Using off-the-shelf data-human interface platforms: traps and tricks

Noname manuscript No.
(will be inserted by the editor)

Using off-the-shelf data-human interface platforms: traps and tricks

Alessia Angeli · Gustavo Marfia · Norman Riedel

Received: date / Accepted: date

Abstract With the development of learning algorithms, the constantly increasing computing power and the available amount of multimedia data, the adoption rate of data science techniques is steadily growing. Machine and deep learning algorithms are already used in a wide variety of ways to solve domain-specific problems. However, the potential of such methodologies will be fulfilled when also non specialized data scientists will be empowered with their use. Focusing on such perspective, this work does not deal with a classical data science problem, but instead exploits existing and available easy to use data-human interfaces. To this aim, we picked an exemplar scenario, amounting to an existing qualitative activity recognition data set that was in the past analyzed utilizing feature selection techniques and custom machine learning paradigms. We here verify how it is today possible, without changing the default settings and/or performing any type of feature selection, to employ the machine and deep learning algorithms provided by different publicly accessible tools (namely, Weka, Orange, Ludwig and KNIME) to address the same problem. Nevertheless, not all of the utilized platforms and algorithms provided satisfactorily results: we here finally discuss the possible issues and opportunities posed by such approach.

Alessia Angeli
Department of Computer Science and Engineering, University of Bologna,
Bologna, Italy
E-mail: alessia.angeli2@unibo.it

Gustavo Marfia
Department for Life Quality Studies, University of Bologna,
Bologna, Italy
E-mail: gustavo.marfia@unibo.it

Norman Riedel
University of Bielefeld,
Bielefeld, Germany
E-mail: norman.riedel@uni-bielefeld.de

Keywords Data-human-interface, machine learning, deep learning

1 Introduction

Today, machine and deep learning algorithms are used in many daily life applications, often without end users understanding or even noticing them [35, 46, 23, 27, 17, 16]. Data science experts are usually hired to apply the most appropriate of such algorithms to the specific context of use [69, 8]. This clearly limits the use of such techniques, as well as the study and assessment of the benefits these may provide to a plethora of application domains. For this reason, for several years, researchers have tried to lower their adoption barrier [56, 37, 36, 9]. Now, recent developments are making such algorithms accessible to non-experts through simplified data-human interfaces [29, 72, 52, 30].

For instance, one of the fields that could greatly benefit from such developments is the field of qualitative activity recognition. Qualitative activity recognition studies aim at determining the quality of execution of a given movement, rather than recognizing the movement itself (which may have been implemented at an earlier stage of the process or simply be known a priori). With this knowledge it is possible, for example, to give (real-time) feedback concerning the quality of an executed movement, an essential information in physical rehabilitation, in sports and in many industrial domains, among others [44, 49, 40, 18, 67].

For this reason, for this study, we resorted to a publicly available qualitative activity recognition data set, containing raw data and calculated parameters obtained from inertial measurement units (IMUs) [68]. The IMUs were attached to athletes while performing a weightlifting exercise with dumbbells. The aim of the researchers that created and first studied such data set was to devise a machine learning model capable of identifying the correct way of lifting dumbbells, as well as four typical mistakes. To this end, they exploited a feature selection approach and a custom model for the classification of the weightlifting movements.

In this paper, hence, we analyzed the data set created by the authors of [68], adopting an approach based on the use of a few of the most prominent simplified data-human interface platforms, namely:

- **Weka**, "Waikato Environment for Knowledge Analysis", developed at the University of Waikato in New Zealand since 1993 [1];
- **Orange**, developed at the University of Ljubljana since 1996 [2];
- **Ludwig**, a deep learning software released in early 2019 by Uber [7];
- **KNIME**, initially developed at the University of Konstanz, which specializes in pharmaceutical applications [3].

In particular, this work extends [60] providing a holistic approach to study and analyze the different possibilities available to a non-expert user to apply machine and deep learning algorithms, through learning platforms and tools, in relation to a specific task. In addition, this work includes a taxonomy of

the considered learning platforms with the aim of aiding the non-expert user with a map of the strengths and weaknesses of such tools.

Our contribution amounts to the verification of the ease-of-use of such platforms, as well as the performance of the machine and deep learning algorithms they provide. This is simply done by putting such interfaces to good use, without changing the default model settings and/or performing any type of feature selection. As a final term of comparison, we also exploited the use of the deep learning libraries provided by an interpreted programming language, namely, Python [4]. In summary, this work provides:

- An assessment and comparison of the results obtainable by non-data scientists when resorting to easy-to-use, off-the-shelf, visual and non-visual based data science platforms applied to a specific and well investigated problem;
- A verification of the possible pitfalls a non-data scientist could fall on with the use of the considered platforms;
- A final analysis of the chosen data set, which serves as a further term of comparison, based on the use of mainstream Python-based learning libraries.

This work is organized as follows. In Section 2 we review the most relevant literature for this work, including data-human interfaces (Subsection 2.1) and activity recognition (Subsection 2.2). Section 3 introduces the platforms and the programming language that have been here employed as user-data interfaces and presents the results obtained with the different approaches. Finally, in Section 4 and in Section 5 we summarize the findings and draw conclusions.

2 Related Work

In this Section we review a sample of the most relevant scientific works concerned with the importance of easy-to-use data-human interfaces (Section 2.1), as well as a few of those related to the domain of human activity recognition (Section 2.2).

2.1 Data-human interfaces

The ever increasing amount of multimedia data is leading many non-experts in data science to face the problem of diagnosing and solving problems with the use of machine and deep learning [26, 20, 33, 65, 12, 15, 70, 31]. To support such needs, the research community has studied and developed platforms which aim at easing the use of the existing models and algorithms by non-experts. In the following, we present three different contributions, alternative to those exploited in this work, where the problem of empowering non-data scientist with adequate data analysis tools has been considered.

In particular, Patel et al. (2010) discussed the difficulties of applying machine learning algorithms for non-data scientists, pointing out the need for tools that could let a wider community of developers effectively use them [55]. To pursue such direction of research, the author created Gestalt, a prototype integrated development environment. Gestalt provides an explicit support for connecting the principal steps in a pipeline of a machine learning algorithm and an interactive graphical interface through which developers can quickly sort and filter examples to drill down into the data they need.

Yang et al. (2018) investigated how non-experts build machine learning solutions depending on the problem they encounter [71]. The authors concluded that a machine learning tool for non-experts should be both easy to use and robust, even though it is challenging. To advance on this insight, the authors discussed design implications and created a concept to demonstrate how designers might guide non-experts to easily build robust solutions.

In [19], Chen et al. (2016) studied how visualization could support users to utilize machine learning. To this aim, they developed a visual interface to engage a group of users in a design exercise, exhibiting the need for visualizations allowing machine learning practitioners to engage in iterative cycles of exploratory observation, hypothesis formation and testing.

Unlike the works discussed in this Section, the present contribution considers different visual and non-visual programming approaches, aiming at showing where critical issues may emerge in the process of approaching data science with off-the-shelf solutions. Considering a real-world scenario, hence, we exhibit the potential and the possible problems that may arise when employing a specific set of algorithms with the individuated platforms.

2.2 Human Activity Recognition

Human Activity Recognition (HAR) is an emerging field of research in pervasive computing and human-computer interaction, due to the enormous improvement of sensor technologies and the constantly increasing computing power. It aims at recognizing human activity based on the data obtained from different sensor sources, such as video cameras, wearable sensors (e.g., accelerometer, magnetometer, gyroscope) or ambient sensors (e.g., radar, sound sensors, pressure sensors, temperature sensors) [22]. Many works have so far explored this approach, utilizing both supervised and unsupervised learning methodologies [68, 60, 70, 61, 54, 21, 43, 10, 57, 32]. HAR has already become part of our everyday life when we think, for instance, of smartphones or smartwatches that are capable of detecting basic activity states with the help of a simple built-in accelerometer [61, 14, 41]. The application areas of HAR are manifold, including the recognition of daily life activities [13, 59, 47], the assessment of skill and performance in sports [68, 66, 39, 42], the monitoring of long-term health conditions for disease diagnostics [45, 64, 26, 20, 58] and training of personnel in industrial and maintenance processes [50, 63, 49].

A number of researches has focused on a qualitative assessment of human activities. The works in this area are more concerned with *how* an activity is performed, rather than with *which* activity was performed.

