Human Activity Recognition in an Artificial Neural Network on Spark

Big Data Programming
IT715A Spring 2019

Alejandro Delgado Sanchis
b18alede@student.his.se

School of Informatics
University of Skövde

April 25, 2019

Contents

1 Introduction

According to *State of the IoT* (2018), today and every time we are advancing more in the world of technology and particularly in the field of intelligent devices. To be able to create these new devices we need thousands of sensors that are responsible for capturing all the data that surrounds us. Data such as temperature, humidity, C02 level or number of vehicles on the road. But these sensors are not so far from us, we carry many of them daily in our pocket in the form of Smartphone. Every day we have more intelligent phones, capable of capturing a lot of information about us. Recently there have been launched several applications for Smartphones that measure the kilometers traveled and pay you money based on that (Bumgardner, 2019). It is obvious that the application company needs to receive something in return. If we stop to read the agreements of use of the application it is clear that all this data are sent to other companies so that they can benefit from it. According to Economist (2017) the data currently has more value than oil. However, it is not only important to have these data but also to know how to extract knowledge from them. Here is where the skills of data scientists come in.

In this project, the whole process that a data scientist would do is carried out. From having the data raw to being able to draw conclusions about them. For it we have the readings of the accelerometer and gyroscopes coming from a smartphone. In this dataset the readings of the sensors have been carried out and labeling that movement has been carried out. In this way we have six characteristics (three accelerometer axes and three gyroscope axes) and an activity associated with each reading. Therefore, the main question we ask in this project is: *How to predict human activity using this dataset and a neural network?.*

Spark has been used to answer this question. This framework is responsible for handling large amounts of data and preparing it for further analysis. But Spark also has a powerful library for machine learning. This is why this library has been used to create a multilayer perceptron classifier. With this, an artificial neural network feedforward has been created capable of classifying the movements of people in twelve possible activities.

2 Analysis

The first step that has to be done in any process on data is the choice of a data set. With this set the realization of a cleaning and normalization will depend on the state of the data. To carry this out we have to understand the entire data set perfectly, in this way we can then choose the correct analysis model.

2.1 Dataset

The dataset used in this project is *Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set* (2015). It is composed of 30 experiments carried out on a group of 30 volunteers in an age range of 19-48 years. All the participants were wearing a Smartphone (Samsung Galaxy S II) on the waist during the experiment execution. Ac-

cording to the description of the dataset, the values have been captured the 3-axial linear acceleration and 3-axial angular velocity at a constant speed of 50Hz using the integrated accelerometer and the gyroscope of the device. In this dataset there are also separate data in train and test, but we are going to take only the data from the folder "RawData". I have chosen this dataset for how powerful it can be to know the movement that each person is doing depending on the reading of different sensors. For example, in the industry world, to know the movements done by the employees and thus, avoid possible injuries in the workplace.

The first step is to know the structure and information available in the dataset. Inside the folder "RawData" there are 61 files available with the readings of the accelerometer and 61 files with the readings of the gyroscope. Each one has 3 columns of double numbers belonging to the three axes of each sensor. The files are named with the experiment number and the user number. Finally, there is a file called labels.txt. In this file there are four columns: experiment number, user number, activity id and the reading interval where the activity was performed. The figure shows the activities tagged in the dataset.

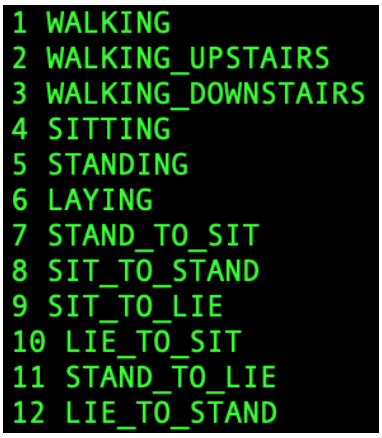

Figure 1: Activities available for classification.

