Tel Aviv University
School of Computer Science

Danielle Ben Bashat – 204212757, Yonit Zall –303031892, Or Fayneh – 308064088

Real-Time Prediction of Cognitive Load Level and Stress Based on GSR and ECG Signals

Project Booklet

Supervisor: Amir Globerson and
Talma Handler

Table of Content

1 Introduction

In our stressful world, the maintenance of an optimal cognitive performance is a constant challenge. Particularly true in complex and stressful working environments, where cognitive performance is crucial (e.g., pilots). A system capable of monitoring its user's mental workload can evaluate the suitability of its interface and interactions for user's current cognitive status and properly change them when necessary. Our project's data is based on Prof. Handler's research that aims to develop a multi-parametric index that captures in real-time cognitive-load induction under stress. Our project's objective was to develop a real-time predictor of cognitive load and stress based on only two physiological signals: Galvanic skin response (GSR) and Electrocardiography (ECG). The signals were measured during an experiment that included a flying simulator task while manipulating five levels of cognitive load and stress. We analyzed the data from 10 subjects that performed this experiment. Six different classifiers were investigated regarding their ability to predict five different cognitive load levels and their ability to predict a stress state. A maximum accuracy of 88% was achieved for predicting the stress state. This project goes one-step toward the goal of developing systems that could detect psychological/ mental states based on human physiological measures in real-time.

2 Background

In the first part of this section, we will give a general description of cognitive load, stress and the prediction importance. Next, physiological signals: ECG and GSR that indicate cognitive load and stress state changes will be presented.

2.1 Defining Cognitive Load and Stress

Cognitive load

The term cognitive load (CL) refers to the amount of cognitive resources required for a person to complete a certain task. CL has been shown to have important implications for learning [1], safety in driving [2] and aviation [3].

Stress

Although definitions of stress vary, there is good consensus in the literature regarding conditions where it is likely to arise [4] [5]. Failure at a task, together with feelings of lack of control, in situations where others evaluate participants is a widely used paradigm for stress induction.

As cognitive workload increases, maintaining task performance within an acceptable range becomes more difficult. Increased cognitive workload may demand more cognitive resources than the human brain has available and may also result in physiological stress. Both cognitive overload and stress may affect the performance. An objective measure of workload could be used to evaluate alternative system designs, allocate workload appropriately to minimize

overload and stress, and intervene in real time before operators become overloaded while performing safety-critical tasks.

2.2 Physiological Signals: ECG and GSR Associated with Cognitive load and stress

Physiological signals have previously been proposed as a method of quantifying CL. The advantage of physiological methods is that they allow a direct and continuous measurement of the current workload level. Some notable successes in CL evaluation have been achieved via signals such as electrocardiogram (ECG) and galvanic skin response (GSR), which are the focus of this research effort and shown to be sensitive to both CL and stress [6] [7] [8] .

2.2.1 Electrocardiogram

Electrocardiography (ECG) is the approach of recording the electrical activity of the heart using electrodes attached to the skin. These electrodes detect the minimal electrical changes on the skin surface that appear from the heart muscle's contraction and relaxation during each heartbeat [9].

2.2.2 Galvanic Skin Response

Galvanic Skin Response (GSR) is a feature of human body continuously change its electrical characteristics. Theoretically, GSR is based on hypothesis, that skin resistance depends on the state of sweat glands (tubular structures of the skin that produce sweat. Since sweating is out of control of the human, it is regulated by the autonomous nervous system (ANS).
High arousal of the sympathetic branch of ANS increases the sweat gland activity, which accordingly increases skin conductance and another way around. This is the reason why GSR can be considered as an index of emotional and sympathetic responses.

3 Methods

In this chapter, we describe the data and the methods used for classification of the data. First, a detailed specification of the recorded data will be provided. Then, we introduce approaches for physiological signal preprocessing which involve filtration, feature extraction, and normalization. In the last part, we will introduce several machine-learning algorithms for data classification that we used for classification.

3.1 Data

The data in this study was collected from an experiment conducted in Functional Brain Center, Wohl Institute for Advanced Imaging, Tel-Aviv Sourasky Medical Center. The experiment aims to develop a multi-parametric index that captures in real-time cognitive-load induction

under stress. As of the date of writing the work this on-going study has about 45 subjects and is still conducting the experiment on a few more subjects.

The task's duration was 20 minutes and included navigation in a 3D flight simulator while engaging in executive function tasks. In the task, there were foursquare rings and the subjects were asked to navigate each time into a specific ring according to two arrows that were shown on the screen. For example, if there are two arrows that one direct up and the second direct right, the subject needed to navigate into the up-right square ring. The arrows were placed on the screen in a confusing order for implementing stroop-like task. In addition to navigate in the flight simulator, there was an auditory N-back task with different difficulty levels. For example, in 1 level of N-back, when the subject heard a letter that was heard right before, he needed to press a button.

There were 26 blocks per a flight task with different difficulty levels in a scale of 1-5. There were two blocks per each level (2x5 = 10 blocks) and the rest of the blocks were consider as baseline (16 blocks). The order of the blocks random for each subject.

The difficulty level of cognitive load was calculated by the difficulty of the auditory N-back (three different levels: 0-2) and the size of the square rings (three different sizes: big, medium and small) as described in Table 1.

