MACHINE LEARNING FOR NETWORK INTRUSION DETECTION

Project Software Documentation
March 2016

Project Summary

Project Duration:
4 weeks

Project Workload:
40 hour/man

Team Workers in Group 16:
040110078 Nurefşan Sertbaş
040090508 Betül Kantepe

Project Leader:
Nurefşan Sertbaş
sertbasn@itu.edu.tr

1. USER GUIDE

In this report, we will introduce you with required environment for realization of the project. We can categorize this procedure into two main part as preprocessing data and building classifier stages. They will be introduced in following sections respectively.

1.1 Preprocessing Data

In this section, we will briefly explain why preprocessing is needed, which operations should be performed in order to prepare data for classification. Also, we will explain how we can prepare our main 6 datasets.

In preprocessing stage, we have used MATLAB as a tool.

Step 1: [datasetgeneration.m]

Firstly we have obtained KDD Cup 1999 Data from online database [1]. The data has the following format

```
0,tcp,http,SF,219,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,39,39,
1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal.
```

Figure 1: KDD Cup data format

In this format, features are separated with a comma so that we will read by using comma as delimiter. In order to read data and adjust it into suitable categories we have wrote a script named '*datasetgeneration.m*'. The script only uses KDD data file in txt format as an input. Then, it reads data and fills the information into cell structure of the data (read_kdd_cell.m function). Then, it reads the labels of the each line (42nd column) and save it newly created txt file named with the class name. For instance, for the line represented in Figure 1 the script creates cell and fills each field in it and writes it into 'maindata_normal.txt' file.

Step 2: [10_90_partition_generation (no encoding).m]

After Step 1, we will partite the data into a training and validation set in such a way that for each of the 23 classes the last 10% is in the validation set and the first 90% is in the training set as stated in Project Description part d. So, we have read data from 'maindata_label.txt' file, took first 90% as training data and saved it as 'train_label.txt'. We have followed the same procedure for the test data in 10_90_partition_generation (no encoding).m script.

Step 3-4: [Dtype_dataset_generation (no encoding).m and Utype_dataset_generation (no encoding).m]

As we stated before we have two main class as Utype and Dtype so that operations are differ for each of them.

- Firstly we have generate D23 dataset by using Dtype class. We have already generate them into separated filed so there is nothing to do we merge them into a text file named 'D23_test.txt' and 'D23_train.txt'.

Then we merge subclasses into main categories as follows for D5 dataset

1. u2r→ perl + loadmodule + rootkit + buffer_overflow
2. Probe→ ipsweep + satan + nmap + portsweep
3. DOS→ teardrop + pod + land + back + neptune + smurf;
4. r2l→ ftp_write + guess_passwd + imap+ multihop + phf + spy + warezmaster + warezclient
5. normal

As a last part of the Dtype datasets we have merged all subclasses except normal class as attack.

- Secondly we have generated Utype datasets based on Dtype datasets. There is only one difference: we have kept the instance in dataset with 10% probability if it is not normal class.

Step 5: [updatefiles_byencoding.m]

This is the last part of the data preprocessing part. There is a critical point that we have 4 columns those are not numerical values **(See Project Report Section 2.3).** So we have did simple enumeration for these columns. Then, we have used libsvm library (svmwrite function) in order to write our data into suitable format for classification.

