DATA MINING

Project Report

Bremen Big Data Challenge - Edition 2019

By Gari Ciodaro, Diogo Cosin, and Ralph Florent

May 17, 2019

Abstract

This report proposes a complete data workflow solution for the supervised classification task introduced in the competition *Bremen Big Data Challlenge*. The available data set is composed of signals obtained by wearable movement sensors placed in the leg of human subjects. The task at hand is then to classify 22 different movements evaluating the modeling procedure according to its accuracy, given the sensors signals sampled at 1000 Hz. The training data set is structured in a way in which each row of it is, in fact, a *.csv* subject file composed of 19 (number of sensors) columns and N_{fr} rows, formally speaking, a $[19 \times N_{fr}]$ matrix, for a specific movement class. Two Preprocessing methods are applied to the data set. In the first method, each subject data set, the $[19 \times N_{fr}]$ matrix, is flatten, that is, reshaped into a single vector belonging to $\mathbb{R}^{N_{fr} \times 19}$. The same procedure is applied to every single subject file in the training data set, that is, to the 6401 subjects files. The result is a $[6401 \times (19 * N_{fr})]$ matrix. In the second method, for each of the 19 sensors, the first three statistical moments (mean, variance, and steepness, respectively) are calculated, as well as the first five coefficients of the signals discrete Fourier transformation. After this feature extraction strategy is applied to all 6401 subject files, a $[6401 \times 8 * 19]$ matrix is obtained. Due to the high dimensionality characteristic of the data set, feature extraction methods are implemented to work around this issue in the Preprocessing method 1. Principal Component Analysis and Binary Logistic Regression are implemented in this case. After the extraction phase, a variety of feed forward neural networks are applied using TensorFlow differing in the topology adopted. Two modeling procedures are then applied to obtain test accuracy results of 84.44% and 91%. The result obtained with the Challenge data set, that is, a completely new data set assessed only during submission time, was 54% and 64%. Some conclusions are offered in our report trying to figure out the reasons for the inferior models' performances in the submission data set opening space for different approaches in future works.

1 Introduction

"The Bremen Big Data Challenge" is a competition organized on a yearly basis by the Universität Bremen in Bremen, Germany. This is a challenge in which a data set is provided and a Big Data task, specified. In this year's edition, teams were supposed to come up with a solution to a classification task. According to the data collected by different sensors positioned in the leg of human subjects, teams should classify various leg movements. The team with the best *Accuracy* is then consecrated the competition winner.

This report describes the data workflow implemented to solve the task proposed in *"The Bremen Big Data Challenge"*. We begin by describing the data set structure. Posteriorly, a brief theoretical review is introduced for the concepts involved in the workflow. The data pipeline is then presented. The strategies adopted in the Data Preprocessing as well as the Data Exploitation are presented. Finally, the results obtained are revealed and discussed.

2 Background of the Data

2.1 Data Source

Provided by *"The Bremen Big Data Challenge 2019" Organizers*, the collected data are based on daily athletic movements [1]. Using wearable sensors above and below the knee (See Figure 1) of the individual (athletic), a dataset of 19 individuals, mainly identified as *subjects*, has been recorded. And as the competition requires, the data of 15 out of the total number of subjects are used as the training dataset and the remaining part as the testing dataset. The dataset is publicly available online on the official website: BBDC or by simply browsing through the following URL: *https://bbdc.csl.uni-bremen.de/index.php/2019h/28-aufgabenstellung-2019.*

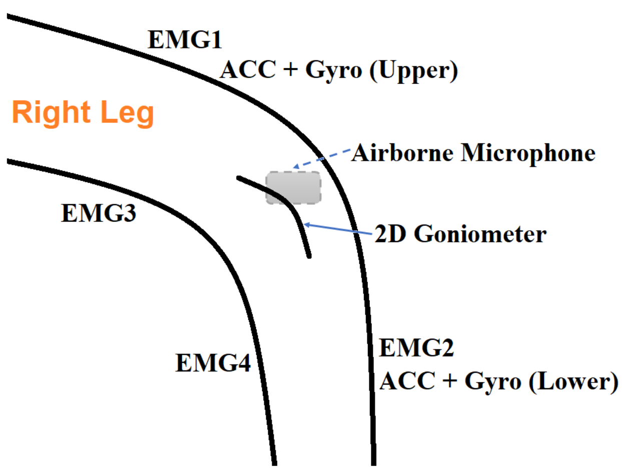

Figure 1: Wearable sensor placement for data measurements.
(source: part of the provided data)

2.2 Description and Format

The data comprise the following 22 movements:

- Race ('run')
- Walking ('walk')