In addition, each subject performed the flight task twice: under stress/ no stress. By total, each subject performed two flight tasks: 26X2 =52. In the stress condition, the subjects were asked to perform as best as they can, and there were additional stressors:

- Distressing sound: loud alarm sound, appears unexpectedly during the task (not required for performance).
- Social ranking: negative feedback on performance and a score relative to others.

Table 1: The cognitive task's difficulty level calculation

Cognitive load level	N-back level	Rings' size
1	1	big
2	1	medium
3	2	medium
4	2	small
5	3	small
Baseline	No-N-back	big
	No-N-back	medium
	No-N-back	small
	0	No rings

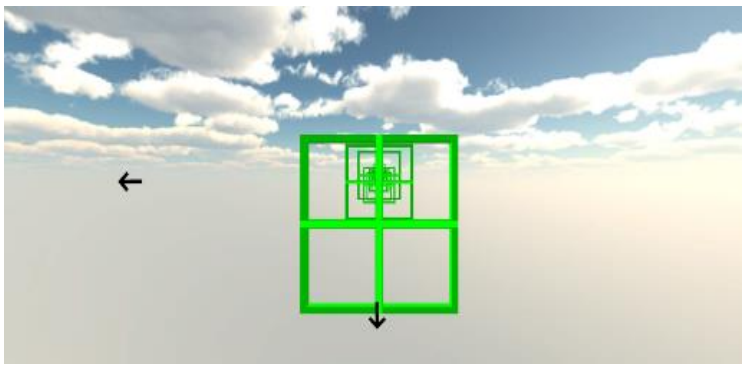


Figure 1: Example of the Flight simulator.

We built a data set of the raw signals based on ECG and GSR signals recordings from the task. Some recording sessions were corrupted, so 20 relevant records were obtained (two per subjects with stress and no stress conditions).

ECG signals were recorded with sampling rate 250 Hz and GSR signals were recorded with sampling rate 1000 Hz. Afterwards, during post-processing, GSR signals were down sampled to 250 Hz for synchronization with the ECG signals. From the experiment's flight task recordings, we extracted the ECG and GSR signals into blocks' segments according to labeled epochs of events including starting and ending the blocks' times and other parameters.

There were 200 blocks segments with different cognitive load levels by total, 20 blocks segments per level (not including the baseline). For increasing the data set and while maintain sufficient ranging time, we divided each block's signals time series into four sliding windows of equal length (2200 samples, or 8.8 seconds) with a window slide unit of length 1100 samples (4.4 seconds). This results in two neighboring windows sharing an overlap of 50% window length. Hence, each input sample for the learning was of size (2X2200) and there were 1400 input samples by total.

Studies, as mentioned earlier, are showing, that cognitive load level are distinguishable via machine learning algorithms trained on physiological data, but unfortunately, accuracies are still not satisfying for industrial use.

3.2 Preprocessing and Feature Extraction

Measured signals may have different types of artifacts caused by sudden muscle movements, equipment, external inference and some other factors. These conditions impede further data processing. Fortunately, there are lots of techniques to eliminate most of the mentioned issues. The methods we used will be described further in this section. Feature extraction is also essential step before data classification. Correctly calculated and selected features will significantly

affect the final score. In this section, we will describe feature extraction methods we used for ECG and GSR signals.

3.2.1 ECG Signal

In this section, the procedure for electroencephalographic data processing will be described. In short, we can divide it into three stages:

1. Remove noise.
2. Extraction of ECG signal from the experiment's flight task recording into the blocks' segments and divided into sliding windows segments (as described in section 3.1).
3. Calculate features.

3.2.1.1 ECG Noise Removal

The ECG data we used was initially post-processed by the experiment's lab. The data was acquired during simultaneous fMRI, thus, was affected by several artifacts, including the gradient artefact (due to the changing magnetic field gradients required for fMRI), and movement artifacts (resulting from movements in the strong magnetic field of the scanner, and muscle activity). The post-processing stage in the lab's experiment included correction of the gradient artifacts and high-pass filtering using Brainvision analyzer software.

In addition, measured signals can show overall patterns that are not intrinsic to the data. These trends can sometimes hinder the data analysis and must be removed. According to that issue, we eliminated linear trends and nonlinear trends using the functions: detrend , polyfit, polyval.

3.2.1.2 ECG Feature Extraction

After the data was divided into time windows as described in section 3.1, next step was to extract the features from the ECG signal. According to the literature an important ECG signal characteristics of stress and cognitive load is Heart rate variability (HRV) [10]. HRV is a phenomenon of volatility in the time interval between the heartbeats.

This physiological mark is measured by the length of consecutive interbeat intervals (RR-intervals) [11].

We used ECG Feature Extractor Toolbox in matlab to extract the following features: average Heart Rate, mean R-R interval distance, Root Mean Square Distance of Successive R-R interval, Number of R peaks in ECG that differ more than 50 millisecond and percentage of them, Standard Deviation of R-R series, Standard Deviation of Heart Rate, Sample Entropy, Power Spectral Entropy, Average Heart Rate Variability, Heart Rate Variability.

3.2.2 GSR Signal

Galvanic skin response carry valuable information for cognitive load and stress recognition. Its processing is simpler comparing to ECG but still requires several steps that will be described further in this section.