1. Class structure/UML

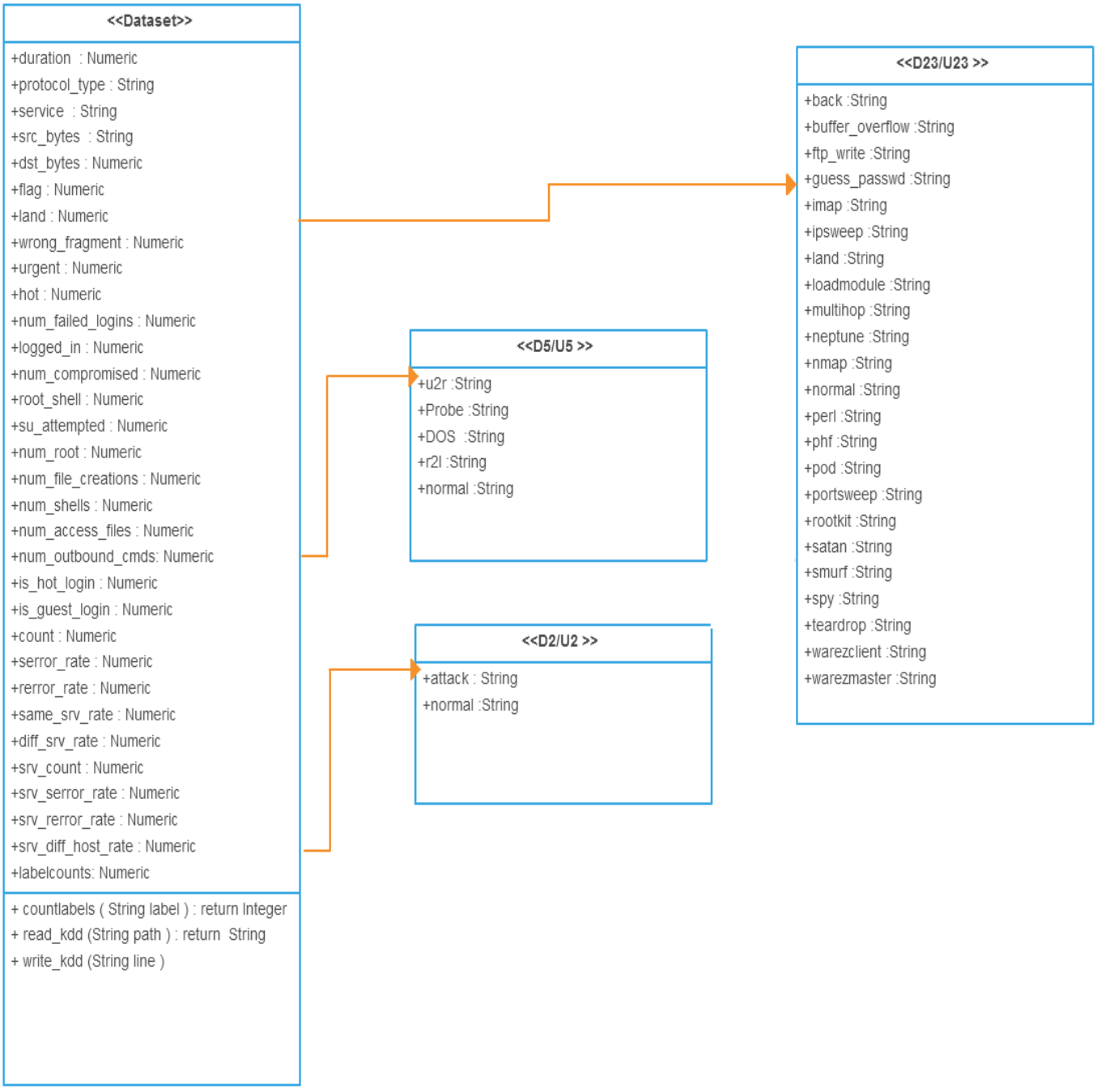

Figure 2: UML Class diagram[2]

2. Building Classifier

We have used IntellijIdea IDE in order to use Spark library with Scala. There are some steps in order to setup needed environment [1].

Step 1: Installing Scala plugin over IntelliJ IDEA

We have used IntelliJ IDEA 15.0.4 version. After the installation of the IDE we should install Scala plugin over it so that we have did the followings respectively.