For instance, Ladha et al. (2013) developed the skill assessment platform ClimbAX for climbing using tri-axial accelerometers [42]: analyzing the acceleration patterns of competitive climbers, they were able to find a correlation between the sensor-data predicted scores, based on performance attributes derived from the raw acceleration data, and the competition scores.

Khan et al. (2015) proposed a generalised skill assessment framework using a hierarchical and stochastic rule induction method [38]. The framework was tested in the context of surgical skill assessment of medicine students.

In [68], instead, Velloso et al. (2013) investigated the feasibility of automatically assessing the quality of the execution of weightlifting exercises. Four Inertial Measurement Units (IMUs) were used to track the motion of a unilateral dumbbell biceps curl (see Figure 1). The participants were asked to perform one set of 10 repetitions of the biceps curl in five different ways: correctly (A) and incorrectly, according to four different but ways common mistakes (B, C, D, E). The gathered data were used as a training data set for the machine learning algorithm (in this case a 10-fold Random Forest). The authors processed the raw data (sampled at 45 Hz) provided by the 4 sensors, calculating 8 features per each, in correspondence to each curl movement: mean, variance, standard deviation, max, min, amplitude, kurtosis and skewness, generating in total 96 derived features. In a second step, utilizing a correlation-based feature selection algorithm, they determined the 17 most relevant features necessary to describe each curl repetition.

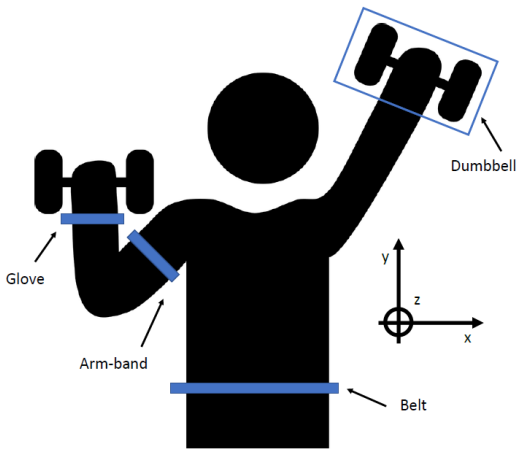

Fig. 1 IMUs setup by Velloso et al. [68].

Building on the raw data set published in [68], a different approach has been adopted compared to the works discussed in this Section. Taking a non-expert data scientist perspective, only raw data is used. In fact, unlike the previously discussed approach, no further analysis of the data set and no feature selection algorithm has been implemented. The sensor data has been considered as provided. The next Section describes in detail how the data set has been put to good use without requiring specific data science expertise.

3 Experimenting with user-data interfaces and models

The data set provided in [68] contains the raw data of the Inertial Measurement Units (IMUs) of its six monitored participants, the extracted features and other additional information such as the participants' names (for a total of 158 features) and the specification related to the feature "classe" (values: A, B, C, D, E). The `CSV` formatted file consists of 39,243 rows, with the variable description in the first row and the raw data in the remaining rows, and 159 columns.

Unlike Velloso et al. (2013) [68], who used a sliding window approach to extract their features [53], the method described in this work handles the raw data without using a feature selection algorithm. The only step taken, when processing the original data set, was to ignore all extracted features and information that did not directly originate from the sensors, such as information about the participants.

Consequently, only the raw Euler angles (roll, pitch, yaw), the raw data provided by the accelerometer, gyroscope and magnetometer and the total acceleration were used as inputs. This results in a total of 52 input features:

$$4IMUs * (3EulerAngles + (3Sensors * 3Axis) + Acc_{\text{total}}) = 52.$$

In a second run the same procedure was repeated using only the Euler angles and the total acceleration as input features, resulting in a total of 16 input features:

$$4IMUs * (3EulerAngles + Acc_{\text{total}}) = 16.$$

For both settings, the target feature is the feature "classe". The label "A" corresponds to the correctly executed movements, while the other labels "B", "C", "D", "E" correspond to different incorrectly executed movements.

In order clearly portray the results a classifier achieves, we here report a few of the most common quantities which are used in literature for such aim. In particular, considering the different rates of True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN), the following different quantities can be computed:

– *Precision*: is the proportion of positive identifications that was actually correct;

$$Precision = \frac{TP}{TP + FP}$$

– *Recall*: is the proportion of actual positives that was identified correctly;

$$Recall = \frac{TP}{TP + FN}$$

– *Accuracy*: is the ratio of number of correct identification to the total number of input samples;

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

– *F1 score*: is a function of *Precision* and *Recall*, in particular

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

F1 score might be a better measure to use than *Accuracy* if we need to seek a balance between *Precision* and *Recall* and there is an uneven class distribution.

– *ROC curve* and *AUC*: an *ROC curve*, Receiver Operating Characteristic curve, is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: True Positive Rate (TPR) and False Positive Rate (FPR), where

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN}$$

AUC measures the entire two-dimensional area underneath the entire *ROC curve* from (0,0) to (1,1) and ranges in value from 0 to 1. A possible interpretation of *AUC* is as the probability that the model ranks a random positive sample more highly than a random negative sample.

The aforementioned metrics are effectively used in data science environments, however require a mathematical background which exceeds the one simply necessary to distinguish a TP from a TN (as well as FP and FN). Therefore, we decided to compute an other important and widely used quantity that has also an interesting visual impact, as it is not summarized by a scalar numeric figure, but instead amounts to a bi-dimensional matrix: the *Confusion Matrix*.

The *Confusion Matrix* (or error matrix) is a matrix where each row represents the actual values while each column represents the predicted ones. The element on row i and on column j returns the number of cases in which class i has been classified as a class j. In this work, in particular, after having computed the *Confusion Matrix*, the element (i, j) of the matrix has been divided by the total number of actual elements of the class related to the row i and multiplied by 100 in order to obtain percentage values and results that may be

simpler to interpret. Next to the rows and columns of each *Confusion Matrix* we then reported the number of respectively actual and predicted elements belonging to the different classes. Later we will refer to this matrix as *Confusion Matrix - proportion of actual*.

In the following Subsections we discuss the results obtained choosing a Random Forest, an other ensemble model and a neural network with Weka, Orange, KNIME and Python. With Ludwig, we employ its default Encoder-Decoder neural network. In addition, for the specific case of neural networks, we also provide a succinct analysis with respect to the parameter values adopted within the learning platforms and the Python libraries. All the models developed for the purpose of this paper are available at the following reference [11].

3.1 Weka

The Weka platform, "Waikato Environment for Knowledge Analysis", represents a graphical interface to open source machine learning software, as shown in Figure 2. In particular, Weka supports several standard data mining tasks such as data pre-processing, clustering, classification, regression, visualization, and feature selection. Containing a plethora of built-in tools for standard machine learning tasks, it is widely used for teaching, research, and industrial applications.

Within the scope of this contribution, Weka was employed to use models which fall under the neural network and ensemble model classes. In particular, the default Multi Layer Perceptron (MLP), Random Forest and Bagging models were built to analyze the aforementioned data set [34, 31]. Using this platform, only the configuration related to the size of the training set was changed with respect to the default values and, in particular, was set to the 66% of the entire data set (the default value was cross validation).

The training of the models in Weka was performed with a laptop with following characteristics:

- RAM: 12GB
- Processor: Intel(R) Core(TM) i7-7500U CPU 2.70GHz-2.90GHz
- System: Windows 10 Home (64-Bit)

When 52 input features were considered, 457 s were required to train the models, with 16 input features this step took 97 s. The *Confusion Matrices - proportion of actual* for MLP, Random Forest and Bagging in both settings are shown in Table 1.

With Random Forest and Bagging it is possible to observe excellent results both considering 52 and 16 input features. With MLP excellent results can be observed considering 52 input features, whereas a substantial increase in the error rate is observed if 16 input features are considered.