3.2.2.1 GSR Noise Removal

We received from the lab raw GSR signals, the signals were smoothed using a moving average of the elements of the signal, with a 20-element sliding window. This method is useful for reducing periodic trends in data .The window slides down the length of the signal vector, computing an average over the elements within each window.

3.2.2.2 GSR Normalization

In order to remove individual difference, the data collected under baseline (as mentioned in section 3.1, each record contains a block of base line condition that imply neutral state) was used to normalize the GSR signal under other conditions [12].
The normalized formula is defined as:

$$X_i' = \frac{X_i - X_{i\,min}}{X_{i\,max} - X_{imin}}$$

Where X_i is the i-th subject GSR , $X_{i\,min}$ is the minimum value of this subjects baseline data, $X_{i\,max}$ is the max value of this subjects baseline, Xi' is the subject value after normalized. For each subject we normalized the data twice – in stress and no stress condition.

3.2.2.3 GSR Feature Extraction

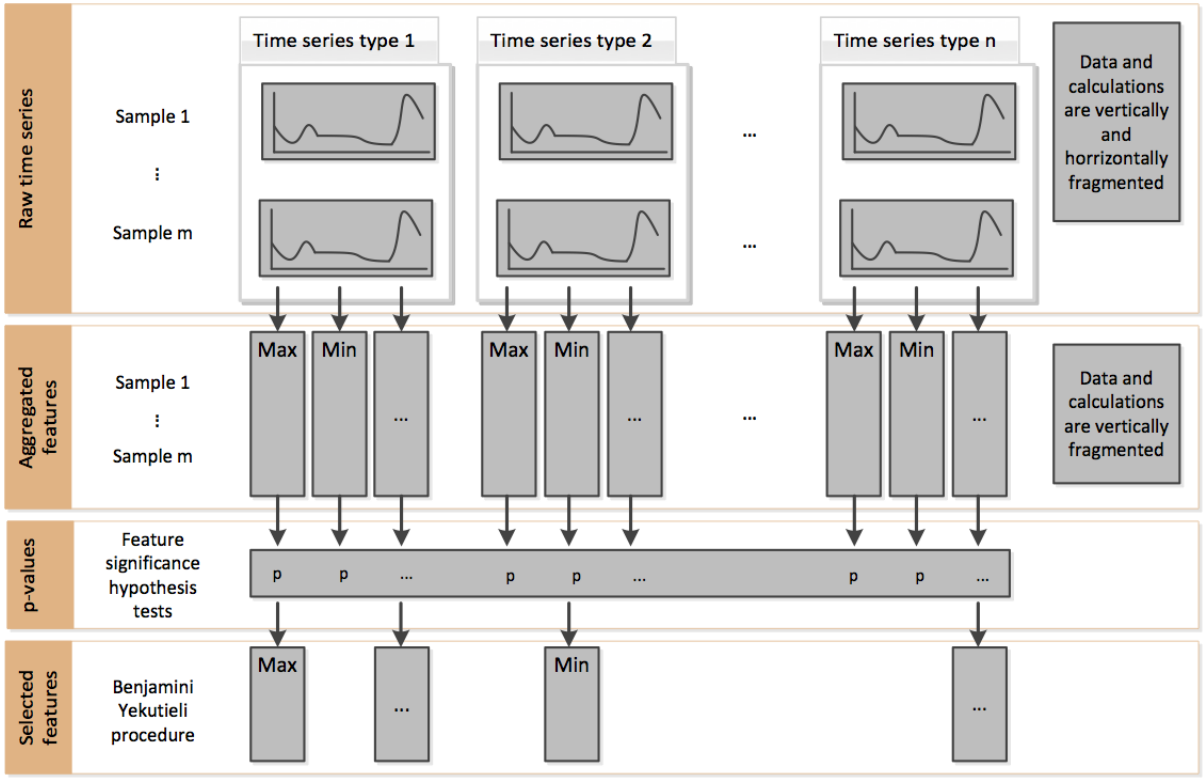
After the data was divided into time windows as described in section 3.1, next step was to extract the features from each time window.
We extracted statistical features which can reflect the change of GSR: mean, standard deviation, median, minimum, maximum of the signal.
Additional 11 time domain features were extracted according to [13]: Numberofstartles, MeanAmplitude, Maxamplitude, MaxTime, Medianamplitude, MedianTime, MeanTime, StandardDeviationAmplitude, StandardDeviationTime, Minamplitude, MinTime.

3.2.3 ECG and GSR Combined Feature Extraction and Filtering

We also used a different approach of feature extraction: instead of calculating a small amount of features one at a time, we used a python package called tsfresh to automatically calculate a large number of time series characteristics, the so-called features.

Firstly, we extracted 1600 features for all records. In a second step, each feature vector was individually and independently evaluated with respect to its significance for predicting the target under investigation. The result of these tests is a vector of p-values, quantifying the significance of each feature for predicting the label/target. The vector of p-values is evaluated on basis of the Benjamini-Yekutieli [14] procedure in order to decide which features to keep. Finally, a 1X110 vector represents each segment record of ECG and GSR.

Figure 2. The three phases of the features filtering process:



3.3 Data Classification

This section describes machine-learning algorithms for solving classification task that were employed in this work. Each method performs in a particular way, having various complexity, learning and prediction procedures, etc. Because of the mentioned reasons, their performance may vary.