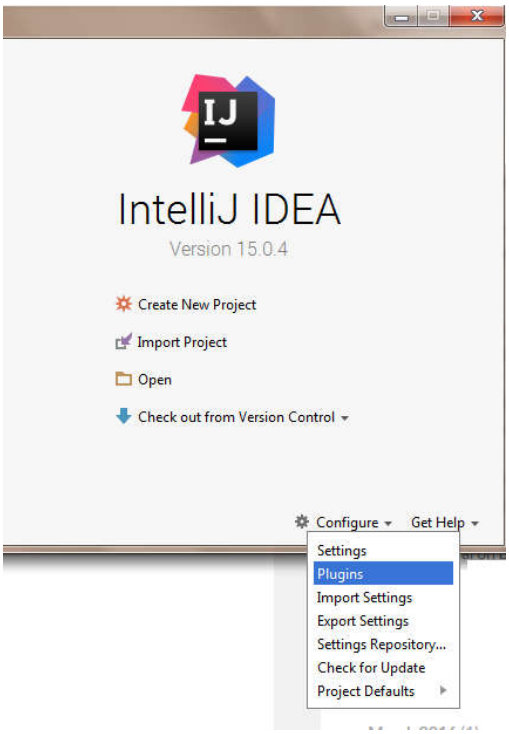

Figure 3: Installing Scala plugin-I

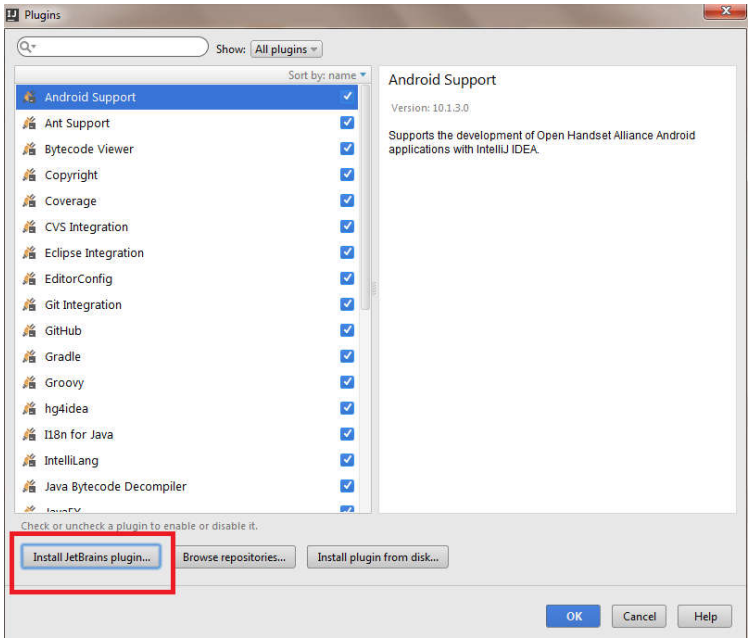

Figure 4: Installing Scala plugin -II

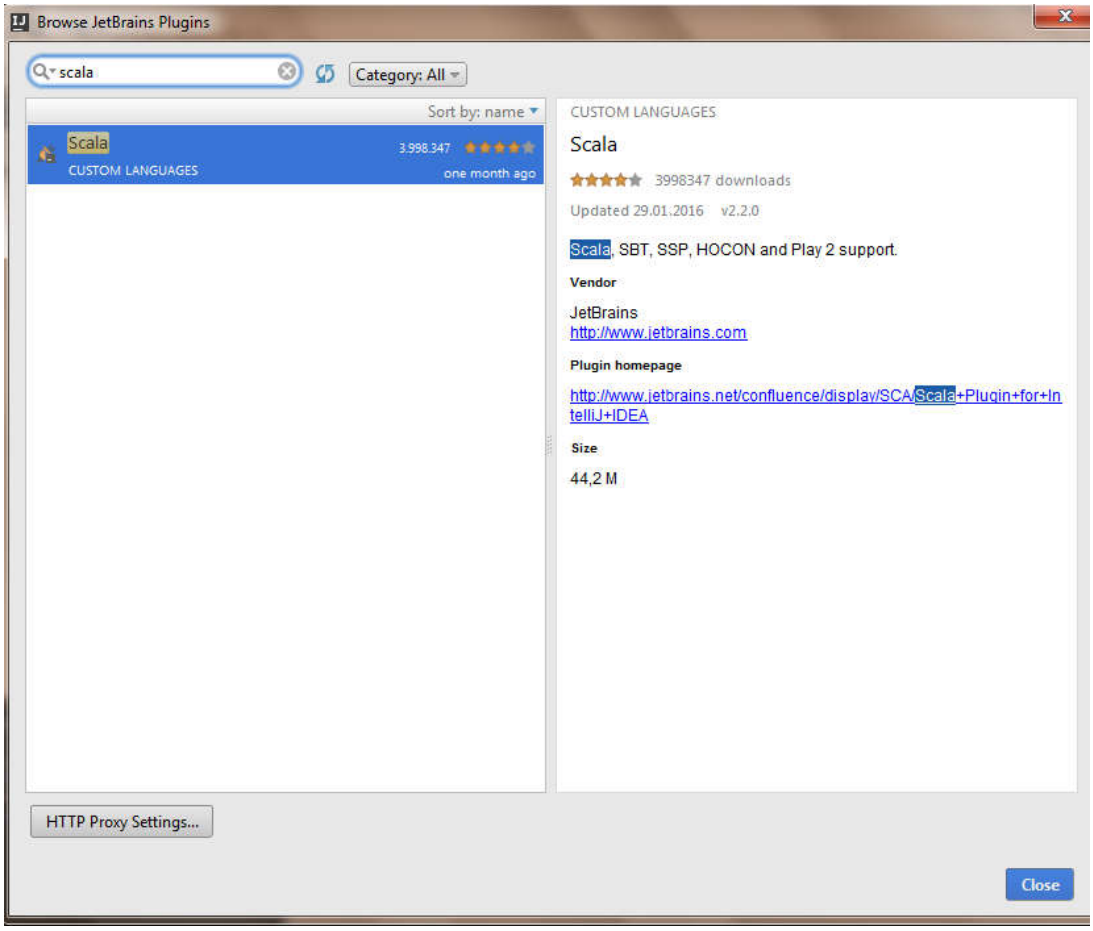

Figure 5: Installing Scala plugin - III

Normally there should be a button named ‘install Scala’ and it should be clicked in order to install the plugin. (In above case we have already installed it.). After finishing the installation, IDE should be restarted.

➢ Step 2: Creating Scala Project

At this step, we have created a new project as can be seen in below figures.

