UNLV

University of Nevada, Las Vegas
Howard R. Hughes College of Engineering
Department of Computer Science

Project Report

Activity Recognition on Wearable Sensor Dataset using Spark Platform

Author

Sina Mahdipour Saravani

sina.mps@unlv.edu

Advisor

Mingon Kang

mingon.kang@unlv.edu

Contents

1- Motivation

Nowadays, we are trying to make everything around us be more and more intelligent to help us live better and easier. Activity recognition is one of the tasks that can help us with that in different domains. If smart objects or systems around us can detect specific activities we perform using our motion data, they can decide to perform certain actions. For example, in the field of IoT and smart homes, we may be interested in letting the lights turn off automatically when we go to bed. Or in the interdisciplinary field of artificial intelligence in healthcare, we are interested to detect potential fall of elderly people to call the emergency service or their family members.

On the other hand, we are living in the era of big data. There are lots of data available in different domains and such data can yield to knowledge discovery, but we need appropriate techniques to work with such huge amounts of data.

In this project we are going to propose machine learning models to classify human activities based on a specific dataset collected in an experiment, yet we want our solution to be scalable in order to be reusable in real big datasets environment.

2- Problem Definition

Our problem defines as providing scalable solution for classifying a person's activity in a room equipped with sensor readers who is wearing a battery-less RFID sensor into four classes of *sitting on the bed*, *sitting on the chair, lying on the bed,* and *ambulating*. Therefore, our problem can be rephrased as follows:

- Given:
    - Motion data of the person wearing a batter-less RFID sensor
- Goal:
    - Classify the person's activity

3- Dataset Description

The dataset we used for this project is obtained from University of California Irvine Machine Learning Repository [1]. It [2] contains the motion data of 14 healthy older people aged between 66 and 86 years old. The person had been equipped with a battery-less wearable sensor on top of their clothing at sternum level and had performed broadly scripted activities. Since the sensor that has been used is passive, the data is sparse and noisy. The experiment had been carried out in two different clinical room settings. The first setting used 4 RFID antennas in the room, one attached to the ceiling and three to the walls. In the other setting only three antennas had been used, two of which installed on the ceiling and one on the wall.

The scripted activities performed by the participants are as follows:

- Walking to the chair
- Sitting on the chair
- Getting off the chair

- Walking to bed
- Lying on the bed
- Getting off the bed
- Walking to the door

Yet in the dataset, they have been categorized into four different groups of *sitting on the bed, sitting on the chair, lying on the bed,* and *ambulating* which include standing and walking around the room [2].

Dataset is provided in two different folders containing experimental data for the two room settings and is divided into chunks of ".csv" files. Data columns are shown in table 1.

C 1	C 2	C 3	C 4	C 5	C 6	C 7	C 8	C 9
Time	Acc Frt.	Acc Ver.	Acc Lat.	Ant. ID	RSSI	Phase	Frequency	Label

Table 1 - dataset columns

First column provides the time in seconds, next three columns include the Acceleration reading in G for frontal, vertical, and lateral axes. Column 5 provides the id of the antenna that the line corresponds to. Sixth, seventh, and eighth columns are the Received Signal Strength Indicator (RSSI), phase, and frequency. Label of the activity is in column 9 which is in distinct range 1 to 4: 1 for *sitting on the bed*, 2 for *sitting on the chair*, 3 for *lying on the bed*, and 4 for *ambulating*.

The dataset includes 75128 samples, 52482 of which had been captured in room setting 1 and the remaining 22646 had been captures in setting 2. Also, the dataset is imbalanced and 21% of the whole data points are in class 1, 7% in class 2, 68% in class 3, and 4% in class 4.