3.3.1 Support Vector Machine

Support vector machine is a supervised algorithm that fits for solving both classification and regression tasks. The goal of SVM is to set up a hyperplane which will separate all training samples in two classes. This hyperplane if described by the equation 2.

$$f(x) = w^t x + b \quad (2)$$

Where w^ta two-dimensional vector according to the number of classes and b is bias. For multidimensional case, special functions called kernels (3) applied to input x as shown in equation 4.

$$k(x_i, x_i) = \varphi(x_i) \cdot \varphi(j) \quad (3)$$

$$f(x) = w^t \varphi(x) + b \quad (4)$$

They project the data to higher-dimensional space, making it linearly separable, this technique also called "kernel trick". The most used kernels are shown in Table XX

Table 5. Common SVM kernels

Name	Formula
Polynomial	$k(x_i, x_j) = (x_i \cdot x_j + 1)^d$
Gaussian	$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$
RBF	$k(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$
Hyperbolic tangent	$k(x_i, x_j) = \tanh\left(x_i \cdot x_j - \delta\right)$

The most fitting hyperplane will be the one that makes maximum margin between classes. The decision rule is based on the equation of the obtained hyperplane and forms in the next way: sign function is applied to the mentioned equation with substituted values of the point to be classified. An obtained value will be -1 or 1 which would say about belonging to one or another class. For multiclass classification, an approach called one-vs-all is used, it means, that multiple hyperplanes are built to separate each class from all others, joint together.

3.3.2 Random Forest

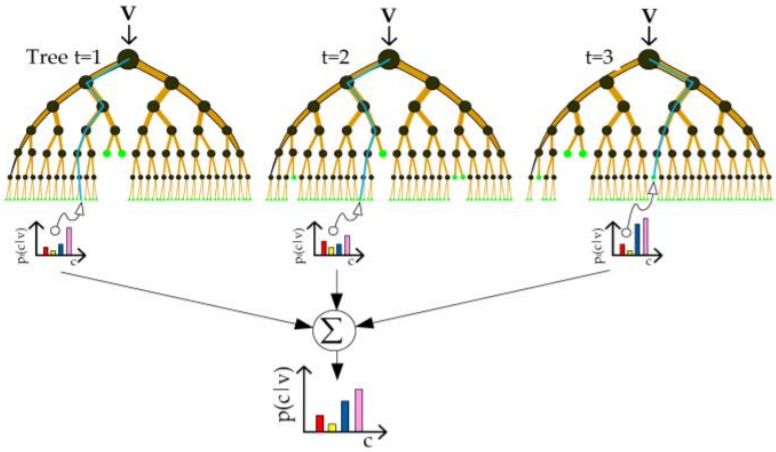
Random Forest are machine learning algorithm that belongs to the class of ensemble methods, to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator. Random Forest also is a method that is based on decision trees and widely used for classification and regression. To describe the mechanics of Random Forest, firstly, it should be clarified how a decision tree works. A decision tree is a binary tree, the primary purpose of this algorithm is to find such variable-value pair from the training data, which will provide optimal (by some criteria) split. Recursively repeating described procedure for newly appeared subsets, the algorithm proceeds until next maximum depth was reached or subset can be no longer divided (such subset is called leaf), so optimal tree is found. To estimate a new split, the algorithm uses one of following metrics:

- information gain
- Gini impurity
- mean square

Selection of the metrics depends on the task. Single decision tree tends to overfit training data, to overcome this issue, a technique called ensemble is used. Ensemble method makes

predictions by combining results of the set of classifiers, in case of Random Forest, it unites decision trees, which are “weak” learners into one “strong” learner, as Figure 3 illustrates.

Figure 3. A Random forest algorithm classification



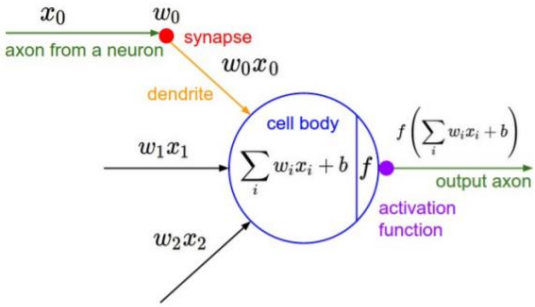
3.3.3 Gradient Boosting

Gradient boosted classifiers (GBMs) are machine-learning algorithm that also belongs to the class of ensemble methods. Whereas random forests build an ensemble of deep independent trees, GBMs build an ensemble of shallow and weak successive trees with each tree learning and improving on the previous. When combined, these many weak successive trees produce a powerful single estimator. At each particular iteration, a new weak, base-learner model is trained with respect to the error of the whole ensemble learnt so far.