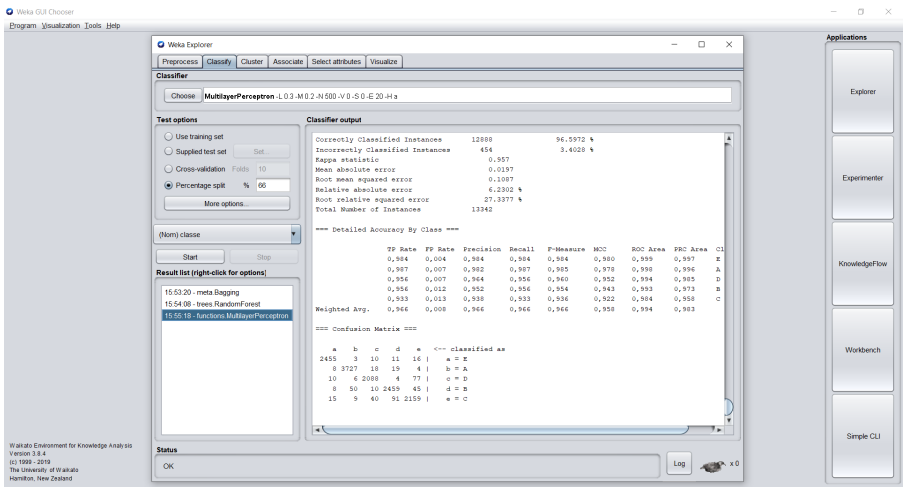

Fig. 2 Model in Weka.

3.2 Orange

Orange is an open source data visualization, machine learning and data mining toolkit. To analyze data, workflows are created by combining different components, called widgets. Using the Orange interactive graphical user interface, complex data analytics pipelines can be built focusing on the data analysis part rather than the coding part. These characteristics, together with the large library of available widgets, make Orange a useful tool for users with little or no coding experience and knowledge in data science in general.

The data analysis has been performed resorting to default Neural Network, Random Forest and AdaBoost models [24, 25]. The visual representation of the final data analysis pipeline constructed with Orange is shown in Figure 3.

The training of the models in Orange was performed with a laptop with following characteristics:

- RAM: 16GB
- Processor: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 1992 Mhz, 4 Core(s), 8 Logical Processor(s)
- System: Microsoft Windows 10 Home

The first run with 52 input features took 244 s to train the models using 66% of the data set and 10 repetitions of train/test. In the second run with 16 input features and the same settings 220 s were required. The *Confusion Matrices - proportion of actual* for Neural Network, Random Forest and AdaBoost in both settings are shown in Table 2.

Neural Network and Random Forest both showed an optimal reliability when the 52 input features were used. Also AdaBoost performed very well, thus showing a higher error rate. Interestingly, the *Confusion Matrix - proportion of actual* of the Random Forest remained almost unchanged as 16 input

Multi Layer Perceptron - 52 features

Actual \ Predicted	A	B	C	D	E	Σ
A	98.70%	0.50%	0.11%	0.48%	0.21%	**3776**
B	1.94%	95.61%	1.75%	0.39%	0.31%	**2572**
C	0.39%	3.93%	93.30%	1.73%	0.65%	**2314**
D	0.27%	0.18%	3.52%	95.56%	0.46%	**2185**
E	0.12%	0.44%	0.64%	0.40%	98.40%	**2495**
Σ	**3795**	**2584**	**2301**	**2166**	**2496**	**13342**

Multi Layer Perceptron - 16 features

Actual \ Predicted	A	B	C	D	E	Σ
A	89.96%	2.67%	2.99%	3.97%	0.40%	**3776**
B	12.33%	61.39%	13.41%	7.70%	5.17%	**2572**
C	2.77%	10.20%	75.50%	9.29%	2.25%	**2314**
D	1.42%	3.43%	9.61%	84.53%	1.01%	**2185**
E	2.40%	5.17%	7.78%	4.85%	79.80%	**2495**
Σ	**3869**	**2120**	**2609**	**2531**	**2213**	**13342**

Random Forest - 52 features

Actual \ Predicted	A	B	C	D	E	Σ
A	99.97%	0.03%	0.00%	0.00%	0.00%	**3776**
B	0.12%	99.88%	0.00%	0.00%	0.00%	**2572**
C	0.00%	0.00%	99.96%	0.04%	0.00%	**2314**
D	0.00%	0.00%	0.37%	99.59%	0.05%	**2185**
E	0.00%	0.00%	0.00%	0.04%	99.96%	**2495**
Σ	**3788**	**2570**	**2321**	**2178**	**2495**	**13342**

Random Forest - 16 features

Actual \ Predicted	A	B	C	D	E	Σ
A	99.95%	0.03%	0.00%	0.00%	0.03%	**3776**
B	0.08%	99.57%	0.31%	0.04%	0.00%	**2572**
C	0.00%	0.26%	99.35%	0.39%	0.00%	**2314**
D	0.05%	0.00%	0.09%	99.82%	0.05%	**2185**
E	0.00%	0.04%	0.08%	0.00%	99.88%	**2495**
Σ	**3777**	**2569**	**2311**	**2191**	**2494**	**13342**

Bagging - 52 features

Actual \ Predicted	A	B	C	D	E	Σ
A	99.71%	0.08%	0.16%	0.00%	0.05%	**3776**
B	0.70%	98.41%	0.51%	0.27%	0.12%	**2572**
C	0.13%	0.13%	99.22%	0.52%	0.00%	**2314**
D	0.09%	0.41%	1.37%	99.08%	0.05%	**2185**
E	0.00%	0.20%	0.24%	0.24%	99.32%	**2495**
Σ	**3788**	**2551**	**2351**	**2168**	**2484**	**13342**

Bagging - 16 features

Actual \ Predicted	A	B	C	D	E	Σ
A	99.07%	0.69%	0.00%	0.21%	0.03%	**3776**
B	1.09%	99.97%	1.40%	0.31%	0.23%	**2572**
C	0.17%	1.08%	98.23%	0.43%	0.09%	**2314**
D	0.14%	0.37%	0.96%	98.44%	0.09%	**2185**
E	0.16%	0.92%	0.48%	0.24%	98.20%	**2495**
Σ	**3780**	**2576**	**2342**	**2183**	**2461**	**13342**

Table 1 From top to bottom: *Confusion Matrices - proportion of actual* of Multi Layer Perceptron, Random Forest and Bagging with Weka using 66% of the data set as training data.

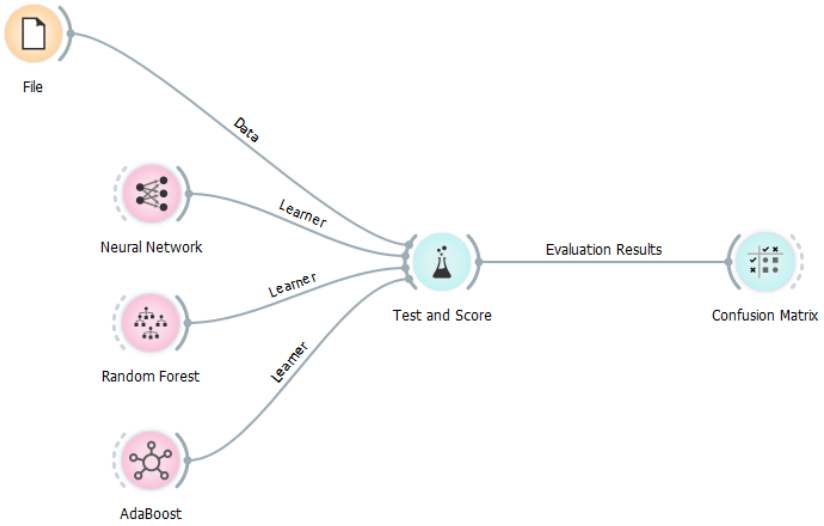

Fig. 3 Model in Orange.

features were used, while the Neural Network error rate increased of almost ten percentage points. This means that Random Forest proved to be the robust algorithm when handling the considered raw data. These results prove that, for this exemplar case, it is possible to construct a reliable activity detection model with minimal effort and without any specific knowledge.

3.3 Ludwig

Ludwig is an open source deep learning toolbox whose models are characterized by a versatile and flexible Encoder-Decoder architecture. Many standard models are provided. However, these models can be customized by changing the standard parameter values. To train the model only a tabular file is needed (a `CSV` formatted file) with the data and a configuration file (a `yaml` formatted file) which defines which columns of the tabular file are input features and which are target features. The configuration file also defines the type of features to be used in the model. The following type of features are currently supported by the Encoder-Decoder architecture implemented in Ludwig: text, numerical, binary, category, set, sequence, image, timeseries, bag. With the help of the described characteristics, Ludwig can help users who do not have specific knowledge in coding or in data sciences in general to apply deep learning algorithms. However, unlike Orange, there is no graphical user interface.