- Standing (standing)
- Sitting ('sit')
- Get up and sit down ('sit-to-stand', 'stand-to-sit')
- Up and down stairs ('stair-up', 'stair-down')
- Jump on one or both legs ('jump-one-leg', 'jump-two-leg')
- Run left or right ('curve-left-step', 'curve-right-step')
- Turn left or right on the spot, left or right foot first ('curve-left-spin-Lfirst', 'curve-left-spin-Rfirst', 'curve-right-spin-Lfirst', 'curve-right- spin-Rfirst ')
- Lateral steps to the left or right ('lateral-shuffle-left', 'lateral-shuffle-right')
- Change of direction when running to the right or left, left or right foot first ('v-cut-left-left', 'v-cut-left-right', 'v-cut-right-left', 'v-cut' right-Rfirst ')

The entire data are available as CSV files, or Comma-Separated Values, and partitioned as training and testing data, respectively represented by the "***train.csv***" and the "***challenge.csv***" files. Starting with the training dataset file (***train.csv***), it contains *UTF-8*[1] character-encoded, line-wise plain texts, whose first line identifies the feature names followed by the feature values. This file contains a total of 6402 lines, which include both the feature names and the feature values. The feature names are *Subjects*, *Datafile*, *Label*, and the feature values map respectively each feature name. For instance, the first feature values of the file are: *Subject02*, `Subject02/Subject02_Aufnahme002.csv`, *stand-to-sit*. Table 1 illustrates a lightweight version of the data partition of the training dataset file.

Subjects	**Datafile**	**Label**
Subject02	`Subject02/Subject02_Aufnahme000.csv`	*curve-left-step*
Subject02	`Subject02/Subject02_Aufnahme001.csv`	*curve-left-step*
Subject02	`Subject02/Subject02_Aufnahme002.csv`	*stand-to-sit*
...	...	...
Subject19	`Subject19/Subject19_Aufnahme438.csv`	*curve-right-step*
Subject19	`Subject19/Subject19_Aufnahme439.csv`	*curve-right-spin-Rfirst*

Table 1: Tabular visualization of the "***train.csv***" dataset

Similarly, the testing dataset file (***challenge.csv***) is formatted using the same structure except for the *Label* column, which is unknown and marked with an *X*. The datafile contains a total of 1739 lines counting both the feature names and feature values. Table 2 displays a lightweight version of the data partition of the testing dataset file.

Important: Recalling that the dataset is divided into training and testing data, the subjects "*Subject01, Subject10, Subject14, Subject15*" are the selected ones that are used as testing data to assess the solutions. Note the difference in the starting and ending rows of Tables 1 and 2.

As observed in both Tables 1 and 2, each line corresponds to a recording of a movement. The columns have the following meanings:

- ***Subject***: The ID of the subject
- ***Datafile***: Path of the file containing the sensor data for this recording. For each subject, there is a folder in which individual data files contain the sensor data for individual motion recordings.

[1] Unicode Transformation Format, extended ASCII, variable-width encoding.

Subjects	**Datafile**	**Label**
Subject01	`Subject01/Subject01_Aufnahme000.csv`	X
Subject01	`Subject01/Subject01_Aufnahme001.csv`	X
Subject01	`Subject01/Subject01_Aufnahme002.csv`	X
...	...	...
Subject15	`Subject15/Subject15_Aufnahme438.csv`	X
Subject15	`Subject15/Subject15_Aufnahme439.csv`	X

Table 2: Tabular visualization of the "***challenge.csv***" dataset

- ***Label***: The movement that was recorded

Notably, the *Label* column of the testing dataset contains repeatedly the letter "X" to indicate that this value is not present. That is, at the time of submitting solutions, the submission should exactly match the testing data, where each X will be replaced by a label. This label corresponds to the classification result of a specific movement. It is important that the spelling (including upper / lower case) of the textual names matches precisely the spelling of the labels in the training data.

As mentioned above, the datafiles are references to other CSV files. For example, the path file `Subject02/Subject02_Aufnahme000.csv` is a CSV file within a folder named *Subject02* located in the root path (i.e., the current directory of the downloaded zip files). The CSV file itself is a dataset with a proper format. Basically, the file has a set of comma-separated numbered-values that looks like this: *32688, 32224, 32991, 32609,32790,33048, 37168, 34610, 27374, 29068, 29264, 28408, 31784, 28133, 29295, 29244, 33216, 37140, 34736.*

Each line has 19 values (numbers) and represents the sensor values measured at one time (sampled at 1000 Hz). In other words, the columns represent the individual wearable sensors recording the human activities (see Figure 1):

EMG1	Airborne	Goniometer X	Goniometer Y	Gyro lower X
EMG2	ACC upper X	ACC lower X	Gyro upper X	Gyro lower Y
EMG3	ACC upper Y	ACC lower Y	Gyro upper Y	Gyro lower Z
EMG4	ACC upper Z	ACC lower Z	Gyro upper Z	

The size of the CSV datafiles varies inappropriately. That is, in most cases, due to inaccurate measurements, random initialization states, mechanical flaws, computational and processing cost, and so on.