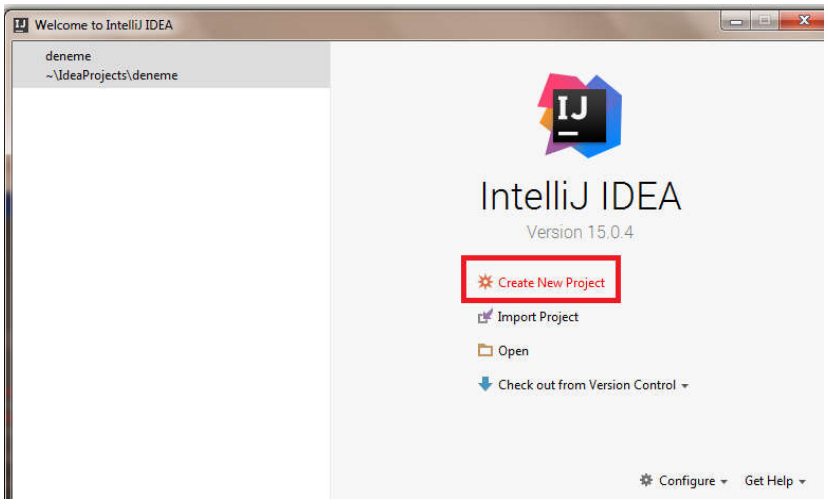

Figure 6: Creating Scala project-I

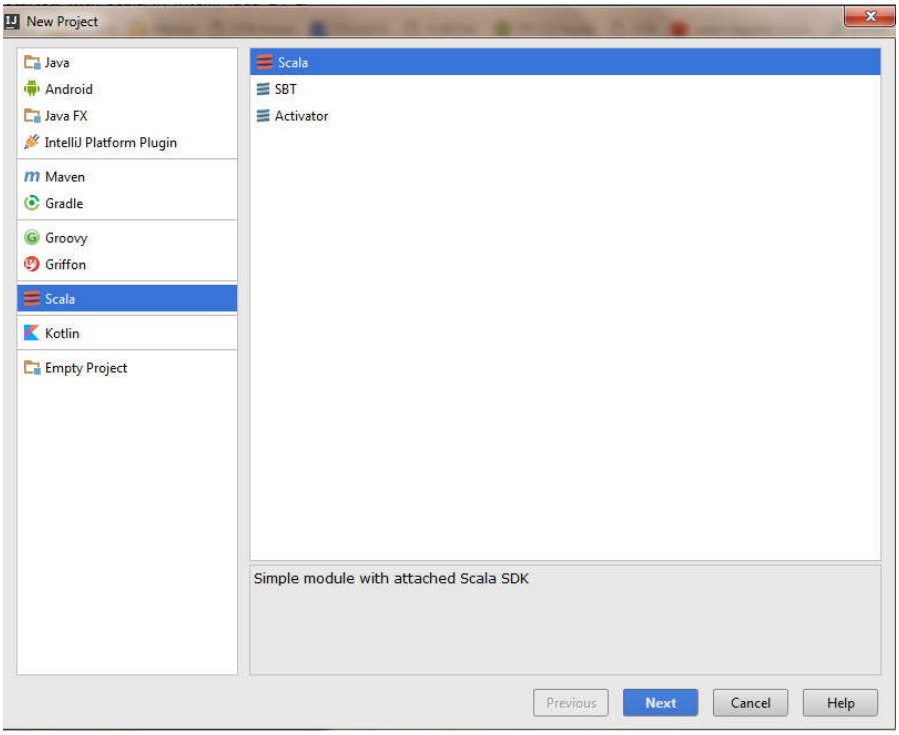

Figure 7: Creating Scala project-II

Then, we continued by clicking Next buttons until we see the below window. Here, we should select JDK then specify the path of Java installation.