In Ludwig the standard Encoder-Decoder model was used without changing any of the default parameters values [51], Figure 4. For data analysis the

Neural Network - 52 features

Actual \ Predicted	A	B	C	D	E	Σ
A	99.83%	0.14%	0.01%	0.01%	0.01%	**37940**
B	0.18%	99.59%	0.16%	0.02%	0.06%	**25820**
C	0.01%	0.25%	99.27%	0.44%	0.02%	**23270**
D	0.03%	0.00%	0.51%	99.22%	0.23%	**21870**
E	0.00%	0.05%	0.05%	0.17%	99.72%	**24530**
Σ	**37931**	**25839**	**23269**	**21855**	**24536**	**133430**

Neural Network - 16 features

Actual \ Predicted	A	B	C	D	E	Σ
A	98.04%	0.96%	0.46%	0.39%	0.15%	**37940**
B	2.65%	91.20%	3.90%	1.03%	1.21%	**25820**
C	0.58%	3.87%	90.41%	4.20%	0.94%	**23270**
D	0.54%	0.81%	4.78%	92.63%	1.23%	**21870**
E	0.25%	1.03%	1.87%	2.52%	94.33%	**24530**
Σ	**38195**	**25243**	**23725**	**22271**	**23996**	**133430**

Random Forest - 52 features

Actual \ Predicted	A	B	C	D	E	Σ
A	99.94%	0.04%	0.01%	0.01%	0.01%	**37940**
B	0.41%	99.38%	0.17%	0.01%	0.02%	**25820**
C	0.01%	0.50%	99.30%	0.19%	0.00%	**23270**
D	0.01%	0.01%	0.81%	99.09%	0.08%	**21870**
E	0.00%	0.04%	0.04%	0.20%	99.73%	**24530**
Σ	**38028**	**25802**	**23343**	**21768**	**24489**	**133430**

Random Forest - 16 features

Actual \ Predicted	A	B	C	D	E	Σ
A	99.78%	0.12%	0.05%	0.02%	0.03%	**37940**
B	0.62%	98.30%	0.93%	0.09%	0.07%	**25820**
C	0.06%	0.57%	98.76%	0.59%	0.02%	**23270**
D	0.02%	0.06%	0.50%	99.32%	0.09%	**21870**
E	0.05%	0.13%	0.24%	0.19%	99.39%	**24530**
Σ	**38044**	**25607**	**23408**	**21938**	**24433**	**133430**

AdaBoost - 52 features

Actual \ Predicted	A	B	C	D	E	Σ
A	98.61%	0.81%	0.20%	0.24%	0.14%	**37940**
B	1.32%	96.17%	1.36%	0.57%	0.59%	**25820**
C	0.35%	1.26%	96.71%	1.24%	0.43%	**23270**
D	0.43%	0.54%	1.34%	97.17%	0.51%	**21870**
E	0.13%	0.74%	0.62%	0.60%	97.91%	**24530**
Σ	**37965**	**25729**	**23378**	**21923**	**24435**	**133430**

AdaBoost - 16 features

Actual \ Predicted	A	B	C	D	E	Σ
A	98.64%	0.81%	0.17%	0.18%	0.20%	**37940**
B	1.18%	95.91%	1.53%	0.72%	0.66%	**25820**
C	0.24%	1.54%	96.84%	0.98%	0.40%	**23270**
D	0.36%	0.77%	0.98%	97.45%	0.44%	**21870**
E	0.18%	0.92%	0.44%	0.61%	97.85%	**24530**
Σ	**37905**	**25823**	**23316**	**21947**	**24439**	**133430**

Table 2 From top to bottom: *Confusion Matrices - proportion of actual* of Neural Network, Random Forest and AdaBoost with Orange using 66% of the data set as training data and 10 repetitions of train/test.

original `CSV` formatted data file was used. According to this file the `yaml` formatted configuration file with the information related to the features was created. Considering the feature types supported by Ludwig, all input features are numerical features, while the target feature is categorical.

csv-format file
filedata.csv

```
roll_belt, pitch_belt, yaw_belt, total_accel_belt, ... , classe
3.70, 41.6, -82.8, 3, 132.0, ... , E
3.66, 42.8, -82.5, 2, 129.0, ... , E
3.58, 43.7, -82.3, 1, 125.0, ... , E
3.56, 44.4, -82.1, 1, 120.0, ... , E
3.57, 45.1, -81.9, 1, 115.0, ... , E
3.45, 45.6, -81.9, 1, 110.0, ... , E
3.31, 46.2, -81.9, 3, 104.0, ... , E
2.91, 46.9, -82.2, 4,  98.6, ... , E
2.31, 47.4, -82.6, 2,  93.2, ... , E
2.00, 47.7, -82.8, 3 ,  88.5, ... , E
...
```

yaml-format file
filedata.yaml

```
input_features:
 -
  name: roll_belt
  type: numerical
 -
  name: pitch_belt
  type: numerical
 -
  name: yaw_belt
  type: numerical
 -
  name: total_accel_belt
  type: numerical
 -
  ...
output_features:
 -
  name: classe
  type: category
```

console

```
> ludwig experiment  --data_csv filedata.csv  --model_definition_file filedata.yaml
```

Fig. 4 Model in Ludwig.

The training of the models in Ludwig was performed with a laptop with following characteristics:

- RAM: 12GB
- Processor: Intel(R) Core(TM) i7-7500U CPU 2.70GHz-2.90GHz
- System: Windows 10 Home (64-Bit)

The data set containing 39,242 rows of raw data was divided as follows in training set, validation set and test set: 27,588 rows for training set, 3,857 rows for validation set and the 7,797 remaining rows for the test set. Considering the aim of this work, it should be noted that the data split was carried out by default without actively needing any indication of how to divide the data set.

For the first run with 52 input features it took 172 s to train the model with 70% of the data set and with 55 epochs. The default values for the epochs number is 100 but after 5 epochs since last validation accuracy improvement an early stopping occurs, as happened in this case. The best value of accuracy on validation set is at epoch 50. For the second run with 16 input features it took 228 s to train the model with 70% of the data set and with 100 epochs. In Table 3 the *Confusion Matrices - proportion of actual* obtained with Ludwig are shown.

Encoder-Decoder - 52 features

Actual \ Predicted	A	B	C	D	E	Σ
A	86.11%	2.01%	7.68%	3.22%	0.98%	**2239**
B	12.83%	59.72%	13.56%	2.05%	11.84%	**1512**
C	6.54%	6.99%	78.66%	0.90%	6.91%	**1331**
D	7.54%	4.20%	23.31%	50.04%	14.92%	**1287**
E	5.39%	12.54%	8.82%	3.71%	69.54%	**1428**
Σ	**2383**	**1274**	**1478**	**1850**	**812**	**7797**

Encoder-Decoder - 16 features

Actual \ Predicted	A	B	C	D	E	Σ
A	72.58%	8.04%	6.07%	9.02%	4.29%	**2239**
B	22.35%	36.24%	15.34%	10.05%	16.01%	**1512**
C	27.72%	7.74%	50.04%	7.44%	7.06%	**1331**
D	14.30%	13.52%	18.73%	44.44%	9.01%	**1287**
E	13.17%	29.06%	19.26%	14.57%	23.95%	**1428**
Σ	**2704**	**1420**	**890**	**1550**	**1233**	**7797**

Table 3 *Confusion Matrices - proportion of actual* of Ludwig Encoder-Decoder model with default parameters using approximately 70% of the data set as training data, 10% as validation data and 20% as test data.

Interesting results have also been obtained with Ludwig. Although percentage values are lower than those obtained with Weka and Orange, the results reveal to be promising, when focusing the attention on the class of correct movements (i.e., A). In fact, the results obtained utilizing 52 input features are significant, considering that in many applications it is sufficient to correctly distinguish between "correct" from "incorrect" movements. The percentage of times a correct movement is detected (i.e., A) obviously decrease when using 16 instead of 52 input features, as shown in Figure 3.

3.4 KNIME

KNIME and in particular the KNIME Analytics Platform is an open source software for creating data science models. KNIME attempts to make understanding data and designing data science workflows and reusable components accessible to everyone users open, and continuously integrating new developments. In fact, with this platform it is possible to create visual data science workflows with an intuitive, drag and drop style graphical interface, without

the need for coding. To build a data science workflow over 2000 nodes are available and there is the possibility to model each step of the analysis in order to control the flow of data. Therefore, like Orange, KNIME is characterized by an interactive graphical user interface.

