix_MKV.mkv
Prostate1.png
output
src
main
java
KafkaRestCall
ObjectDetection.java
ObjectFeatureExtraction
resources
test
target
abc.mpg
Classification.iml
demos.iml
hs_err_pid13693.log
hs_err_pid13855.log
hs_err_pid13951.log
demos
ObjectFeatureExtraction.java
ObjectDetection.java
KafkaRestCall.java
* Created by marmik2621 on 12/2/16.
*/
public class KafkaRestCall {
public static void restCall(String msg)
{
Client client = Client.create();
try {
msg = URLEncoder.encode(msg, "UTF-8");
} catch (UnsupportedEncodingException e) {
e.printStackTrace();
}
WebResource web = client.resource("http://localhost:8000/Client?topic=test&msg="+msg);
ClientResponse response1 = web.type("application/json")
.get(ClientResponse.class);
String resp = response1.getEntity(String.class);
System.out.println(resp);
}
}
Run ObjectDetection
146
147
147
147
148
149
149
150
Process finished with exit code 130 (interrupted by signal 2: SIGINT)
Compilation completed successfully with 2 warnings in 4s 527ms (9 minutes ago)
30:1 LF: UTF-8:


KafkaProducer.java (Extra /media/marmik2621/Extra/StudyMater...s/Project/KafkaProducerRest/src/main/java) - gedit
10:47 PM
Open
Save
Undo
AudioClassification.scala
AudioFeatureExtract.scala
Formating.java
testing.txt
training.txt
KafkaProducer.java
import kafka.javaapi.producer.Producer;
import kafka.producer.KeyedMessage;
import kafka.producer.ProducerConfig;
import java.util.Properties;
/**
* Created by Naga on 05-10-2016.
*/
public class KafkaProducer {
public String SendMessage(String topic, String msg){
// new KafkaProducer(); //Setting properties for kafka producer
Producer<Integer, String> producer;
Properties properties = new Properties();
properties.put("metadata.broker.list", "localhost:9092");
properties.put("serializer.class", "kafka.serializer.StringEncoder");
properties.put("request.required.acks", "1");
properties.put("message.max.bytes", "10000000");
producer = new Producer<Integer, String>(new ProducerConfig(properties));
System.out.println("Topic: "+topic);
System.out.println("Message: "+msg);
KeyedMessage<Integer, String> data = new KeyedMessage<Integer, String>(topic, msg);//Encoding the Video
producer.send(data);
System.out.println("Message Sent");
producer.close();
return "Message Sent";
}
}
Java
Tab Width: 8
Ln 1, Col 1
INS

Results

Training and Testing Data were divided into ratio and Accuracy was Calculated.

Machine Learning Algorithms:

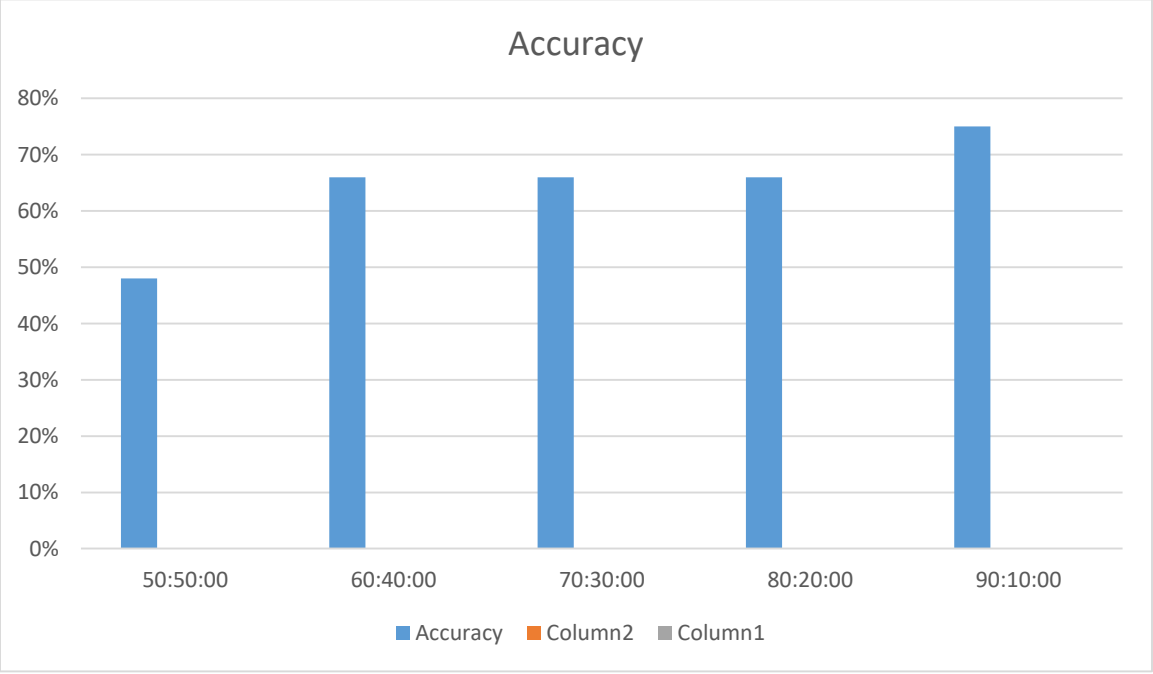
- Decision Tree

  Decision tree learning uses a decision tree as a predictive model which maps observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modelling approaches used in statistics, data mining and machine learning.