3 General Comments

3.1 Notation

Let us define a modeling procedure as a function $\mathcal{P}(\Theta) : \mathbb{R}^m \longrightarrow \mathbb{R}^k$ where m is the number of features, k is the number of classes, and $\Theta : \{Preprocessing_{method}, Feature_{extration}, Statistical_{Tecnique}\}$ is a set representing parameters. Giving this abstraction, Θ modifies the structure $\mathcal{P}$ but always takes a *feature vector* $X \in \mathbb{R}^m$, and returns a *Probability vector* $Y \in \mathbb{R}^k$ where each component $\in [0, 1]$.

It is clear that the $\mathcal{P}(\Theta)$ that represents exactly the reality regarding athletics movements is unknown to us, which let us to defined a measure of the amount of veracity that a giving $\mathcal{P}$ has compared to reality.

$$accuracy = \frac{\sum Correct_{classification}}{N} \tag{1}$$

3.2 Cross Validation

Given the modeling procedure $\mathcal{P}$ defined in the Subsection 3.1 for a given learning task, one usually seeks to evaluate its performance in the so-called training data and testing data. The training data is not but the set of data points utilized during the training procedure, whereas the testing data is a completely new data points set. Ideally, an efficient modeling procedure $\mathcal{P}$ will fit the training data and also generalize well to new data. However, during training time, only training data is available. In this sense, the statistical method *Cross validation* can be applied to estimate training and testing error even though only training data is available. This way, it is possible to estimate if the model is overfitting or underfitting using only the training data set [2].

Despite being simple, Cross validation is still a powerful method that can be applied to estimate the testing error. The simplicity is in the process. The Training data set is artificially split into two subsets. One partition is then used as the training data while the other one is used as the testing data [2].

Apart from this more straightforward data partition strategy, one may also split the data set into K partitions, more commonly called *folds*. The process follows by holding one the of *folds* as the testing data set while the remaining K - *1 folds* are aggregated as the training data set. The process is repeated K times given that each partition is utilized as the testing data set exactly once. The model is then assessed by taking the average of the results obtained during the K repetitions. The optimal model will perform well in the testing and training data, avoiding this way overfitting and underfitting, respectively. In more educated terms, it will present low variance when the different K training data sets are introduced by the cross validation, and low bias if it fits the training data set efficiently [2].

3.3 Curse of Dimensionality

The curse of dimensionality is a common machine learning problem. It happens when the dimension of the *input vetors*, m, is much higher than the number of data points N. In this case, the data points will be highly sparsed distributed in the m-dimensional coordinate system defined by the vector space spanned by the linear combination of the *input vectors*. In other words, the data points will present large distances from each other. Consequently, the modeling procedure $\mathcal{P}$ will have difficulties in finding similarities between the highly sparsed data points [2].

A solution for this curse is to reduce the dimensionality of the data set by applying feature extraction functions such as Principal Component Analysis and *K-means* [2]. The idea in this stage is to reduce the data points dimension without highly compromising in loss of information. A rule of thumb practically applied is to respect a ratio of 10 features per data point [2].

4 Data Preprocessing

Two *Preprocessing methods* procedures were implemented.

Preprocessing method 1:

1. Take a **subject file** (each file contains a class movement). It can be viewed as $[19 \times N_{fr}]$ matrix composed by 19 columns vectors $S_{data} \in \mathbb{R}^{N_{fr} \times 1}$ where the number of records in file is $N_{fr} \in \mathbb{N}^{>0}$.

2. Transpose each S_{data} into a row vector, concatenating them into one single vector $S_{concat} \in \mathbb{R}^{1 \times 19 * N_{fr}}$

3. Repeat step 1 and 2 for every subject file.

4. Create a dataset with the rows of step 3 and the corresponding label (extracted from the name of the file). This data set is a matrix D of dimensions $[6401 \times (19 * N_{fr})]$. For $N_{fr} = 56810$ D is $[6401 \times 1079390 + 1]$.