For a correct use of a classifier we need to place all the characteristics in columns next to the "label" column. As we will mention in the conclusions this was very difficult because these labels are stored in another file and in the form of intervals.

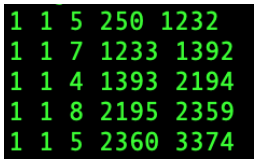

Figure 2: Top 5 lines from labels.txt file.

In the figure above, we see the first five lines of the labels file, where reference is made to the first experiment and the first user. In other words, the file "acc_exp01_user01.txt" and

"gyro_exp01_user01.txt". Within these files between lines 250 and 1232, activity 5 (Standing) was carried out and then from line 1233 to 1392 the volunteer carried out activity 7 (Stand to sit).

To be able to attach the label column to the rest of the characteristics what I am going to do is to create a new file with the same number of lines as the number of sensor readings. Using a script in Python I will insert the corresponding activity in each reading, for example, for the first case I have written a 0 from line 1 to line 249 and from line 250 to line 1392 we will introduce a 5. In this way I have a file with the corresponding labels for each reading. The next thing is to join that file with the features for it I will use the Spark framework.

2.2 Spark

Once we have the labels ready we will put all the files and labels together in a Spark dataFrame. To do this we use the read() function and then the join() function. For a correct join we created several index columns, since they must match each of the readings of the accelerometer with those of the gyroscope and later with the labels.

With this we can now use the Mllib Spark library. This framework has many machine learning options. For example for clustering or text mining but also for classification. For this project we will use the Multilayer perceptron technology. According to Documentation-Spark (2019) this classifier is based on the feedforward artificial neural network and consists of multiple layers of nodes. Each layer is fully connected to the next layer in the network. Concerning the activation functions this neural network has the following:

Nodes in intermediate layers use sigmoid (logistic) function:

$$\mathrm{f}(z_i) = \frac{1}{1 + e^{-z_i}} \tag{1}$$

Nodes in the output layer use softmax function:

$$\mathrm{f}(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{N} e^{z_k}} \tag{2}$$

The configuration of the activation functions cannot be changed but is interesting enough to create the classifier.

Finally, we need to evaluate the quality of our classifier. Mllib library has several possibilities of evaluation of multiclass classification for label based. For this work we are going to use only the accuracy value of the neural network. According to *Evaluation Metrics-API* (2019) the formula used to calculate accuracy is the following:

$$ACC = \frac{TP}{TP + FP} = \frac{1}{N} \sum_{i=0}^{N-1} \hat{\delta} \left(\hat{\mathbf{y}}_i - \mathbf{y}_i \right) \tag{3}$$

In addition to this evaluation method Spark Mllib has more like the Confusion Matrix, but for this work we are going to focus only on this accuracy evaluation mode, which is sufficient to evaluate this neural network.

3 Implementation

In this section we will show part of the code that was needed to achieve the objectives of this project.

3.1 Preparing the data

The first script that we are going to implement is "prepareLabels.py".

```
# Main Dataset Path
path_Dataset = "./HAPT Data Set/RawData/"

# File where there are all the labels
file_labels = open(path_Dataset + "labels.txt", "r")

# New file
file_data = open("./labels_ready.txt", "w+")

# To know when I change of experiment
previousExp = 0

# Index for all lines in labels.txt
index_general = 0

# Index for each experiment in labels.txt
index_local = 0

# Loop each line in labels.txt
for aline in file_labels:

        # Split the columns
        values = aline.split()

        # Check if is a different experiment
        if (int(values[0]) != previousExp):
                index_local = 0

        # If there is not a label to the line I will write a 0 in the file
        if (index_local != int(values[3])):
                for i in range(index_local, int(values[3])):
                        file_data.write(values[0] + " " +
                                        str(index_general) + " 0\r\n")
                        index_local += 1
                        index_general += 1

```

```
        # If there is a label to the line I will write the correspond label
        for a in range(index_local, int(values[4])):
                file_data.write(values[0] + " " +
                                str(index_general) + " " + values[2] + "\r\n")
                index_local += 1
                index_general += 1

        # I update the previousExp
        previousExp = int(values[0])

# Close the files
file_labels.close()
file_data.close()
```

As explained in the analysis part, this is the first step. This script in Python aims to create a new file with the labels of each reading of the sensors with an index. Once the script is executed there will be as many lines as readings from the sensors. Also note that there are readings that have no tag assigned and therefore will be marked as activity 0, which will later be filtered.