In KNIME we adopted the same models used in Orange or, when this was not possible, models that belonged to the same class (e.g., AdaBoost and Boosting both belong to the ensemble models class). Therefore, the default Multi Layer Perceptron (MLP), Random Forest and Boosting models have been used to analyze the data set [28]. The only configuration of the KNIME nodes that has been changed, with respect to the default values, is the size of the training set, which was set to the 66% of the entire data set, while the default value would have been the 10% of the entire data set. The visual representation of the final data science workflow constructed with KNIME is shown in Figure 5.

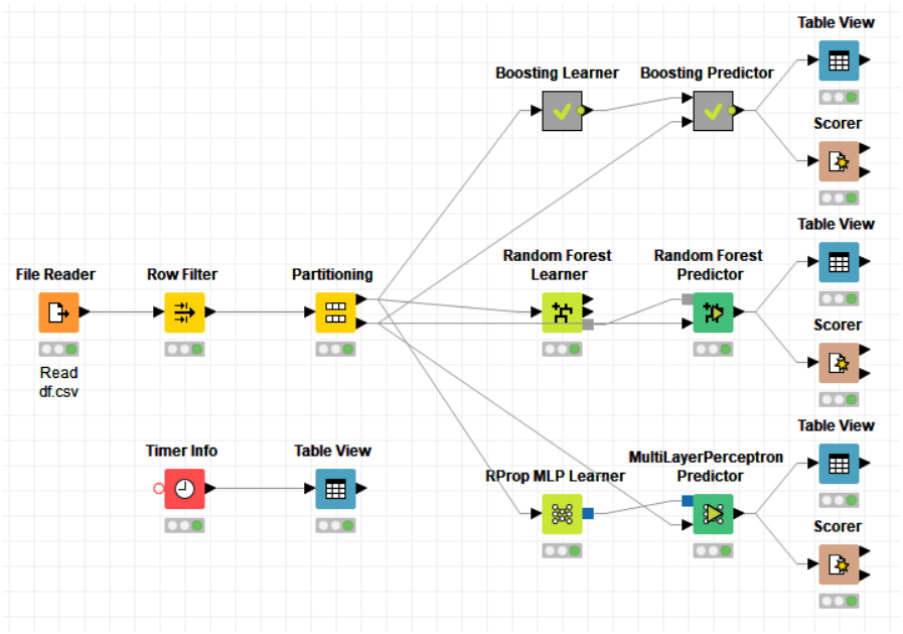

Fig. 5 Model in KNIME.

The training of the models in KNIME was performed with a laptop with following attributes:

- RAM: 12GB
- Processor: Intel(R) Core(TM) i7-7500U CPU 2.70GHz-2.90GHz
- System: Windows 10 Home (64-Bit)

When using 52 input features, the training of the models took 83 s, whereas with 16 input features the time required was 32 s. The *Confusion Matrices* -

proportion of actual for MLP, Random Forest and Boosting in both settings are shown in Table 4.

With the Random Forest algorithm, we observed results that are very similar to those obtained with the Random Forest built in Weka and in Orange, for both the 52 and the 16 input features settings. Boosting and MLP, were not as successful instead, in fact a closer look at the default parameter values reveals that these may not be suitable for the considered classification problem. In fact, as shown in Table 5, the main default parameters values for the MLP model are the following: 100 maximum number of iterations, 1 hidden layer and 10 neurons per layer. In essence, one hidden layer number and ten neurons may not be sufficient for this task.

3.5 Python

As a final experiment, we performed the analysis of the data employing Python, an interpreted, high-level, general-purpose programming language that is widely used by the machine and deep learning community [62]. Python, in fact, provides a plethora of libraries available for data science applications. Among these, the most widely used ones for modeling are scikit-learn and Keras [5, 6]. Scikit-learn represents a general purpose library, as it includes both machine and deep learning models. Keras, instead, is centered on the use of deep learning, hence, of neural network based algorithms. In the following we explain the differences of using these two libraries from a non-data scientist perspective.

We hence employed scikit-learn functions to build neural network, i.e., a Multi Layer Perceptron (MLP), AdaBoost and Random Forest models resorting as much as possible to the default parameter values. Regarding the default parameters, for all the considered models, only one value was changed: instead of the default value of the 75% of data, we used the 66% of the entire data set for training, to resemble as much as possible the operational settings before.

The training of the models in Python was performed with a laptop with following attributes:

- RAM: 12GB
- Processor: Intel(R) Core(TM) i7-7500U CPU 2.70GHz-2.90GHz
- System: Windows 10 Home (64-Bit)

Training required 24 s and 16 s when considering the 52 and 16 input features, respectively. The *Confusion Matrices - proportion of actual* for MLP, Random Forest and AdaBoost for both settings are shown in Table 6.

Random Forest and MLP exhibit very good performance (close to 100%), considering both the 52 and 16 input features, while we observe high error rates when employing AdaBoost.

As anticipated, resorting then to Keras, we implemented a layer by layer neural network and analyzed its behavior when using the default values for the model parameters. It is important now to note that it is necessary to set at least a few parameters in Keras (in fact, not all of the required parameters

Multi Layer Perceptron - 52 features

Predicted

Actual	A	B	C	D	E	Σ
A	68.28%	6.97%	9.24%	10.64%	4.88%	**3789**
B	11.73%	37.31%	13.68%	11.38%	25.89%	**2565**
C	18.78%	20.51%	44.05%	12.03%	4.63%	**2311**
D	13.41%	8.08%	27.86%	27.45%	23.21%	**2215**
E	8.08%	20.71%	21.77%	15.31%	34.12%	**2462**
Σ	**3818**	**2384**	**2872**	**1958**	**2310**	**13342**

Multi Layer Perceptron - 16 features

Predicted

Actual	A	B	C	D	E	Σ
A	77.82%	7.46%	8.44%	3.39%	2.89%	**3778**
B	18.71%	45.82%	15.61%	8.55%	11.31%	**2608**
C	13.22%	13.61%	56.35%	10.82%	6.01%	**2330**
D	6.38%	16.98%	22.95%	31.69%	22.00%	**2209**
E	4.63%	21.10%	19.32%	9.52%	45.43%	**2417**
Σ	**3989**	**2679**	**3013**	**1533**	**2128**	**13342**

Random Forest - 52 features

Predicted

Actual	A	B	C	D	E	Σ
A	99.97%	0.00%	0.00%	0.00%	0.03%	**3789**
B	0.16%	99.81%	0.04%	0.00%	0.00%	**2565**
C	0.00%	0.22%	99.61%	0.17%	0.00%	**2311**
D	0.00%	0.00%	0.32%	99.50%	0.18%	**2215**
E	0.00%	0.00%	0.00%	0.20%	99.80%	**2462**
Σ	**3792**	**2565**	**2310**	**2213**	**2462**	**13342**

Random Forest - 16 features

Predicted

Actual	A	B	C	D	E	Σ
A	99.87%	0.13%	0.00%	0.00%	0.00%	**3778**
B	0.08%	99.50%	0.42%	0.00%	0.00%	**2608**
C	0.04%	0.17%	99.44%	0.34%	0.00%	**2330**
D	0.00%	0.00%	0.32%	99.64%	0.05%	**2209**
E	0.00%	0.04%	0.08%	0.12%	99.75%	**2417**
Σ	**3776**	**2605**	**2337**	**2212**	**2412**	**13342**

Boosting - 52 features

Predicted

Actual	A	B	C	D	E	Σ
A	76.56%	2.96%	13.35%	5.15%	1.98%	**3789**
B	15.13%	59.61%	12.94%	3.94%	8.38%	**2565**
C	27.65%	10.13%	50.32%	8.65%	3.25%	**2311**
D	21.44%	7.04%	16.03%	46.00%	9.48%	**2215**
E	6.13%	13.12%	6.46%	21.28%	53.01%	**2462**
Σ	**4554**	**2354**	**2515**	**2039**	**1880**	**13342**

Boosting - 16 features

Predicted

Actual	A	B	C	D	E	Σ
A	66.97%	9.79%	9.00%	11.04%	3.20%	**3778**
B	16.10%	42.06%	16.53%	13.23%	12.08%	**2608**
C	12.83%	21.24%	46.22%	15.06%	4.64%	**2330**
D	7.51%	15.84%	8.87%	55.73%	12.04%	**2209**
E	6.91%	25.36%	19.61%	17.19%	30.33%	**2417**
Σ	**3582**	**2925**	**2518**	**2774**	**1543**	**13342**

Table 4 From top to bottom: *Confusion Matrices - proportion of actual* of Multi Layer Perceptron, Random Forest and Boosting with KNIME using 66% of the data set as training data.