Preprocessing method 2:

1. Take a **subject file**(each file contains a class movement). It can be viewed as $[19 \times N_{fr}]$ matrix composed by 19 columns vectors $S_{data} \in \mathbb{R}^{N_{fr} \times 1}$, where number of records in file is $N_{fr} \in \mathbb{N}^{>0}$.

2. For each S_{data}, decompose it as $s_{data} = \mu + \omega$, where μ is a smoothed version of s_{data} calculated using lowess with tree points average weighted linear regression. A complete derivation of this algorithm can be found in [3]. Having s_{data} and μ, calculate $\omega = s_{data} - \mu$. Create a vector $\mu_{sta} \in \mathbb{R}^3$ with first tree statistical moments of μ, that is, the average, the variance, and the steepness. Calculate the discrete Fourier transformation of ω, extract the first five coefficients and arrange them in a vector $\omega_{fft} \in \mathbb{R}^5$. A complete derivation of this algorithms can be found in [4]. Take μ_{sta} and concatenate it with ω_{fft} into a row vector $S \in \mathbb{R}^8$.

3. Concatenate each S into a single vector $S_{row} \in \mathbb{R}^{8*19}$.

4. Repeat steps 1 to 3 for every **subject file** and stack the vectors S with its corresponding label into a matrix D is $[6401 \times 8 * 19]$.

For the $Preprocessing_{\ method\ 1}$ the parameter, N_{fr} had to be set, since each **subject file** had different number records. By observing a 100 random sample of files, we concluded rather arbitrarily that $N_{fr} = 56810$ was a reasonable number of records. In case a particular file did not meet this requirement, the signal per sensor would repeat itself until the desired N_{fr} was reached. The idea with $Preprocessing_{\ method\ 2}$ was to capture the general properties of the movement(using μ_{sta}) in terms of its statistical moments. On the other hand, we also attempted to capture information about the periodicity of the movement using ω_{fft}.

5 Data Exploitation

Depending on the $Preprocessing_{method}$, some $Feature_{extration}$ and $Statistical_{Tecnique}$ combination would be more adequated to implement than others. Here we only present the highest *accuracy* combinations.

5.1 Data Exploitation with $Preprocessing_{\ method\ 1}$

One problem that is immediately observed with $Preprocessing_{\ method\ 1}$ is the immense dimensionality of the sample space. Therefore, trying to find a $Statistical_{Tecnique}$ capable of learning an adequate decision function is basically impossible. To bypass this problem, we implemented a $Feature_{extration}$ and $Statistical_{Tecnique}$ shown in the Figure 2.

We theorized that the importance of features in matrix D measured by its variance will strongly depend on the particular movement involved, acknowledging this, we filtered matrix D by class, and applied principal component analysis $PCA_{|class} : \mathbb{R}^{1079390} \longrightarrow \mathbb{R}^{275}$ to extract the first 275 principal components (this transformation will be the beginning of a branch in Figure 2) that accounts to 90 % of the variance. Note that the first level in the Figure 2 has 22 $PCA_{|class}$ extractors. For training, matrix D is passed without filtering per label to each $PCA_{|class}$ and then feed to a *Binary logistic regression* $LogR_{|class} : \mathbb{R}^{275} \longrightarrow [0, 1]$, where the classes are encoded in *one vs all* manner. Each of the $LogR_{|class}$ will output a degree of believe that a particular X belonges to a class k. Finally, concatenating the degree of belief predicted by each $LogR_{|class}$, we construct a $X' \in \mathbb{R}^{22}$. Finally, X' is passed to a single feed forward neuronal network $NN : \mathbb{R}^{22} \longrightarrow \mathbb{R}^{22}$ that predicts our final probability vector Y in a *hot encoded* structure. We take the maximum component and assign its corresponding label to that observation.

$$\Theta_1 := \{Preprocessing_{\ method\ 1}, PCA_{|class}, LogR_{|class}, NN\} \tag{2}$$