4- Scalable Solution for Big Data with Spark

Spark is a computing platform to develop applications that run on clusters of computing nodes, and the point to use it is to be able to divide the tasks and data to chunks and compute each result individually. This way we can handle big data. There is very important question we should ask ourselves when trying to develop scalable programs using Spark: What happens when we use libraries and data structures other than Spark's in our code? The answer is straightforward; those sections of our code do not execute in clusters of nodes. More specifically, the only parts of our program that runs on different nodes are the parts using only Spark's libraries and data structures. Whatever else will be executed on a single node. Consequently, if we want to provide scalable code to solve a problem with Spark platform, we need to keep the data and tasks as much as possible in Spark's data structures and mostly use Spark's libraries. This was somehow challenging for us in this project.

Spark currently (version 2.4.4) does not support deep learning algorithms and most of the projects that we see talking about deep learning with Spark are only using Spark for data processing. Therefore, we are limited to use more traditional machine learning algorithms to solve our problem. We talk about them in the following section.

5- Models and Implementation

In this section of the report, we will first introduce the preprocessing tasks we applied to the data, then introduce the models we used and finally discuss the implementation and challenges of our proposed solution.

5-1- Data Preprocessing

As we saw in section 3, the dataset we are using is imbalanced. In order to come up with the most generalizable models we need to balance the data. We used both oversampling and undersampling. Regarding room setting 1, for minority classes 2 and 4, we used oversampling technique and repeated the datapoints to reach 15000 data samples for each. In majority classes 1 and 3, we randomly undersampled the data points to reach 15000 per class. For room setting 2, using the same approach, we obtained 3500 data samples for each class.

The ranges of numerical values of different feature in the used dataset are different; therefore, in order to avoid biasing our models to certain features with larger values and also in order to let our models converge (more in neural network models) to their optimum faster, we need to normalize our dataset. In this project we used MinMax normalization. The important and tricky point we should pay attention to when using this normalization method is not to use test data in creating the model. If we first normalize the data (which is a common mistake) we are using the minimum and maximum values found in the whole dataset which may later be a data point in the test split. Therefore, we first split the data to train and test sets and then apply the normalization to come up with minimum and maximum values only in train set, and use those values in proposed models.

5-2- Models

We developed three different machine learning algorithms for classification for this problem: Logistic Regression, Multilayer Perceptron (MLP) Neural Network, and Decision Tree. We kept the logistic regression model simple and did not use Ridge or LASSO regularization. Having 200 rounds of iterations was sufficient for this model. Regarding the MLP model, finding the best network structure was one of the most challenging parts and it still can be optimized to perform better. The best network structure that we found to work with imbalanced data is having 4 hidden layers with 50, 30, 8, and 6 nodes in these layers respectively and the value of training iterations to be 1000. To work with balanced data, having 4 layers with 10, 9, 8, and 6 nodes in these layers respectively and training for 400 rounds of iterations is selected as the most generalized model in our experiments. According to our problem definition, the decision of classification, logically, seems to be predictable by rule-based AI; that is the reason why we chose our third model to be a decision tree. We configure this model to use Gini impurity with maximum tree depth of 5.

5-3- Implementation

We used Spark Python library in this project. There are different functions written to perform the desired tasks that we will discuss them below.

- ***read_in_np()***
  We initially developed this function to read our dataset files into a comprehensive *numpy* array and finally convert it into a Spark's data frame. Yet, as we said in section 4, this we would not benefit from distributed computing power of Spark and this code would be executed on a single node; therefore, we finally did not use this function.