Neural Network/MLP default values

	Epochs	Hidden Layers	Neurons per Layer	Activation Function	Loss Function	Optimizer
Weka	500	1	(attributes + classes) / 2	Sigmoid Function	/	/
Orange	200	1	100	ReLu Function	optimized Log Loss Function	Adam
KNIME	100	1	10	clipped Logistic Function	/	Rprop
Python with scikit-learn	200	1	100	ReLu Function	optimized Log Loss Function	Adam
Python with Keras	1	*no default value*	*no default value*	Linear Function	*no default value*	*no default value*

Table 5 Default values for principal Neural Network/MLP model parameters in the different considered learning platforms and programming language.

have a default value). In Table 5 we specify which parameters are necessary for the MLP model, while lacking pre-set default values (i.e., hidden layers number, neurons number in a layer, loss function, optimizer). In addition, we observe that the default settings utilized for the number of epochs and the activation function are ill suited to approach a real world problem, as they amount to a single epoch and to a linear activation function.

We hence proceeded adopting the default settings, wherever there were available, deciding to choose as values for the remaining parameters those suggested in [48], a well known beginners' data science blog. Hence, we finally used the following parameter values: 1 epoch (default value), 2 hidden layers with 32 neurons for each layer, "linear" activation function (default value), "categorical_crossentropy" loss function and "adam" optimizer. The *Confusion Matrices - proportion of actual* obtained in this case for the two feature sets (i.e., considering respectively 52 and 16 input features) are shown in Table 7.

The performance of the described MLP model appears unacceptable. This could be due to the default parameters values: these may not be suitable for the considered classification problem. In particular, we note that considering a linear activation function the resulting MLP behaves as a linear model. Therefore, when using Keras in Python, and in particular a model which requires the tuning of different parameters, some knowledge related to how the specific model works is necessary to obtain any significant result.

We finally decided to verify the complexity of building a MLP model in Keras which is capable of obtaining significant results. We hence used the same lines of code, simply changing the default values as follows: 40 epochs and 80 epochs respectively considering 52 and 16 input features, 2 hidden layers with 32 neurons for each layer, "relu" activation function, "softmax" activation function for the output layer, "categorical_crossentropy" loss function and "adam" optimizer. The *Confusion Matrices - proportion of actual* for the MLP models in both settings (i.e., considering respectively 52 and 16 input features) are shown in Table 7. It is possible to observe that with only a few modifications it was possible to obtain now interesting results (all diagonal values fall above the 90% threshold).

Multi Layer Perceptron - 52 features

Predicted

Actual	A	B	C	D	E	Σ
A	99.79%	0.21%	0.00%	0.00%	0.00%	**3829**
B	2.87%	94.10%	2.37%	0.16%	0.50%	**2578**
C	0.04%	1.84%	95.29%	2.74%	0.09%	**2336**
D	0.46%	0.28%	6.65%	90.51%	2.11%	**2181**
E	0.00%	0.66%	1.78%	0.70%	96.86%	**2418**
Σ	**3906**	**2499**	**2475**	**2059**	**2403**	**13342**

Multi Layer Perceptron - 16 features

Predicted

Actual	A	B	C	D	E	Σ
A	97.41%	1.66%	0.21%	0.37%	0.35%	**3738**
B	0.93%	92.05%	5.60%	0.27%	1.16%	**2590**
C	0.09%	2.52%	95.00%	1.37%	1.03%	**2341**
D	0.28%	0.41%	7.13%	91.07%	1.10%	**2173**
E	0.20%	0.96%	2.72%	1.92%	94.20%	**2500**
Σ	**3678**	**2538**	**2600**	**2080**	**2446**	**13342**

Random Forest - 52 features

Predicted

Actual	A	B	C	D	E	Σ
A	100.00%	0.00%	0.00%	0.00%	0.00%	**3829**
B	0.08%	99.88%	0.04%	0.00%	0.00%	**2578**
C	0.00%	0.00%	100.00%	0.00%	0.00%	**2336**
D	0.00%	0.00%	0.55%	99.40%	0.05%	**2181**
E	0.00%	0.00%	0.00%	0.04%	99.96%	**2418**
Σ	**3831**	**2575**	**2349**	**2169**	**2418**	**13342**

Random Forest - 16 features

Predicted

Actual	A	B	C	D	E	Σ
A	99.97%	0.03%	0.00%	0.00%	0.00%	**3738**
B	0.23%	98.80%	0.89%	0.04%	0.04%	**2590**
C	0.00%	0.04%	99.66%	0.30%	0.00%	**2341**
D	0.00%	0.00%	0.23%	99.72%	0.05%	**2173**
E	0.00%	0.00%	0.12%	0.16%	99.72%	**2500**
Σ	**3743**	**2561**	**2364**	**2179**	**2495**	**13342**

AdaBoost - 52 features

Predicted

Actual	A	B	C	D	E	Σ
A	68.06%	18.41%	8.64%	3.29%	1.59%	**3829**
B	13.15%	63.42%	9.27%	6.90%	7.25%	**2578**
C	2.05%	6.98%	80.99%	7.23%	2.74%	**2336**
D	3.44%	4.03%	10.13%	73.59%	8.80%	**2181**
E	3.18%	7.82%	4.92%	4.96%	79.11%	**2418**
Σ	**3145**	**2780**	**2802**	**2198**	**2417**	**13342**

AdaBoost - 16 features

Predicted

Actual	A	B	C	D	E	Σ
A	67.42%	17.26%	8.56%	4.47%	2.30%	**3738**
B	12.66%	62.01%	11.20%	5.52%	8.61%	**2590**
C	1.07%	13.46%	74.93%	8.80%	1.75%	**2341**
D	7.18%	3.68%	13.21%	58.77%	17.17%	**3173**
E	5.44%	10.68%	8.04%	6.40%	69.44%	**2500**
Σ	**3165**	**2913**	**2852**	**1953**	**2459**	**13342**

Table 6 Top to bottom: *Confusion Matrices - proportion of actual* of Multi Layer Perceptron, Random Forest and AdaBoost implemented with Python and scikit-learn functions using 66% of the data set as training data.

MLP Keras default - 52 features

Actual \ Predicted	A	B	C	D	E	Σ
A	37.85%	56.37%	0.74%	3.38%	1.66%	**3786**
B	22.66%	63.96%	2.63%	4.72%	6.03%	**2586**
C	32.30%	63.27%	2.30%	0.54%	1.59%	**2393**
D	21.45%	65.62%	0.19%	9.09%	3.65%	**2135**
E	29.85%	58.48%	1.60%	5.24%	4.83%	**2442**
Σ	**3979**	**8131**	**194**	**585**	**453**	**13342**

MLP Keras default - 16 features

Actual \ Predicted	A	B	C	D	E	Σ
A	52.72%	9.37%	18.93%	5.02%	13.96%	**3883**
B	50.98%	16.76%	16.72%	3.82%	11.70%	**2589**
C	65.00%	1.36%	16.97%	7.08%	9.59%	**2274**
D	41.04%	16.96%	22.15%	7.03%	12.82%	**2176**
E	54.63%	13.64%	22.23%	4.92%	4.59%	**2420**
Σ	**7060**	**1528**	**2574**	**727**	**1453**	**13342**

MLP Keras modified - 52 features

Actual \ Predicted	A	B	C	D	E	Σ
A	98.71%	0.92%	0.13%	0.21%	0.03%	**3786**
B	4.80%	92.81%	1.89%	0.00%	0.50%	**2586**
C	0.21%	1.46%	93.73%	4.22%	0.38%	**2393**
D	0.84%	0.28%	4.92%	93.68%	0.28%	**2135**
E	0.45%	0.94%	0.78%	1.80%	96.03%	**2442**
Σ	**3895**	**2499**	**2421**	**2153**	**2374**	**1334**

MLP Keras modified - 16 features

Actual \ Predicted	A	B	C	D	E	Σ
A	94.93%	2.45%	0.82%	1.44%	0.36%	**3883**
B	1.43%	92.08%	3.24%	1.12%	2.12%	**2589**
C	0.00%	4.88%	90.46%	3.91%	0.75%	**2274**
D	0.14%	0.46%	4.78%	93.75%	0.87%	**2176**
E	0.04%	1.03%	1.12%	2.44%	95.37%	**2420**
Σ	**3727**	**2625**	**2304**	**2273**	**2413**	**13342**

Table 7 Confusion Matrices - proportion of actual of Multi Layer Perceptron implemented layer by layer with Python and Keras using 66% of the data set as training data.