- Random Forest

  Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression)

Ratio



Project Management

Plan & Project Timelines, Members, Task Responsibility

Arunit Gupta & Marmik Patel

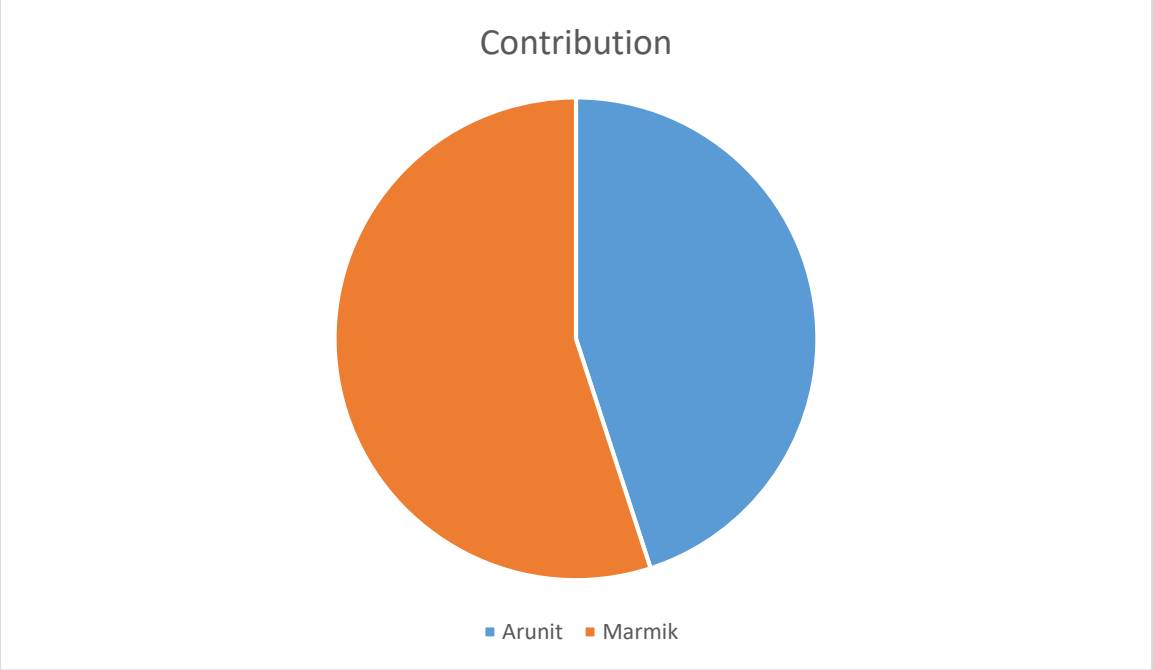
- Proposal 9/2/2016
- Increment 1 - 9/23/2016
- Increment 2 - 10/14/2016
- Increment 3 - 11/18/2016
- Increment 4 - 12/5/2016

- **Implementation status report**

Key Frame & Feature Extraction - Completed
Annotation of Videos- Completed
Cancer Stage Detection- Future Work

- Time taken (#hours) 120 hours
- Contributions (members/percentage)

Arunit Gupta- 45% (Data set collection, Analysis, Machine learning Accuracy, Training and Testing data, Reprt formation, Spark and Kafka Connection)
Marmik Patel- 55% (Architecture, Storm and Kafka connection, Storm topologies, kafka REST API implementation, Training and testing model, Video Annotation)



References

- http://www.michael-noll.com/blog/2014/10/01/kafka-spark-streaming-integration-example-tutorial/
- https://github.com/apache/spark/blob/master/examples/src/main/scala/org/apache/spark/examples/streaming/KafkaWordCount.scala
- https://www.tutorialspoint.com/apache_kafka/apache_kafka_integration_spark.htm
- http://spark.apache.org/docs/latest/streaming-kafka-integration.html
- http://kafka.apache.org/documentation.html
- https://kafka.apache.org/090/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html
- http://spark.apache.org/docs/latest/streaming-kafka-integration.html
- https://www.tutorialspoint.com/apache_kafka/apache_kafka_integration_storm.htm
- http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
- http://www.vlfeat.org/overview/hog.html