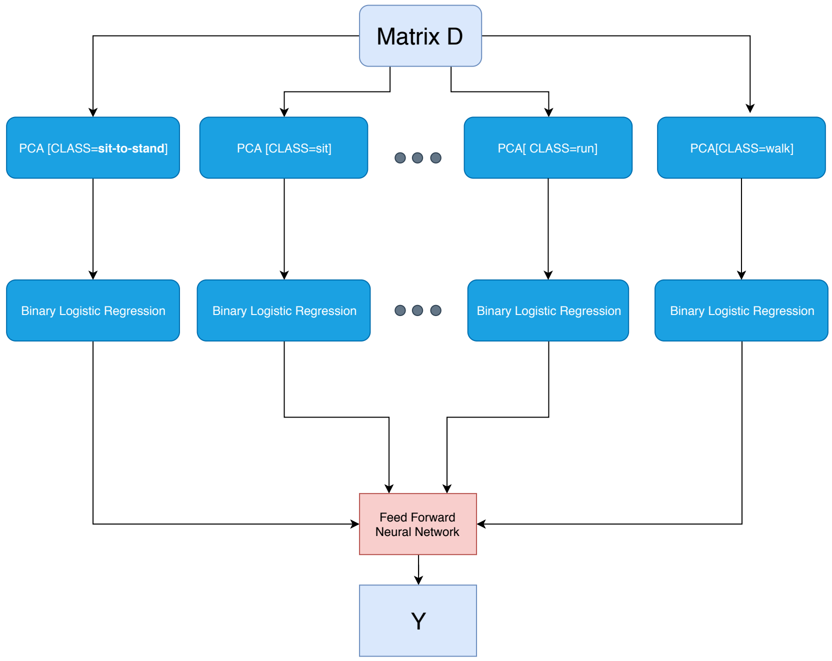

Figure 2: $\mathcal{P}(\Theta_1)$ Scheme

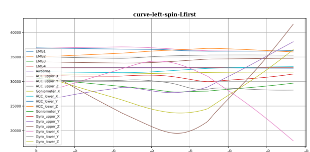

Figure 3: curve-left-spin-Lfirst

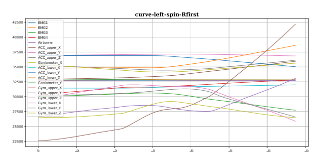

Figure 4: curve-left-spin-Rfirst

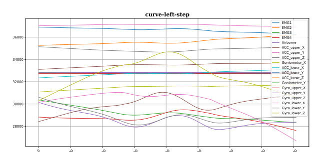

Figure 5: curve-left-step

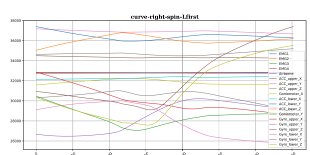

Figure 6: curRight-spin-Lfirst

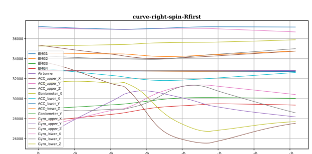

Figure 7: curRight-spin-Rfirst

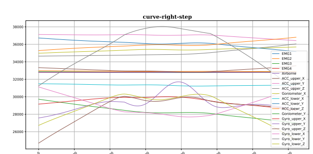

Figure 8: curRight-step

5.2 Data Exploitation with $Preprocessing_{\ method\ 2}$

The dimension of X using $Preprocessing_{\ method\ 2}$ is R^{152} so no further dimension reduction is required. We implemented a variety of feed forward neural networks using TensorFlow, where we varied the Network architecture composed by the number of hidden units $H_{units} \in \mathbb{N}^{>0}$ per hidden layer $H_L \in \mathbb{N}^{>0}$ the regularization parameter $\alpha \in \mathbb{R}^{>0}$, and the activation functions, namely $\{Logistic, tanh, relu\}$. From Figures 3 to 25 we can observe μ signal examples per label.