3.3.4 Multilayer Perceptron

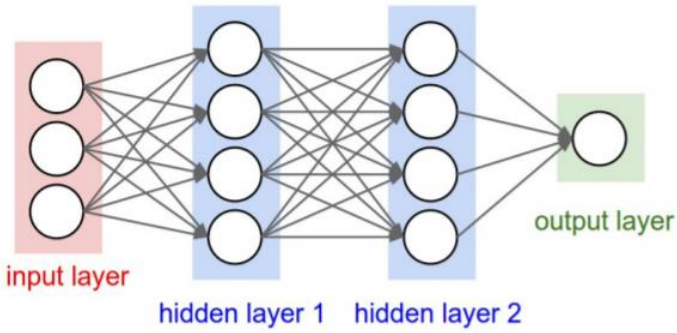
A multilayer perceptron (MLP) is a supervised learning algorithm that belongs to the class of feed-forward neural networks. MLP comprises simple hierarchically connected elements - artificial neurons. These elements are roughly simplified models of biological neurons. A function of a single artificial neuron – to produce a single value based on an input vector, as depicted in the Figure 4.

Figure 4. An artificial neuron model



To obtain this value, a special function called activation function applies to the dot product of the input signal with the neuron's weight vector plus bias term. Weights and bias are parameters of a neuron that adjust their values during the training phase to provide best possible classification results. There are several activation functions that used in practice. MLP can be presented as an acyclic graph with neurons as vertices and connections between the neurons as edges, so outputs of a neuron on the given layer will be inputs of neurons on the next layer. Important to note, that neurons on the same level have no connections within the layer. Graphical representation of three-layer MLP is shown in Figure 5

Figure 5. A three-layer neural network

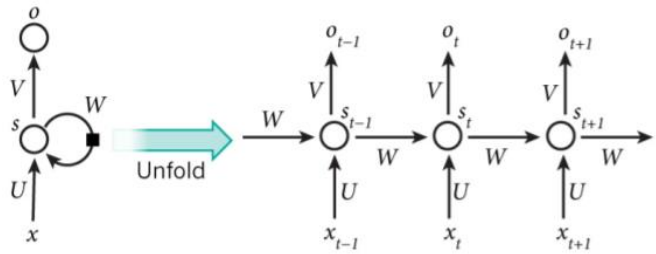


Mostly, the last layer of MLPs do not have activation, instead, depending on the task, a particular function for classification or regression is used. For mentioned learning procedure, MLP uses learning technique called backpropagation, it applies iterative adjustment of weights until achievement of global minima approximation of task-specific function that is also called loss function.

3.3.5 Recurrent Neural Networks

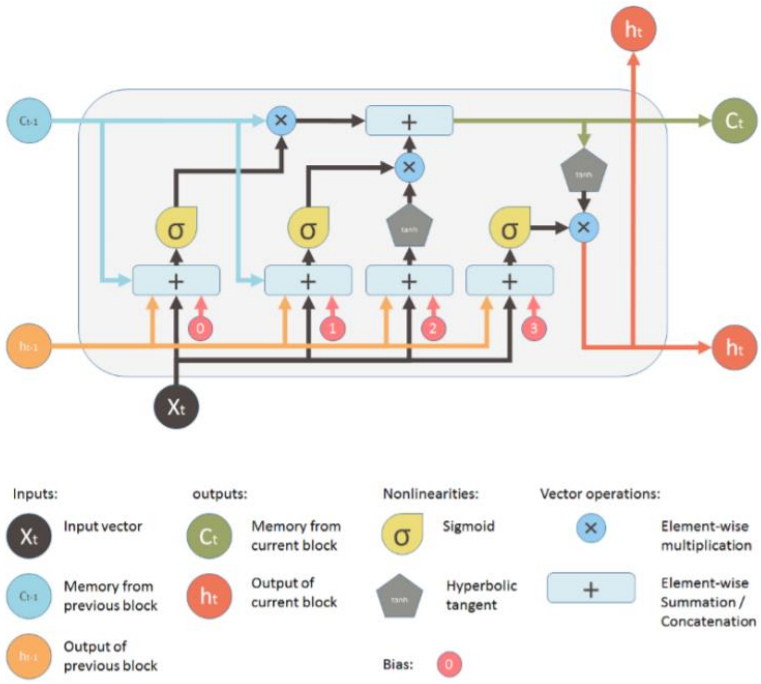
Recurrent neural networks (RNNs) are a specific type of neural networks intended to process sequential data. They are capable of both classification and regression tasks. A simplified explanation of RNNs can be formulated in the following way. Such network has some "memory" about computations on the previous steps and uses it to process subsequent elements. Schematic representation of RNN is shown in Figure 6.

Figure 6. Folded and unfolded graphical representation of RNN



Unfolding the RNN means depicting recurrent neurons for each element of the sequence. According to the image above, x_t is the input for time point t (x_t can be a single number as well as vector); s_t is the hidden state for time point t. In general, it is the mentioned "memory" of the network and calculates based on the current input x_t and previous hidden state s_{t-1} by formula s_t = f(Ux_t + W s_{t+1}). f is an activation function. Mostly, it's tanh or ReLu . o_t is output at the time t; it can be forwarded to another recurrent layer or function selected for the exact task. In contradiction to other types of neural networks, RNN uses the same parameters (U, W, V) for all steps, which means that the same procedure repeats with different inputs. This significantly reduces the number of parameters to learn but brings some disadvantages. s_t can not "remember" information captured from earlier steps. LSTMs that were mentioned above can mitigate this problem. LSTMs have no difference in architecture comparing to RNNs. However, they have a different structure of the hidden state which is shown in Figure 7.