Once this file is created we can start using Spark to read all the sensor readings and match them with the label file created above. Below is some of the Scala code for this.

```
val dfAcceleration_tmp = spark.read.
                      option("header", false).
                      option("inferSchema", false).
                      option("delimiter", " ").
                      schema(schemaAcceleration).
                      csv(path_Dataset + "/acc*").
                      select("acceleration_x","acceleration_y","acceleration_z").
                      withColumn("file_name_A", getNameClean(input_file_name()))

val dfAcceleration = dfAcceleration_tmp.
                      orderBy($"file_name_A").
                      withColumn("indexA",monotonically_increasing_id+1)

val dfGyroscope_tmp = spark.read.
                       option("header", false).
                       option("inferSchema", false).
                       option("delimiter", " ").
                       schema(schemaGyroscope).
                       csv(path_Dataset + "/gyro*").
                       select("gyroscope_x","gyroscope_y","gyroscope_z").
                       withColumn("file_name_B", getNameClean(input_file_name()))
```

```
22
23  val dfGyroscope = dfGyroscope_tmp.
24                    orderBy($"file_name_B").
25                    withColumn("indexB", monotonically_increasing_id+1)
26
27  val dfLabelsReady = spark.read.
28                      option("header", false).
29                      option("inferSchema", false).
30                      option("delimiter", " ").
31                      schema(schemaLabelsReady).
32                      csv("../tmp/labels_ready.txt").
33                      select("Id_Experiment", "Index","label")
```

In the dataset of the accelerometer and gyroscope has also been created an automatic index and attached the name of the file. For the file name field an UDF (User-Defined Functions) function has been used that eliminates the extension and the beginning of the name. This is useful later because one of the problems was that Spark reads the files in an unordered manner. This was a big problem since we must keep the order at all times to match these datasets with the file of the labels. To solve this is sorted by file name in each reading before of create the index.

The last data processing to be carried out is the joining of the three datasets (accelerometer, gyroscope and labelsReady). For this we use the created index. Finally, to facilitate the machine learning process, we eliminate all unnecessary columns, filter by classes greater than 0 and sort by index. For this we use the implementation below.

```
1   /*
2   * Join dfLabelsReady with dfGyroscope and dfAcceleration
3   */
4   val dfNew = dfLabelsReady.
5                 join(dfAcceleration,
6                 dfAcceleration("indexA") === dfLabelsReady("Index"),
7                 "inner")
8
9   val dfReady_tmp = dfNew.join(dfGyroscope,
10                    dfNew("indexA") === dfGyroscope("indexB"), "inner").
11                    drop(dfNew("file_name_A")).
12                    drop(dfGyroscope("file_name_B")).
13                    drop(dfNew("indexA")).
14                    drop(dfNew("Id_Experiment")).
15                    drop(dfGyroscope("indexB"))
16
17  /*
18  * Final dataset with the features and target value. I
```

```
 * filtered label > 0
 */
val dfReady = dfReady_tmp.
              filter($"label" > 0 ).
              orderBy($"indexA").
              drop($"Index")
```

With all this we already have the data clean and ready to apply the technique of classification of a neural network.

3.2 Creating a Neural Network

To start machine learning with the dataset we must also make some modifications to the data so that the classification process can be carried out successfully.