$$\Theta_2 := \{Preprocessing_{\ method\ 2}, NN\} \tag{3}$$

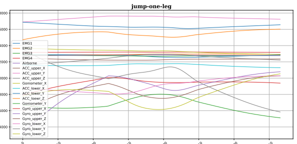

Figure 9: jump-one-leg

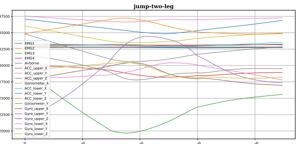

Figure 10: jump-two-leg

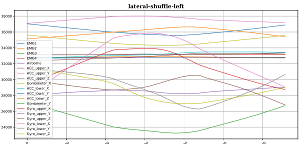

Figure 11: lateral-shuffle-left

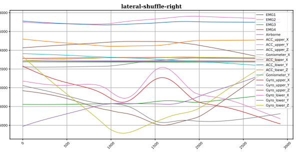

Figure 12: lateral-shuffle-right

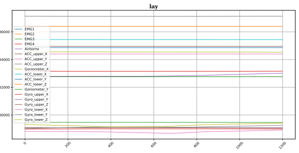

Figure 13: lay

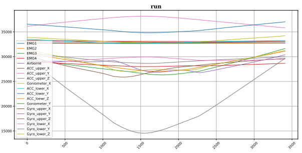

Figure 14: run

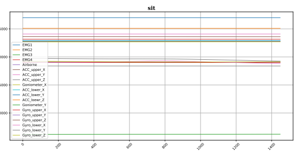

Figure 15: sit

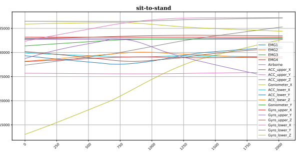

Figure 16: sit-to-stand

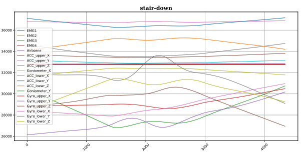

Figure 17: stair-down

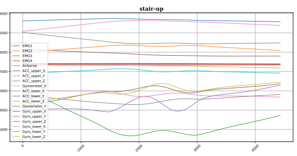

Figure 18: stair-up

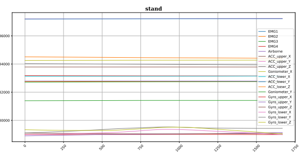

Figure 19: stand

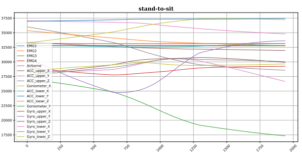

Figure 20: stand-to-sit

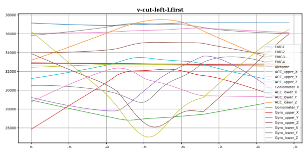

Figure 21: v-cut-left-Lfirst

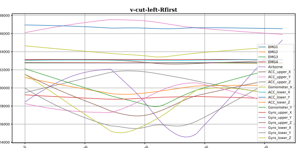

Figure 22: v-cut-left-Rfirst

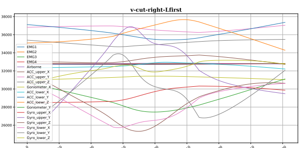

Figure 23: v-cut-right-Lfirst

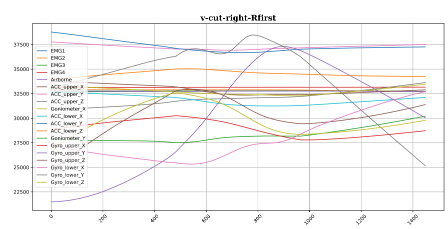

Figure 24: v-cut-right-Rfirst

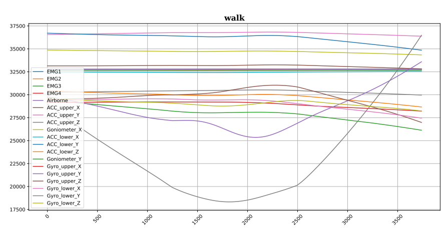

Figure 25: walk

6 Data Analysis

6.1 Data Analysis with $\mathcal{P}(\Theta_1)$

The last procedure to complete the full scheme in Figure 2 is the implementation of the *Fully-Connected Neural Network (FCNN)*. Applying this *Deep Learning* approach on the preprocessed data serves as an alternative technique to optimize the loss function and increase accuracy.

When implementing FCNN using *Pandas* [5] and *Scikit-Learn* [6], there are several specifications to account for. Among them figure the *activation function*, the regularization factor *alpha*, and the number of neurons per *ith* hidden layer. To briefly summarize some parameters, as per relevance or rule of thumbs, we describe their optional values:

- The solver for the weight optimization ('lbfgs', 'sgd', 'adam')
- The size of hidden units ranging between 100 ± 20 for *ith* hidden layer
- The regularization factor varying between 0.1 and 0.00001[2]
- The activation function for the hidden layers ('identity', 'logistic', 'tanh', 'relu')

Given the preprocessed data, whose order of magnitude scales to 1000+ data points, the default parameters reveal themselves ideal for testing out first. Then, based on the obtained results, we would assess the accuracy of the prediction by tuning the other parameters (e.g. alpha) accordingly with the expectations of optimizing the classifier and improving the accuracy.

The settings for the default values when running the MLPClassifier are:

Solver	**Activation Function**	**Hidden Units**	**Alpha**
lbfgs	*relu*	*50*	*0.01*

Surprisingly, both the training and testing errors score an error rate of 83.7%. Hoping to find the parameters that work best for the task optimization, we performed *Grid Search* by combining above-mentioned options and tweaking them in the same range as in our first test. For instance, the chosen values for alpha were $10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$, and 10^{-5}.

Finally, the best parameters set found on the classifier development set are:

Solver	**Activation Function**	**Hidden Units**	**Alpha**
lbfgs	*tanh*	*20*	*0.01*

With an improved accuracy of 84.44%.