4 Discussion

In this work we considered the platforms a non-data scientist may invoke to make sense of a collection of raw data related to a sports activity (i.e., unilateral dumbbell biceps curl in our scenario). To this aim we adopted four different data-human interface learning platforms, which do not require any coding skills from their users (but, obviously, the raw data) and a programming language as Python, utilizing in Python the functions made available by two different libraries.

The main results of our experimental campaign, from the point of view of the choice of the employed default models, are:

1. The default Random Forest model has proven to be reliable throughout all the adopted approaches;

2. The other ensemble models, i.e., Bagging, AdaBoost and Boosting, and Neural Networks models led to both good and bad results, depending on the platform.

The main results, instead, from the point of view of the platforms, are:

1. Weka shows excellent results (above the 98% threshold) with the Random Forest and Bagging models and both possible features sets. The MLP model, however, obtained excellent results only with the 52 features set. This for all the considered classes of movements;
2. Orange obtained very interesting results (above the 90% threshold) with all models and feature sets, for all the considered classes of movements;
3. Ludwig obtained acceptable results (above the 85% threshold) when considering the class of correct movements and the 52 feature set;
4. KNIME obtained very interesting results (above the 90% threshold) with both feature sets, for all the considered classes of movements, with the Random Forest model. The other models employed in this work (MLP and Boosting) led to unacceptable results (below the 80% threshold);
5. The model offered by the scikit-learn library in Python led to controversial results. The MLP and Random Forest models exhibited diagonal values on the *Confusion Matrices - proportion of actual* which all exceeded the 90% threshold. The AdaBoost model, instead, fell as low as the 58.77% correctly classified for one of the considered cases;
6. The MLP model offered by the Keras library in Python led to very poor, unacceptable, results when utilizing the default values. Just changing the default parameter values, but not the code structure, we showed how, instead, the model could always exceed the 90% threshold, for both feature sets.

In essence, our work indicates that the safest platform for a non-expert user of data science techniques may be Orange, whereas the best algorithmic choice, among the tested ones, may be the Random Forest. Clearly, further investigations are required, considering additional and diverse data sets as well as further machine and deep learning algorithms.

Regarding the Neural Network/MLP models, implemented in the considered learning platforms and programming language, in relation to which an in-depth analysis with respect to the different default settings was carried out, an observation about the phenomenon of overfitting is necessary. In particular, if a model has a default setting characterized by parameters values that lead it to have a highly complexity, it is possible that the model is too faithful to the training set and therefore not able to generalize on the test set. Nevertheless, it is possible to observe that, in the cases that present the aforementioned characteristics (e.g., Neural Network/MLP models a high number of neurons per layer), excellent results are obtained on the test set and therefore we can conclude that no overfitting occurred.

Finally, for the sake of completeness, we also briefly report on the data pre-processing steps that were required with the different learning platforms. Therefore, we implemented the following procedures:

- Locate and delete rows containing null values in the data set. In particular, this operation was expressively needed with KNIME and Python. Such situation is instead handled automatically in Weka, Orange and Ludwig;
- Convert the values of the target features "classe" from categorical to numeric. This operation was only necessary when utilizing the Python programming language with the Keras libraries. The other adopted learning platforms handled this situation automatically.

Please note, the least possible steps were implemented to keep the pre-processing phase as simple as possible. In essence, we solely aimed at removing any possible error messages. Any other possible data pre-processing, e.g., data normalization, has not been applied to the initial raw data, unless already included in the default settings of the different learning platforms.

5 Conclusion

The aim of this work was to demonstrate that there are simple ways of applying machine and deep learning algorithms to easily create models that may still provide significant results related to a specific problem, without necessarily having a specialized knowledge in the field of data science. For this purpose, a classification problem from the field of qualitative activity recognition was considered and solved with the help of the machine and deep learning platforms Weka, Orange, Ludwig and KNIME. A further analysis related to the same classification problem was then carried out utilizing the Python programming language. To pursue the aim of this work, the models were put to good use without setting any unnecessary parameters. Our analysis exhibits how the employed learning algorithms, in relation to the considered raw data set, can be powerful, but parameter dependent. Therefore, data-human interfaces and the learning algorithms they implement may certainly represent a useful tool for non-expert data scientists, nevertheless, they should carefully be put to good use.

6 Acknowledgement

The authors gratefully thank the University of Bologna for the Alma Attrezzature 2017 grant and the Golinelli Foundation for the Data Science scholarship.

References

1. (1993) Weka. URL https://www.cs.waikato.ac.nz/ml/weka/
2. (1996) Orange. URL https://orange.biolab.si/

3. (2006) KNIME Open for Innovation. URL www.knime.com
4. (2006) Python programming language. URL www.python.org
5. (2007) Scikit-learn. URL www.scikit-learn.org
6. (2015) Keras. URL www.keras.io
7. (2019) Ludwig Deep Learning. URL https://uber.github.io/ludwig/
8. Van der Aalst WM (2014) Data scientist: The engineer of the future. In: Enterprise interoperability VI, Springer, pp 13–26
9. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, et al. (2016) Tensorflow: A system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), pp 265–283
10. Alsheikh MA, Selim A, Niyato D, Doyle L, Lin S, Tan HP (2016) Deep activity recognition models with triaxial accelerometers. In: Workshops at the Thirtieth AAAI Conference on Artificial Intelligence
11. Angeli A, Riedel N, Marfia G (2019) Data science models. URL http://shorturl.at/asxF0
12. Athey S (2018) The impact of machine learning on economics. In: The economics of artificial intelligence: An agenda, University of Chicago Press
13. Bao L, Intille SS (2004) Activity recognition from user-annotated acceleration data. In: International conference on pervasive computing, Springer, pp 1–17
14. Bayat A, Pomplun M, Tran DA (2014) A study on human activity recognition using accelerometer data from smartphones. Procedia Computer Science 34:450–457
15. Bohanec M, Borštnar MK, Robnik-Šikonja M (2017) Explaining machine learning models in sales predictions. Expert Systems with Applications 71:416–428
16. Bujari A, Licar B, Palazzi CE (2011) Road crossing recognition through smartphone's accelerometer. In: 2011 IFIP Wireless Days (WD), IEEE, pp 1–3
17. Bujari A, Licar B, Palazzi CE (2012) Movement pattern recognition through smartphone's accelerometer. In: 2012 IEEE Consumer Communications and Networking Conference (CCNC), IEEE, pp 502–506
18. Buscher G, Dumais ST, Cutrell E (2010) The good, the bad, and the random: an eye-tracking study of ad quality in web search. In: Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval, ACM, pp 42–49
19. Chen D, Bellamy RK, Malkin PK, Erickson T (2016) Diagnostic visualization for non-expert machine learning practitioners: A design study. In: 2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), IEEE, pp 87–95
20. Chen M, Hao Y, Hwang K, Wang L, Wang L (2017) Disease prediction by machine learning over big data from healthcare communities. Ieee Access 5:8869–8879
21. Chen Y, Xue Y (2015) A deep learning approach to human activity recognition based on single accelerometer. In: 2015 IEEE International Confer-