According to Databricks (2019), the first process to perform is the creation of a vector with all the characteristics we want to use to classify it using the existing method in MLlib of VectorAssembler(). The next step is to scale the characteristics between a range. Although most of the readings we have are concentrated in a range it is always better to scale for a higher result. For this we use the StandardScaler() method. Finally we index the characteristics and the label column using StringIndexer() and VectorIndexer(). All is achieved with the following implementation.

```
/*
* Using VectorAssembler(), To create a vector with all features.
*/
val assembler = new VectorAssembler().
                    setInputCols(Array("acceleration_x", "acceleration_y",
                    "acceleration_z", "gyroscope_x", "gyroscope_y", "gyroscope_z")).
                    setOutputCol("features")
val features = assembler.transform(dfReady)

/*
* Using StandardScaler(), To Standard all the features using Standard Deviation.
*/
val scaler = new StandardScaler().
             setInputCol("features").
             setOutputCol("scaledFeatures").
             setWithStd(true).
             setWithMean(false)

val scalerModel = scaler.fit(features)
val scaledData = scalerModel.transform(features)

```

```scala
/*
* I created a Index to label and features columns using
* StringIndexer() and VectorIndexer().
*/
val labelIndexer = new StringIndexer().
                   setInputCol("label").
                   setOutputCol("indexedLabel").
                   fit(scaledData)

val featureIndexer = new VectorIndexer().
                     setInputCol("scaledFeatures").
                     setOutputCol("indexedFeatures").
                     setMaxCategories(numFeatures).
                     fit(scaledData)
```

The next step is to split the data into train and test. To do this we will choose the same percentage as in the paper Reyes-Ortiz et al. (2014), 70% for train and 30% for test. In this way we can compare the results obtained with them later. To create the Multilayer Perceptron a function has been used that is called with the desired layer architecture. This function returns the predictions about the test set. These results are then used to evaluate the quality of our classifier. Below we can see the complete implementation.

```scala
/*
* To split the data between train and test. For that I used
* two global variables.
*/
val splits = scaledData.
             randomSplit(Array(trainSet, testSet))

val trainingData = splits(0)
val testData = splits(1)

/*
* This function is to create a model given an architecture of neural
* network. Return the predictions to testData.
*/
def CreateModel(layers: Array[Int]): org.apache.spark.sql.Dataset[_] = {

  val trainer = new MultilayerPerceptronClassifier().
                setLayers(layers).
                setLabelCol("indexedLabel").
                setFeaturesCol("indexedFeatures").
                setBlockSize(128).
                setSeed(System.currentTimeMillis).
```

```
                setMaxIter(200)

  val labelConverter = new IndexToString().
                      setInputCol("prediction").
                      setOutputCol("predictedLabel").
                      setLabels(labelIndexer.labels)

  val pipeline = new Pipeline().
               setStages(Array(labelIndexer,
               featureIndexer, trainer, labelConverter))

  val model = pipeline.fit(trainingData)

  return model.transform(testData)
}

println("-------------------------------------------------------")
println("--------------- CREATING NEURAL NETWORK ---------------")
println("-------------------------------------------------------")

/* Neural network configuration */
val layers = Array[Int](numFeatures, 17, 17, numClasses)

/* Called a function CreateModel and return test prediction */
val predictions = CreateModel(layers)

println("-------------------------------------------------------")
println("----------------------- RESULTS -----------------------")
println("-------------------------------------------------------")

/* Multilayer Perceptron Classifier Evaluation */
val evaluator = new MulticlassClassificationEvaluator().
                setLabelCol("indexedLabel").
                setPredictionCol("prediction").
                setMetricName("accuracy")

val accuracy = evaluator.evaluate(predictions)
println("Test Error = " + (1.0 - accuracy))
```

With all this implementation we can now begin to obtain results from our classified human activity.

4 Results

To analyze the results of our classifier we have used the percentage of accuracy and error that returns the method of evaluation of the library itself. It should be highlighted that as a neural network it takes a long time between each of the tests because it has a high computational cost.

The main objective is to choose the best layered architecture for the neural network to obtain a good result. To know the best architecture we have created a battery of tests with different number of layers and configurations. In the table 1, we can see the results of accuracy and error obtained.