6.2 Data Analysis with $\mathcal{P}(\Theta_2)$

Giving that the same $Stastistical_{Technique}$, namely *Fully-Connected Neural Network (FCNN)* was used for $\mathcal{P}(\Theta_2)$ the prior section parameters definitions holds the same. However, instead of using *scikit-learn*, we set up an environment where the *FCNN* algorithm was implemented using *Keras* with *TensorFlow* Back-end for *GPU* usage.

6.2.1 Experimental Setup

Training a *FCNN* high dimensional dataset D $[6401 \times 153]$ is computationally expensive(in fact this is the reason we were force to use *keras*). So trying a *cross-validation* plus *grid-search*, as in $\mathcal{P}(\Theta_1)$, is not practically possible. We used instead a simply 75% and 25% train-test split, and randomly tested different *FCNN* architectures. We also apply min-max scale so that *FCNN* was trained using features $\in [0, 1]$ interval, for increasing the convergence time.

[2]Practice shows that by varying alpha the classifier has greater chance to perform way better than by adjusting the number of units per layer, or trying other algorithms.

7 Results and Discussions

7.1 Results and Discussions for $\mathcal{P}(\Theta_2)$

$H_{L1}(H_{units}, fun)$	H_{L2}	H_{L2}	α	$Train_{Score}$	$Test_{Score}$	$Challenge_{score}$
60,relu	87,tanh	23, sigmoid	0.000001	95%	91%	64%