Figure 7. A hidden state structure of the LSTM block



In contrast to RNN, LSTM block outputs not only h_t values, but $C_t$values that provide long-term memory. One more difference, which also shown on Figure 7, is that LSTM output h_t goes to entire model output like in RNNs and to the next block simultaneously, this mechanism provides short memory to considered model. This type or recurrent units are very effective in the processing of long-term sequences.

4 Results

In this chapter, we will present comparison of results of physiological signal classification using the data sets that were described in Chapter 3.1. Several machine learning algorithms were tested. We used the following classifiers: gradient boosting, random forest, SVM, multilayer perceptron and several RNN models (See Chapter 3.3). To evaluate classification performance we used two techniques. The first technique is called repeated random sub sampling validation. This technique implies testing the model on the records of 30% of all subjects, whereas training was performed on all 70% records from all subjects. We performed this split 10 times and the results are then averaged over the 10 splits. The second technique - Leave-one-out cross-validation implies testing the model on the records of one single subject, whereas training was performed on all records from remaining subjects. Thus, having 10 subjects, each training was held on nine subjects and tested on the one subject. The process is repeated 10 times and the average classification score is the estimate of the performance of a method.

4.1 Classification Using RNN

RNN's imply sequence. Thus, vectors of 8.8 second were extracted from the ECG and GSR records, inputs were represented by 2X 2200 matrix. Afterwards, we trained RNN and LSTM models on classifying 5 levels of cognitive load and on classifying the condition of stress no stress. The evaluation was held by repeated random sub sampling validation. WE trained the following models:

1. LSTM
    - 1 hidden layer unidirectional
    - 32 hidden layers unidirectional
2. Basic RNN
    - 32 hidden layers

All networks were trained using a learning rate and a lambda loss of 0.001. We tried several minibatches sizes and different number of epochs. We minimize categorical cross-entropy loss in all cases, and we use the Adam optimizer, an extension of the classic stochastic gradient descent optimizer.

Labels were "one hot vector" of size 1X5 when classifying cognitive load and of size 1X2 when classifying stress/ no stress condition. A summary of the accuracy is described in table 6.1.

Table 6.1 Classification accuracy results using RNN and LSTM models

		LSTM		Simple RNN
		1 layer	multi layers	multi layers
5 CL levels	1 subject	test: 0.25 train: 0.71	test: 0.3 train: 1	test: 0.325 train: 0.78
	all subjects	test: 0.2 train: 0.29	test: 0.23 train: 0.31	test: 0.18 train: 0.27
Stress/ no Stress	1 subject	test: 0.98 train: 0.99	test: 1 train: 1	test: 0.97 train: 0.98
	all subjects	test: 0.52 train: 0.61	test: 0.52 train: 0.65	test: 0.51 train: 0.6

4.1.1 Conclusions

In table 6.1, it can be seen that the RNN and LSTM models were not able to learn the data of 10 different subjects and thus train and test accuracy were low. When training the models on one subject all models got a very high accuracy in classifying the stress state, but only a high train accuracy when classifying cognitive load. As the ratio between the amount of available training data and the number of extracted features is too small, we assumed this was as a result of overfitting. We decided that the number of features must therefore be reduced for good generalization and performance. We tried two methods of feature extraction to reduce dimensions. In addition we taught that the amount of data might not be enough for training RNN and LSTM models, instead we decided to use simpler – less complex models.

4.2 Classification with Other Models

In the next part, we will describe two additional approaches for classifying the data. In each approach, we used a different technique for feature extraction. We than trained the data on 10 subjects with four different classifiers, for each classifier we optimized its hyper-parameters.

4.2.1 Manually Feature Extraction Approach

In this approach we used preprocessing technique and features extraction described in Chapters: 3.2.1.2, 3.2.2.3 for ECG and GSR respectively, we obtained 1X24 features for each segment out of 1400 segments collected from for all subjects (140 records from each subject). Results are shown in tables 6.2 -6.3.

Table 6.2. Classification accuracy results using repeated ransom sub sampling validation

Classifier/Label	5 CL levels	2 CL levels	Stress/no Stress	2 CL level X Stress/no Stress
Basic Neural Network (MLP)	Test: 0.20 Train: 0.22	Test: 0.56 Train: 0.58	Test: 0.54 Train: 0.56	Test: 0.22 Train: 0.21
Gradient Boosting	Test: 0.34 Train: 1	Test: 0.34 Train: 1	Test: 0.34 Train: 1	Test: 0.33 Train: 1
SVM	Test: 0.17 Train: 0.95	Test : 0.61 Train: 0.85	Test: 0.48 Train: 0.97	Test: 0.18 Train: 0.98
Random Forest	Test: 0.31 Train: 1	Test: 0.65 Train: 0.99	Test: 0.89 Train: 1	Test: 0.32 Train: 1

Table 6.3. Classification accuracy results using Leave-one-out cross-validation

Classifier/Label	5 CL levels	2 CL levels	Stress/no Stress	2 CL level X Stress/no Stress
Basic Neural Network (MLP)	Test: 0.20 Train: 0.21	Test: 0.6 Train: 0.59	Test: 0.53 Train: 0.59	Test: 0.21 Train: 0.21
Gradient Boosting	Test: 0.34 Train: 1	Test: 0.34 Train: 1	Test: 0.34 Train: 1	Test: 0.33 Train: 1
SVM	Test: 0.19 Train: 0.98	Test : 0.61 Train: 0.99	Test: 0.5 Train: 0.98	Test: 0.19 Train: 0.97
Random Forest	Test: 0. 23 Train: 0. 98	Test: 0.54 Train: 0.99	Test: 0.50 Train: 0.99	Test: 0.202 Train: 0.98

4.2.2 Automatic Feature Extraction Approach

We used dimensionality reduction algorithm (PCA) and feature selection as described in section 3.2.3. After feature selection, we obtained 1X110 features for each segment out of 1400 segments collected from for all subjects (140 records from each subject). Next, for each classifier we applied PCA and tested several dimensions reductions in order to find the best dimension for each classifier. Results are shown in tables 7 -8.