ence on Systems, Man, and Cybernetics, IEEE, pp 1488–1492
22. Cook D, Feuz KD, Krishnan NC (2013) Transfer learning for activity recognition: A survey. Knowledge and information systems 36(3):537–556
23. Crisci C, Ghattas B, Perera G (2012) A review of supervised machine learning algorithms and their applications to ecological data. Ecological Modelling 240:113–122
24. Demšar J, Zupan B, Leban G, Curk T (2004) Orange: From experimental machine learning to interactive data mining. In: European Conference on Principles of Data Mining and Knowledge Discovery, Springer, pp 537–539
25. Demšar J, Curk T, Erjavec A, Gorup Č, Hočevar T, Milutinovič M, Možina M, Polajnar M, Toplak M, Starič A, et al. (2013) Orange: data mining toolbox in python. The Journal of Machine Learning Research 14(1):2349–2353
26. Fatima M, Pasha M (2017) Survey of machine learning algorithms for disease diagnostic. Journal of Intelligent Learning Systems and Applications 9(01):1
27. Faust O, Hagiwara Y, Hong TJ, Lih OS, Acharya UR (2018) Deep learning for healthcare applications based on physiological signals: A review. Computer methods and programs in biomedicine 161:1–13
28. Fillbrunn A, Dietz C, Pfeuffer J, Rahn R, Landrum GA, Berthold MR (2017) Knime for reproducible cross-domain analysis of life science data. Journal of biotechnology 261:149–156
29. García M, Domínguez C, Heras J, Mata E, Pascual V (2018) An on-going framework for easily experimenting with deep learning models for bioimaging analysis. In: International Symposium on Distributed Computing and Artificial Intelligence, Springer, pp 330–333
30. Guyon I, Chaabane I, Escalante HJ, Escalera S, Jajetic D, Lloyd JR, Macià N, Ray B, Romaszko L, Sebag M, et al. (2016) A brief review of the chalearn automl challenge: any-time any-dataset learning without human intervention. In: Workshop on Automatic Machine Learning, pp 21–30
31. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. ACM SIGKDD explorations newsletter 11(1):10–18
32. Hammerla NY, Fisher J, Andras P, Rochester L, Walker R, Plötz T (2015) Pd disease state assessment in naturalistic environments using deep learning. In: Twenty-Ninth AAAI Conference on Artificial Intelligence
33. Heaton J, Polson N, Witte JH (2017) Deep learning for finance: deep portfolios. Applied Stochastic Models in Business and Industry 33(1):3–12
34. Holmes G, Donkin A, Witten IH (1994) Weka: A machine learning workbench. In: Proceedings of ANZIIS'94-Australian New Zealnd Intelligent Information Systems Conference, IEEE, pp 357–361
35. Jordan MI, Mitchell TM (2015) Machine learning: Trends, perspectives, and prospects. Science 349(6245):255–260
36. Ketkar N (2017) Introduction to keras. In: Deep Learning with Python, Springer, pp 97–111

37. Ketkar N (2017) Introduction to pytorch. In: Deep learning with python, Springer, pp 195–208
38. Khan A, Mellor S, Berlin E, Thompson R, McNaney R, Olivier P, Plötz T (2015) Beyond activity recognition: skill assessment from accelerometer data. In: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, ACM, pp 1155–1166
39. Kranz M, MöLler A, Hammerla N, Diewald S, PlöTz T, Olivier P, Roalter L (2013) The mobile fitness coach: Towards individualized skill assessment using personalized mobile devices. Pervasive and Mobile Computing 9(2):203–215
40. Kroes M, Kessels AG, Kalff AC, Feron FJ, Vissers YL, Jolles J, Vles JS (2002) Quality of movement as predictor of adhd: results from a prospective population study in 5-and 6-year-old children. Developmental Medicine and Child Neurology 44(11):753–760
41. Kwapisz JR, Weiss GM, Moore SA (2011) Activity recognition using cell phone accelerometers. ACM SigKDD Explorations Newsletter 12(2):74–82
42. Ladha C, Hammerla NY, Olivier P, Plötz T (2013) Climbax: skill assessment for climbing enthusiasts. In: Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing, ACM, pp 235–244
43. Lane ND, Georgiev P, Qendro L (2015) Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, ACM, pp 283–294
44. Lara OD, Labrador MA (2013) A survey on human activity recognition using wearable sensors. IEEE communications surveys & tutorials 15(3):1192–1209
45. Lau SL, König I, David K, Parandian B, Carius-Düssel C, Schultz M (2010) Supporting patient monitoring using activity recognition with a smartphone. In: 2010 7th International Symposium on Wireless Communication Systems, IEEE, pp 810–814
46. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. nature 521(7553):436
47. Logan B, Healey J, Philipose M, Tapia EM, Intille S (2007) A long-term evaluation of sensing modalities for activity recognition. In: International conference on Ubiquitous computing, Springer, pp 483–500
48. Malik F (2019) Neural networks: a solid practical guide. URL https://medium.com/fintechexplained/neural-networks-a-solid-practical-guide-9f343594b02a
49. Marfia G, Roccetti M (2017) A practical computer based vision system for posture and movement sensing in occupational medicine. Multimedia Tools and Applications 76(6):8109–8129
50. Maurtua I, Kirisci PT, Stiefmeier T, Sbodio ML, Witt H (2007) A wearable computing prototype for supporting training activities in automotive production. In: 4th International Forum on Applied Wearable Computing 2007, VDE, pp 1–12

51. Molino P, Dudin Y, Miryala SS (2019) Ludwig Deep Learning. URL https://eng.uber.com/introducing-ludwig/
52. Naik A, Samant L (2016) Correlation review of classification algorithm using data mining tool: Weka, rapidminer, tanagra, orange and knime. Procedia Computer Science 85:662–668
53. Ortiz Laguna J, Olaya AG, Borrajo D (2011) A dynamic sliding window approach for activity recognition. In: Konstan JA, Conejo R, Marzo JL, Oliver N (eds) User Modeling, Adaption and Personalization, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 219–230
54. Parkka J, Ermes M, Korpipaa P, Mantyjarvi J, Peltola J, Korhonen I (2006) Activity classification using realistic data from wearable sensors. IEEE Transactions on information technology in biomedicine 10(1):119–128
55. Patel K (2010) Lowering the barrier to applying machine learning. In: Adjunct proceedings of the 23nd annual ACM symposium on User interface software and technology, ACM, pp 355–358
56. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, et al. (2011) Scikit-learn: Machine learning in python. Journal of machine learning research 12(Oct):2825–2830
57. Plötz T, Hammerla NY, Olivier PL (2011) Feature learning for activity recognition in ubiquitous computing. In: Twenty-Second International Joint Conference on Artificial Intelligence
58. Pourbabaee B, Roshtkhari MJ, Khorasani K (2017) Deep convolutional neural networks and learning ecg features for screening paroxysmal atrial fibrillation patients. IEEE Transactions on Systems, Man, and Cybernetics: Systems 48(12):2095–2104
59. Ravi N, Dandekar N, Mysore P, Littman ML (2005) Activity recognition from accelerometer data. In: Aaai, vol 5, pp 1541–1546
60. Riedel N, Angeli A, Marfia G (2019) Qualitative activity recognition using machine and deep learning: Experimenting with data-human interfaces for non data-scientists. In: Proceedings of the 5th EAI International Conference on Smart Objects and Technologies for Social Good, ACM, pp 7–12
61. Ronao CA, Cho SB (2016) Human activity recognition with smartphone sensors using deep learning neural networks. Expert systems with applications 59:235–244
62. Rossum G (1995) Python reference manual
63. Stiefmeier T, Roggen D, Ogris G, Lukowicz P, Tröster G (2008) Wearable activity tracking in car manufacturing. IEEE Pervasive Computing (2):42–50
64. Sung M, Marci C, Pentland A (2005) Wearable feedback systems for rehabilitation. Journal of neuroengineering and rehabilitation 2(1):17
65. Tarca AL, Carey VJ, Chen Xw, Romero R, Drăghici S (2007) Machine learning and its applications to biology. PLoS computational biology 3(6):e116

66. Tessendorf B, Gravenhorst F, Arnrich B, Tröster G (2011) An imu-based sensor network to continuously monitor rowing technique on the water. In: 2011 Seventh International Conference on Intelligent Sensors, Sensor Networks and Information Processing, IEEE, pp 253–258
67. Ugulino W, Velloso E, Fuks H (2019) Human activity recognition. URL http://groupware.les.inf.puc-rio.br/harixzz34dpS6oks
68. Velloso E, Bulling A, Gellersen H, Ugulino W, Fuks H (2013) Qualitative activity recognition of weight lifting exercises. In: Proceedings of the 4th Augmented Human International Conference, pp 116–123
69. Waller MA, Fawcett SE (2013) Data science, predictive analytics, and big data: a revolution that will transform supply chain design and management. Journal of Business Logistics 34(2):77–84
70. Wang J, Chen Y, Hao S, Peng X, Hu L (2019) Deep learning for sensor-based activity recognition: A survey. Pattern Recognition Letters 119:3–11
71. Yang Q, Suh J, Chen NC, Ramos G (2018) Grounding interactive machine learning tool design in how non-experts actually build models. In: Proceedings of the 2018 on Designing Interactive Systems Conference 2018, ACM, pp 573–584
72. Zorrilla M, García-Saiz D (2013) A service oriented architecture to provide data mining services for non-expert data miners. Decision Support Systems 55(1):399–411