Table 3: Resuts on $\mathcal{P}(\Theta_2)$

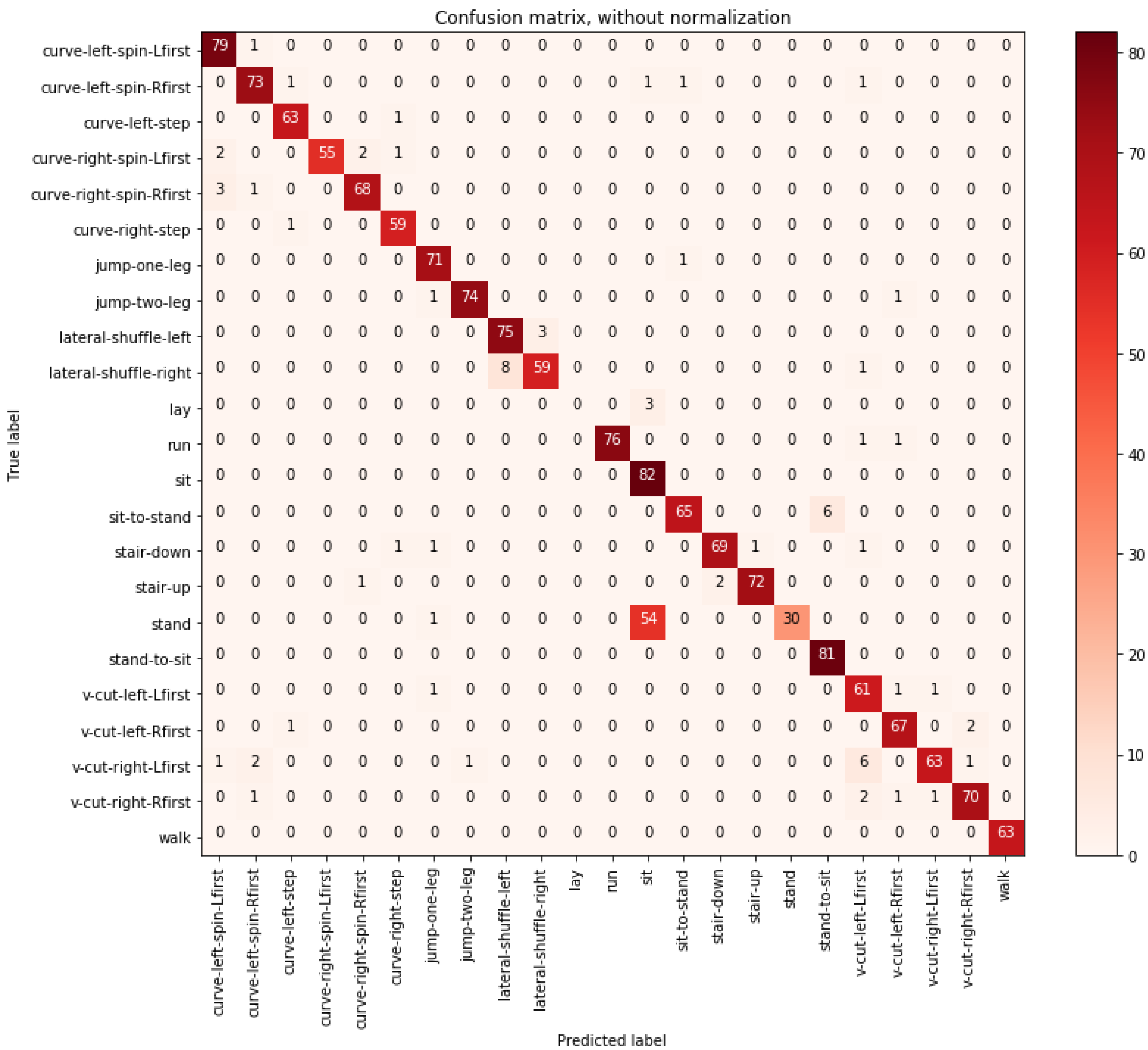

Figure 26: $\mathcal{P}(\Theta_2)$ Confusion matrix on test data.

From Figure 26, we can conclude that most misclassification occurs with the classes *stand* and *sit*, which intuitively are mechanically similar. This can be further confirmed by looking at the smoothen (μ) plots 15 and 19. We tried to modify the $Preprocessing_{\ method\ 1}$ to include some manually crafted features that will indicate a clear difference between *stand* and *sit*. They can be summarized as follows:

- The first and second derivate of μ, since we believed the direction of the movement had to be different.
- The maximum value of the μ, since sensor data related to distance from the floor must be different for both classes.
- increase to five the number of statistical moments.

Unfortunately, none of them presented better results on the test data. The main reason being that this particular adjustment had to be me made to all features. This caused other features to get noise. As a result, the decrease in misclassification gained in the pair *stand*, *sit* was undermined by the increase of misclassification on other features, particularly *sit-to-stand* and *stand-to-sit*.

7.2 Results and Discussions for $\mathcal{P}(\Theta_1)$

In Table 4, we can observe that the Challenge score is inferior to that reached on $\mathcal{P}(\Theta_2)$. Since this was the case, we focused our efforts of optimization on $\mathcal{P}(\Theta_2)$.

H_{units}	Activation function	α	CV_{train}	CV_{test}	Challenge score
20	tanh	0.01	(??)	84.44%	54%

Table 4: Resuts on $\mathcal{P}(\Theta_1)$

8 Conclusion

8.1 Conclusion for $\mathcal{P}(\Theta_1)$

- In retrospective, we realized that we introduced some *data leakage* on this implementation, since, while constructing $PCA_{|class}$, we feed the algorithms with the whole dataset. This could be an explanation for the difference in performance on our CV_{test} and challenge score.
- In $Preprocessing_{\ method\ 1}$ we arbitrarily choose the number of time steps to be 56810 per signal. We believed that this decision should be made with more empirical knowledge of the phenomena to increase the performance of $\mathcal{P}(\Theta_1)$.
- We also implemented *Boosting* with *random forest* with 4% increase in CV_{test}. However, the competition had already finished and we could not test the result on the challenge data.

8.2 Conclusion for $\mathcal{P}(\Theta_2)$

- After the competition finished, we realized that one reason for the difference in performance between our test and the challenge score might have been that the splitting of our data was done randomly. We believe that setting aside entire subjects would have been a better approach, after all, an individual should walk and run in a similar manner.
- In order to increase the performance of $\mathcal{P}(\Theta_2)$, we believe that a window sampling the signal S_{data} should be implemented using expert knowledge on the subject. As a side note, we tried the latest by splitting S_{data} into 3 sectors and then by repeating $Preprocessing_{\ method\ 2}$. Unfortunately, this is very computationally expensive, and, for this reason, the attempt had to be suspended after days of processing time.

Appendix A Code Repository

All the code implemented and utilized during the execution of the data workflow described in this report is available at the GitHub repository https://github.com/ralflorent/bremen-big-data-challenge-2019.

References

[1] University of Bremen. *The Bremen Big Data Challenge 2019*. May 2019. URL: https://bbdc.csl.uni-bremen.de/.

[2] Herbert Jaeger. *Machine Learning Lectures Notes*. Jacobs University Bremen, Mar. 2019. URL: http://minds.jacobs-university.de/uploads/teaching/lectureNotes/LN_ML4IMS.pdf.

[3] William S Cleveland. "Robust locally weighted regression and smoothing scatterplots." In: *Journal of the American statistical association* 74.368 (1979), pp. 829–836.

[4] Soontorn Oraintara, Ying-Jui Chen, and Truong Q Nguyen. "Integer fast Fourier transform." In: *IEEE Transactions on Signal Processing* 50.3 (2002), pp. 607–618.

[5] Open Source community. *pandas.Dataframe*. Mar. 2019. URL: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html.

[6] The Scikit-Learn community. *sklearn neural network MLPClassifier*. Apr. 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.