Id Result	Accuracy	Error	Layer Architecture	Comments
1	61%	39%	(6, 8, 12)	Not Data scaled
2	71%	29%	(6, 15, 20, 15, 12)	Data scaled
3	64%	36%	(6, 8, 8, 12)	Not Data scaled
4	69%	31%	(6, 15, 20, 15, 12)	Not Data scaled
5	70%	30%	(6, 30, 40, 40, 30, 12)	Data scaled
6	65%	35%	(6, 30, 40, 30, 12)	Not Data scaled
7	72%	27%	(6, 17, 17, 12)	Not Data scaled
8	75%	25%	(6, 17, 17, 12)	Data scaled
9	58%	42%	(6, 17, 20, 17, 12)	With noise and not Data scaled[1]
10	68%	32%	(6, 20, 20, 20, 12)	Not Data scaled

Table 1: Results with several layer configurations.

According to the table above we can see that the best result is obtained with experience 8. In this one, we obtain a 75% of accuracy and 25% of error. For this we have configured the neural network with two hidden layers of 17 nodes each. The data have also been scaled according to their standard deviation. The test 9 has been made including all the readings that were not labeled[1], i.e. where the label is equal to 0. This test was the worst of all because it included a lot of noise in the network.

One of the proposition of this project is to compare the results obtained with the paper Reyes-Ortiz et al., 2014. Although the same percentage of training data and test data has been used, a proper comparison could not be made. In this paper another classification algorithm has been used. It is basically Support Vector Machine (SVM) combined with more improvements. In this paper an error of 7.72% is reached while with this method of Multilayer perceptron classifier a 25% has been achieved. The difference is quite big but still we have achieved a good classifier.

5 Conclusion

To finish this project we are going to draw some conclusions and possible improvements. The first thing to say is that a fairly high level of classification has been achieved, which answers the question we asked ourselves at the beginning of this project. This classifier can be used with 75% accuracy to classify activities according to the readings of the accelerometer and gyroscope sensors.

In order to carry out this solution, problems of different magnitudes have been overcome. The main problem resulted in the understanding of data and placing the correct existing label in a file along with the sensor readings. This was finally solved with the python script shown in the implementation section. Also problems when using the Spark machine learning library.

Some improvements could be made to achieve higher scores. One of the possibilities is to use other tools for the creation of neural networks, such as the [2] TensorFlow framework. It could also be improved if it is possible to enlarge the dataset with more readings. Finally to emphasize that a neural network cannot always be the best option to classify. It would be interesting to know in a future work for which type of classification is more appropriate a neural network.

[2] 'TensorFlow is an end-to-end open source platform for machine learning.'

References

Bumgardner, W. (2019), 'How to get paid to walk using apps'.
URL: *https://www.verywellfit.com/walking-apps-that-earn-rewards-3434997*

Databricks (2019), 'Multilayer perceptron classifier'.
URL: *https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3741049972324885/1019862370390522/4413065072037724/latest.html*

Documentation-Spark (2019), 'Classification and regression'.
URL: *https://spark.apache.org/docs/2.4.0/ml-classification-regression.htmlmultilayer-perceptron-classifier*

Economist, T. (2017), 'The world's most valuable resource is no longer oil, but data'.
URL: *https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data*

Evaluation Metrics-API (2019).
URL: *https://spark.apache.org/docs/latest/mllib-evaluation-metrics.htmlmulticlass-classification*

Reyes-Ortiz, J.-L., Oneto, L., Ghio, A., Sam, A., Anguita, D. & Parra, X. (2014), 'Human activity recognition on smartphones with awareness of basic activities and postural transitions', *Artificial Neural Networks and Machine Learning ICANN 2014 Lecture Notes in Computer Science* p. 177184.

Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set (2015).
URL: *http://archive.ics.uci.edu/ml/datasets/Smartphone-Based+Recognition+of+Human+Activities+and+Postural+Transitions*

State of the IoT (2018).
URL: *https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/*