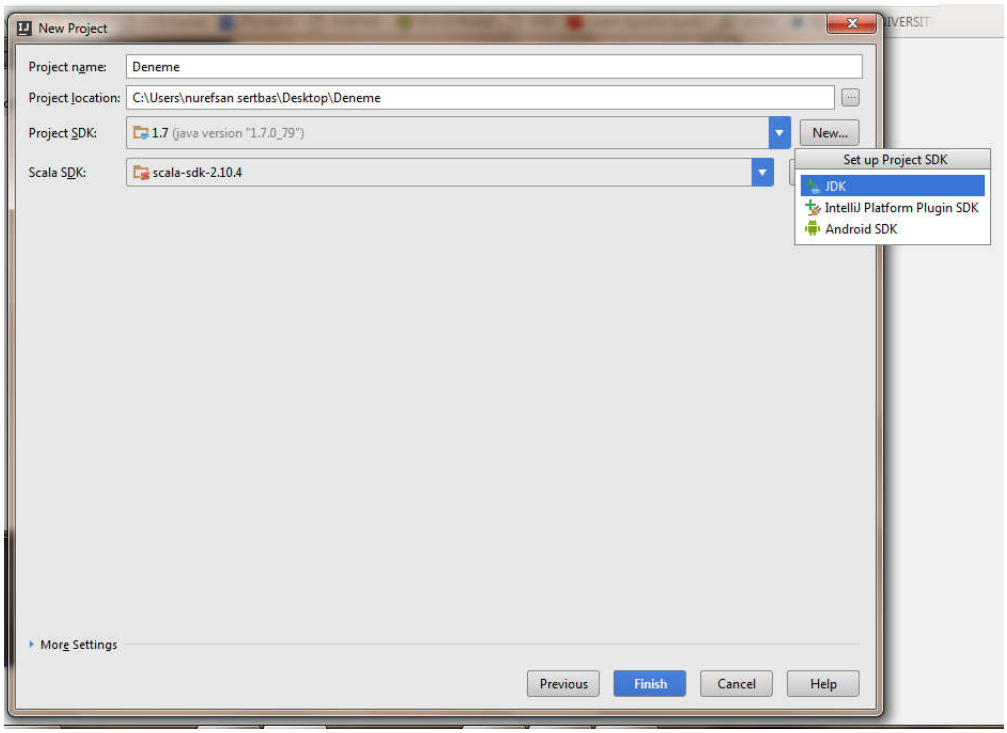

Figure 8: Creating Scala project-III

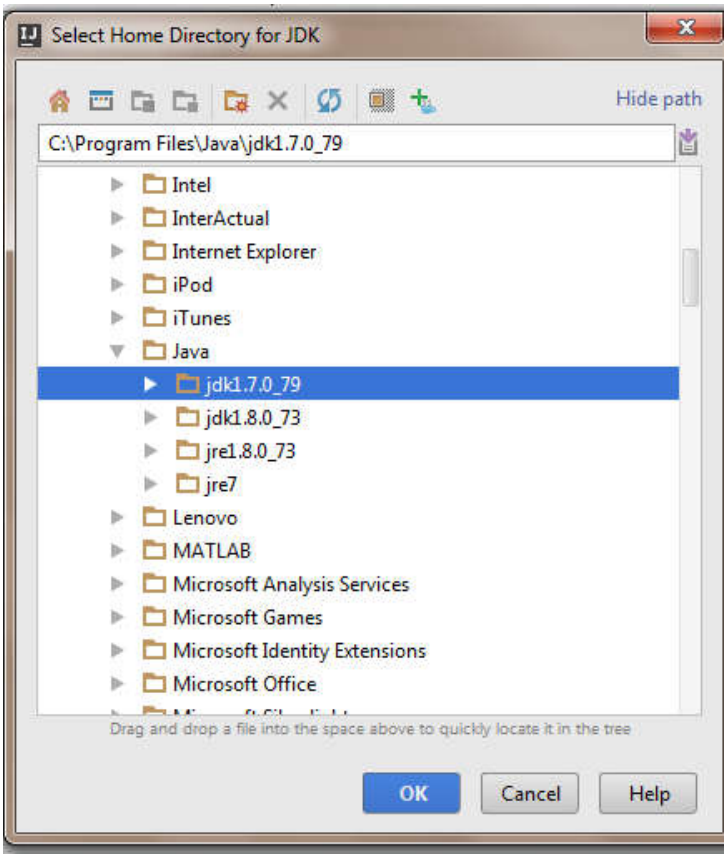

Figure 9: Creating Scala project-IV

➢ Step 3: Creating Scala Application

In order to creating a project we should follow a procedure such as
src -> New -> Scala Class.

Then, we should select **object as Kind value.**

➢ Step 4: Build Apache Spark Application in IntelliJ IDEA

We should follow below steps respectively.

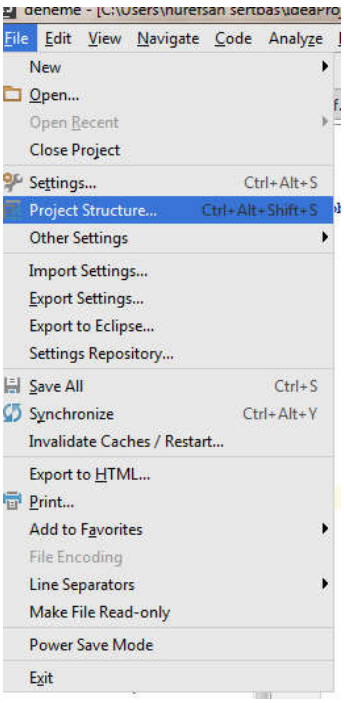

Figure 10: Build Apache Spark Application-I

Then, we have added java from + button.