- ***normalize_data(dataset)***
  In this function, we used Spark's *MinMaxScaler* module to normalize the input dataset. We initialized the module to map data points to range [0, 1] for each feature separately. Then, we fit the scaler model to the input dataset to figure out the minimum and maximum values and finally transform the input data to the scaled version. As the last step, since this transform method adds data column with name "scaled" to input dataset, we renamed it to "features" and removed the previous "features" column before retuning the normalized dataset.
- After defining the mentioned functions, we read the dataset files using Spark's *SQLContext* module in once and we passed the path to balanced and imbalanced data folders to it. Also we let it infer the schema for our dataset and after that we assembled all feature columns into a single "features" vector to have our data ready.
- ***train_test_models(lr, mlp, dtree, acc_eval, f1_eval, trainset, testset)***
  This function is responsible to fit the input models on the trainset and predict and evaluate their results on the testset. As you see, it gets the initialized Spark models for Logistic Regression (*lr*), Multilayer Perceptron Neural Network (*mlp*), Decision Tree (*dtree*), Accuracy Evaluator (*acc_eval*), and F1-Score Evaluator (*f1_eval*) plus the training and test data frames (*trainset, testset*) as inputs. Finally it returns Logistic Regression accuracy (*lr_acc*), Logistic Regression f1-score (*lr_f1*), MLP accuracy (*mlp_acc*), MLP f1-score (*mlp_f1*), Decision Tree accuracy (*dtree_acc*), and Decision Tree f1-score (*dtree_f1*).
- In this step we initialize and configure our base models for the proposed three models by calling their constructor from corresponding Spark libraries. The point to mention is that we used the same random seed for all models (for their potential random parameter initializations) to keep the evaluating methods fair.
- ***cross_validation(lr, mlp, dtree, acc_eval, f1_eval, datadf)***
  In this module we implemented the required steps for evaluating models using 10-fold cross validation. The inputs with the same name are exactly what we described in *train_test_models* function and *datadf* is the Spark's data frame containing the whole dataset. We first randomly shuffle the data lines (again with fixed random seed) using Spark's *orderBy* function, then we split them into 10 groups of semi equal size using Spark's *randomSplit* function. After that we needed to join 9 of the groups to make the trainset and pick one for testing. With the help of Spark's *union* function, and two loop with the summing iterations of size 10, we did so. Then, we normalized the data using the defined function and called the *train_test_models* function to get the results. We summed the evaluation metrics in the 10-fold cross validation loop and finally returned their values divided by 10.
- ***holdout(lr, mlp, dtree, acc_eval, f1_eval, datadf)***
  All the inputs have been introduced in previous functions. So, we jump to explain this function's logic. It first shuffles the dataset, then splits it to 80% of training data and 20% of test data, then normalizes data and calls the *train_test_models* function to get the results and finally returns them.
- Finally we print the evaluation metrics for our models and stop the Spark context.

5-4- Challenges

The first challenge we encountered in this project was to only use Spark's libraries and data structure. This caused our developing time to increase because lots of the logics we wanted to implement were easier using previously known data structures and libraries. For example, developing the splitting of data into trainset and test set in cross validation was challenging and we needed to check different Spark's data manipulation functions to achieve the desired functionality. Keeping all the data processing on Spark's fast data frames was the goal we made our best to achieve. Our insisting on using only Spark's libraries and data structure in our code explanation in section 5-3 is to show the scalability of our solution.

The other challenge that limited us to perform more experiments, was the training time for MLP neural network model specially using the cross validation approach that took around 5 hours per execution for room setting 1. There are lots of possible network structures differing in number of layers and nodes per layer that makes finding the best model very time consuming. For this purpose, we could not even benefit from Spark's *ParamGridBuilder* module because it did not save us any time.

6- Experiments and Results

In this section we will report the evaluation metrics for different models we created to solve the problem. These metrics are accuracy and f1-score. The reason to use f1-score is that it combines the precision and recall and gives a good measurement to evaluate the trade-off between them. We carried out more than 50 different experiments to come up with the best possible models.

First, we start with imbalanced data. Table 2 contains the metrics for imbalanced data of room setting 1 using holdout method, whereas table 3 includes them related to room setting 2.

	Accuracy	F1-score
Logistic Regression	89%	86%
MLP NN	94%	94%
Decision Tree	90%	89%

Table 2 - Evaluation metrics of imbalanced data of room setting 1 with holdout method

	Accuracy	F1-score
Logistic Regression	98%	95%
MLP NN	98%	97%
Decision Tree	98%	98%

Table 3 - Evaluation metrics of imbalanced data of room setting 2 with holdout method

Even though the performance of our models seem to be very good here, but we should pay attention to the point that the data is imbalanced and if the models were to just ignore the minority classes, they cold have achieved the base accuracy of 87% for room setting 1 as we see in following calculations:

# of samples in class 2 = 4381

# of samples in class 4 = 1956

# of all samples in room setting 1 = 52482

Accuracy with only ignoring minority classes = $\frac{52482-(4381+1956)}{52482} \times 100 = 87\%$

Therefore, to evaluate our models better, we balanced the data using the methods described in section 5-1 and achieved the metrics reported in tables 4 and 5 again using holdout method.