Table 7. Classification accuracy results using repeated random sub sampling validation

Classifier/Labels	**5 CL levels**	**2 CL levels**	**Stress/no Stress**	**2 CL level X Stress/no Stress**
Basic Neural Network (MLP)	Test: 0.36 Train: 0.1 PCA = 90	Test: 0.63 Train: 1 PCA = 90	Test: 0.88 Train: 1 PCA = 60	Test: 0.35 Train: 1 PCA = 90
Gradient Boosting	Test: 0.31 Train: 0.78 no PCA	Test: 0.65 Train: 1 no PCA	Test: 0.85 Train: 1 no PCA	Test: 0.23 Train: 1 no PCA
SVM	Test: 0.3 Train: 0.78 PCA = 20	Test : 0.66 Train: 0.87 PCA =10	Test: 0.82 Train: 0.94 PCA = 10	Test: 0.30 Train: 0.78 PCA = 30
Random Forest	Test: 0.37 Train: 1 no PCA	Test: 0.66 Train: 1 no PCA	Test: 0.85 Train: 1 no PCA	Test: 0.36 Train: 1 no PCA

Table 8. Classification accuracy results using Leave-one-out cross-validation

Classifier/Labels	5 CL levels	2 CL levels	Stress/no Stress	2 CL level X Stress/no Stress
Basic Neural Network (MLP)	Test: 0.24 Train: 0.97 PCA = 30	Test: 0.67 Train: 1 PCA = 90	Test: 0.6 Train: 1 PCA = 30	Test: 0.26 Train: 0.94 PCA = 30
Gradient Boosting	Test: 0.25 Train: 0.98 PCA = 10	Test: 0.56 Train: 0.98 no PCA	Test: 0.54 Train: 1 PCA = 10	Test: 0.24 Train: 1 PCA = 10
SVM	Test: 0.22 Train: 1 no PCA	Test : 0.61 Train: 1 PCA =60	Test: 0.54 Train: 1 no PCA	Test: 0.21 Train: 1 no PCA
Random Forest	Test: 0. 24 Train: 0. 73 PCA = 30	Test: 0.61 Train: 0.86 PCA = 30	Test: 0.51 Train: 0.95 PCA = 10	Test: 0.26 Train: 0.72 PCA = 30

4.2.3 Conclusions

1. Regarding the two different approaches for feature extraction, we attained better results with the automatic feature extraction and filtering technique. Still, it is important to mention that the difference in accuracy was not big in gradient boosting and in random forest. This was very surprising as in the automatic extraction approach a much larger amount of features was calculated. This fact implies that the relationship of the labels to the set of features is not strong as both techniques got similar results.

The following conclusions refer only to the automatic feature extraction results.

2. All models had high accuracy in predicting stress / no stress condition. We attained better result when training simple machine-learning classifiers with the selected features than training RNN's with raw data sequences.
3. Specific subjects have more distinguishable or imperceptible GSR and ECG responses; due to these individual differences, our models were not able to classify new data from new subjects in the Leave-one-out cross-validation technique.
4. As seen in Table 7, test accuracy on classifying cognitive load was unsatisfactory. After we reduced inputs dimensions, we were surprised by the difference between training and testing result (we received a very high training accuracy, but a relative low test accuracy). These results are generally due to over-fitting. However, in simple models we used, like SVM, we expected to get similar results for test data and train data for two main reasons:

first, the ratio between the amount of available training data and the number of extracted features is considered good. (According to the literature, a "thumb rule" for a "good" ratio between the training data-set size and the feature space is at least 10 [15]). Second, both the train data and test data are sampled from the same distribution meaning that the test data is relatively similar to the train data.

At first, we taught there might be an error in the way we split the data to train and test but after additional tests, we ruled out this explanation. To our understanding, the low-test result is likely due to two primary reasons: First, a different bug in our code we did not find. Second, According to [15] overfitting can be a problem even for low-dimensional data, especially if the relationship of the labels to the set of features is not strong. To our opinion, this might be the problem in our data set.

5 RT Application

5.1 Description of the Product from the Perspective of the User

When the user launches the application he has to choose between two running mode: static test mode or real-time test mode.