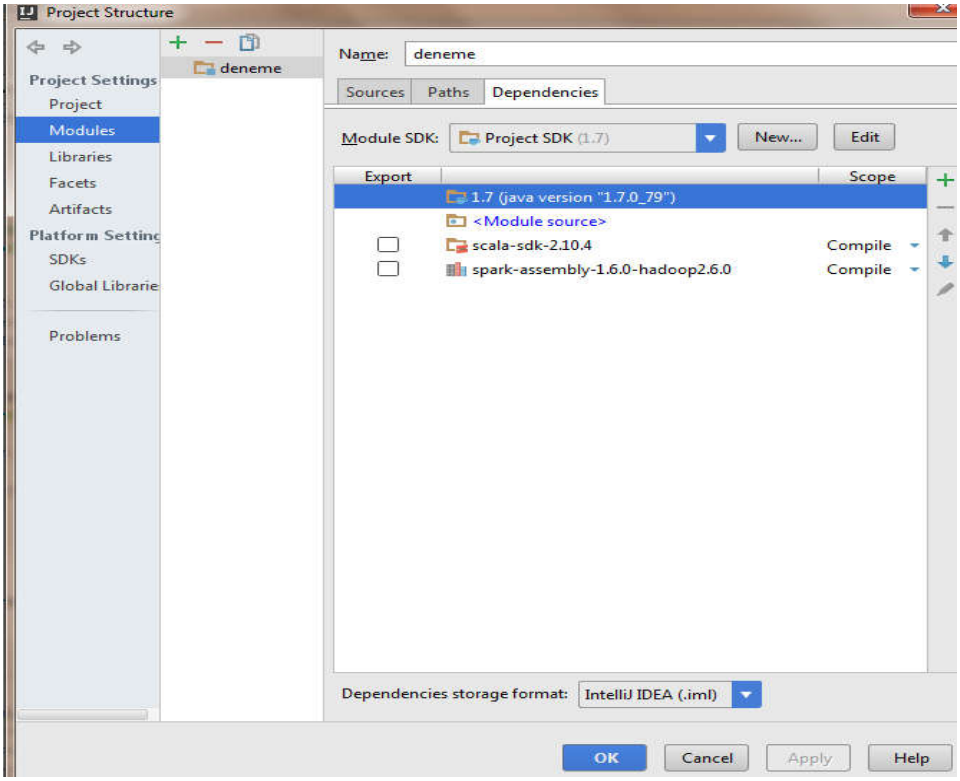

Figure 11: Build Apache Spark Application-II

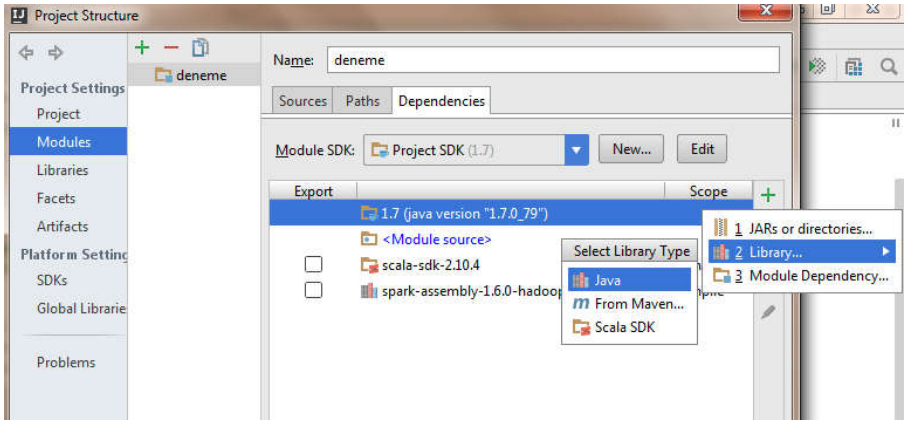

Figure 12: Build Apache Spark Application-III

Then we have selected spark-assembly-x.x.x-hadoopx.x.x.jar as a library and pressed OK button. Then, we have encountered new window named Configure Library, pressed OK. The final configuration likes this.

After all of those, there is one more critical point. We should download winutils.exe from Internet and put it into a folder in C or D. Then, we should give its path in our code as can be represented in below figure.

```
object nw_intdetection {

  def main(args: Array[String]) {
    System.setProperty("hadoop.home.dir", "c:\\winutil\\")
    val conf = new SparkConf().setAppName("Simple Application").setMaster("local")
    val sc = new SparkContext(conf)
```

Figure 13: Build Apache Spark Application-IV

3. Code Details

At this point, our environment is ready to build our classifier. In this section, we will introduce you with the detailed description of the decision tree code of the Apache Spark platform. We have imported some MLLib packages as can be seen below.

```
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.tree.DecisionTree
import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.{SparkConf, SparkContext}
```

Figure 14: MLLib package imports

Remaining part of the code is given below.

```
val trainingData= MLUtils.loadLibSVMFile(sc, "src\\new_U23_train.txt")
val testData=MLUtils.loadLibSVMFile(sc, "src\\new_U23_test.txt")
// Train a DecisionTree model.
//  Empty categoricalFeaturesInfo indicates all features are continuous.

val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "gini"
val maxDepth =4
val maxBins =64

val t0 = System.currentTimeMillis()
val model = DecisionTree.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo,
  impurity, maxDepth, maxBins)
val t1 = System.currentTimeMillis()

println("Model Building time: " + (t1 - t0) + "ms")

val t2 = System.nanoTime()
val predictionAndLabels = testData.map { case LabeledPoint(label, features) =>
  val prediction = model.predict(features)
  (prediction, label)
}
val t3 = System.nanoTime()
println("Testing time: " + (t3 - t2) + "ns")

val metrics = new MulticlassMetrics(predictionAndLabels)

// Confusion matrix
println("Confusion matrix:")
println(metrics.confusionMatrix)

val testErr = predictionAndLabels.filter(r => r._1 != r._2).count.toDouble / testData.count()
println("Test Error = " + testErr)
println("Learned classification tree model:\n" + model.toDebugString)
```

Figure 15: Decision Tree code written in Scala

It is ready to run the classifier with specified parameters. In order to detail explanation and description of the parameters see **Project Report Section 2.4.**

References

[1] "Build Apache Spark Application in IntelliJ IDEA 14.1." Nan Xiaos Blog. Web. 22 Mar. 2016.

[2] "New Getting Started with UML." New Getting Started with UML. Web. 22 Mar. 2016.