	Accuracy	F1-score
Logistic Regression	71%	71%
MLP NN	88%	88%
Decision Tree	78%	78%

Table 4 - Evaluation metrics of balanced data of room setting 1 with holdout method

	Accuracy	F1-score
Logistic Regression	79%	78%
MLP NN	86%	86%
Decision Tree	88%	88%

Table 5 - Evaluation metrics of balanced data of room setting 2 with holdout method

As mentioned before, we see that decision tree model somehow performs well in comparison to logistic regression, because the problem is more of predictable with if-else clauses than predictable with coefficients and regression. Like before, MLP model is overperforming others.

Finally, in tables 6 and 7, we have reported the evaluation metrics regarding cross validation method for room setting 1 for both balanced and imbalanced datasets. This way we can have the best view to judge our models. However, we need to repeat this experiment more and more in future with more randomization to be more fair.

	Accuracy	F1-score
Logistic Regression	83%	83%
MLP NN	86%	87%
Decision Tree	85%	86%

Table 6 - Evaluation metrics of imbalanced data of room setting 1 with cross validation method

	Accuracy	F1-score
Logistic Regression	70%	70%
MLP NN	80%	79%
Decision Tree	77%	76%

Table 7 - Evaluation metrics of balanced data of room setting 1 with cross validation method

According to the cross validation results, we see that our model, specially the MLP model need more optimization to achieve the best possible accuracy. During the experiments this model behaved the most promising.

7- Future Work

As mentioned in previous section, the results for MLP and decision tree models are promising and the first step to continue this work should be carrying out more experiments for potential improvements of these models and preventing them from overfitting and underfitting.

In this project we looked at the first column of our dataset as time stamps. In order to recognize and classify the activity of a person using motion data, it is of more interest to look at data as time sequences. However, we decided that the first step should be developing the straightforward models and analyzing their results to be able continue the research in this topic. The other reason preventing us from performing sequence models to data, other than the available time for the project, was the lack of such models in Spark. The idea we have for doing so in future is to look at a time window of data points and group them together ending up in having more features of the same type but for all the time stamps in the time window to be able to classify the motion in that specific time window.

Finally, as the data donators have confirmed in [2], the data from this dataset is sparse and noisy, because of using a passive sensor for the experiments. Therefore, to continue the research and create better models for the same topic, we may be able to use other available datasets using better experimental tools.

8- Running the code

The developed project with required data is publicly available at GitHub in the following link:

https://github.com/sinamahdipour/activity_detection

The provided code can be easily executed using the following commands:

To run with holdout evaluation: *python3 main.py [-s1/-s2/-s1b/-s2b] -ho*

To run with 10-fold cross validation evaluation: *python3 main.py [-s1/-s2/-s1b/-s2b] -cv*

In the mentioned command one of the -s1, -s2, -s1b or -s2b arguments must be present to indicate which dataset you want to use. They respectively stand for setting 1 imbalanced, setting 2 imbalanced, setting 1 balanced, and setting 2 balanced.

References

[1] University of California Irvine, "UCI Machine Learning Repository," University of California Irvine, [Online]. Available: https://archive.ics.uci.edu/ml/index.php. [Accessed 5 December 2019].

[2] R. L. S. Torres, D. Ranasinghe and R. Visvanathan, "Activity recognition with healthy older people using a batteryless wearable sensor Data Set," UCI, 12 Dec 2016. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Activity+recognition+with+healthy+older+people+using+a+batteryless+wearable+sensor. [Accessed 5 Dec 2019].

[3] S. Torres, Ranasinghe, Shi and Sample, "Sensor enabled wearable RFID technology for mitigating the risk of falls near beds," in *2013 IEEE International Conference on RFID*, 2013.