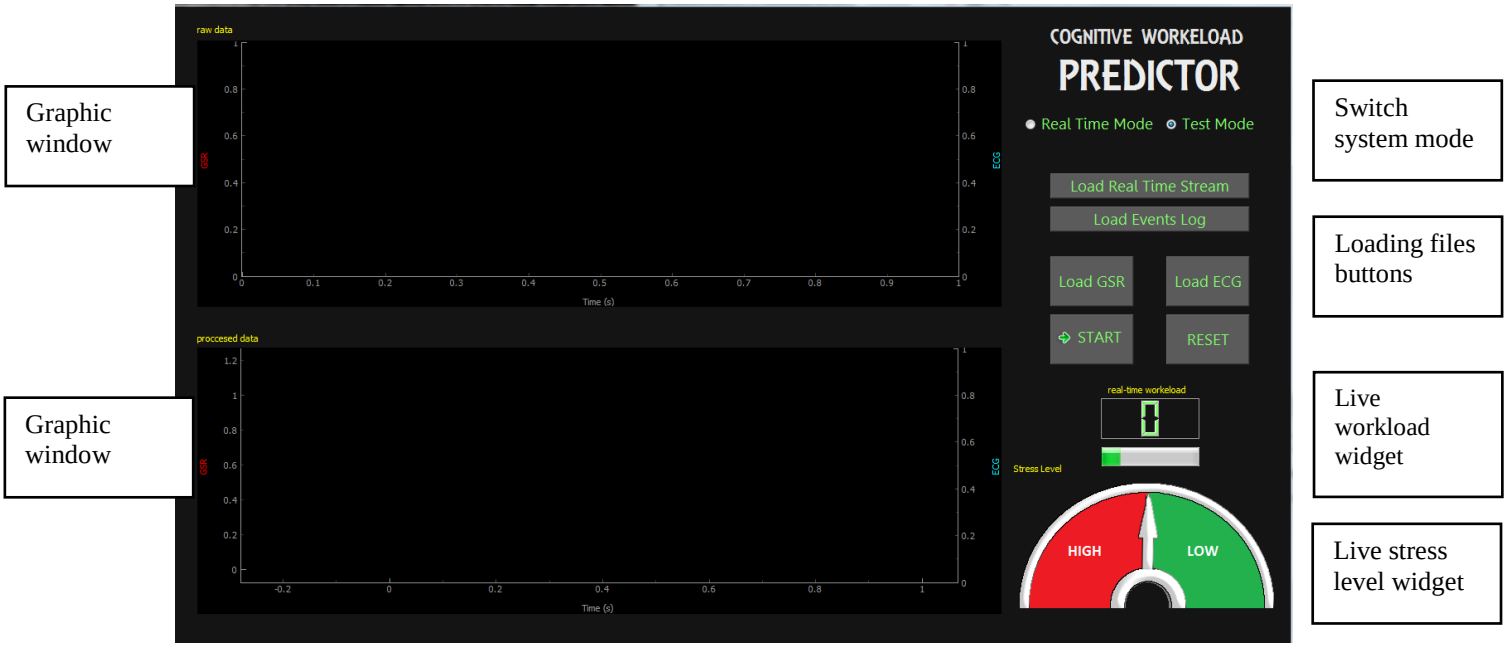
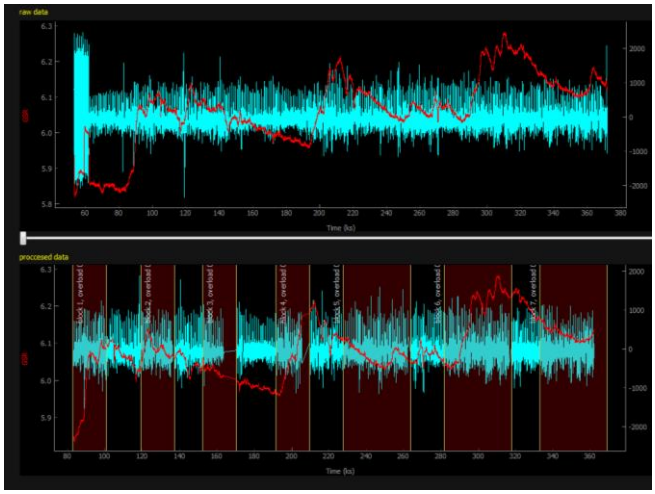


Figure 9. The application-starting screen.

5.1.1 Static Test Mode

When in static test mode the user has to load into the system 3 data files recorded from experiment session of a subject: ECG & GSR signals and an events log of the experiment. After loading, the raw signals are presented on the upper graphic window of the application. The user will get the system predictions of the subject's cognitive load and stress levels of the different sessions throughout the experiment by clicking on the start button.

In the background, the system processing the signals data files (filtering, down-sampling, etc.,) and after dividing the data into blocks by extracting each experiment session duration times from the events log file. When finished, the system predict each block session cognitive overload and stress level by classifying all the segment-size signals on each block and taking the median of them.



The user can see on the lower graphic window the processed ECG & GSR signals, partitions of the data to sessions and the system's

predictions of cognitive load and stress levels for each session.

In anytime the user may reset the system by clicking on the reset button.

5.1.2 Real Time Test Mode

When in real time test mode, the user has to connect the system to live data stream of an experiment. In our lab, the system that used for synchronizing and streaming data from the different physiological devices to the computer called LSL (Lab Streaming Layer). When connecting to the LSL serial port, the computer receives synchronizing data from the physiological devices and time. In our project we simulate real time by reading a data file we prepared simulating the same procedure as the LSL does, so in real using the user can connect the system to the LSL serial port instead.

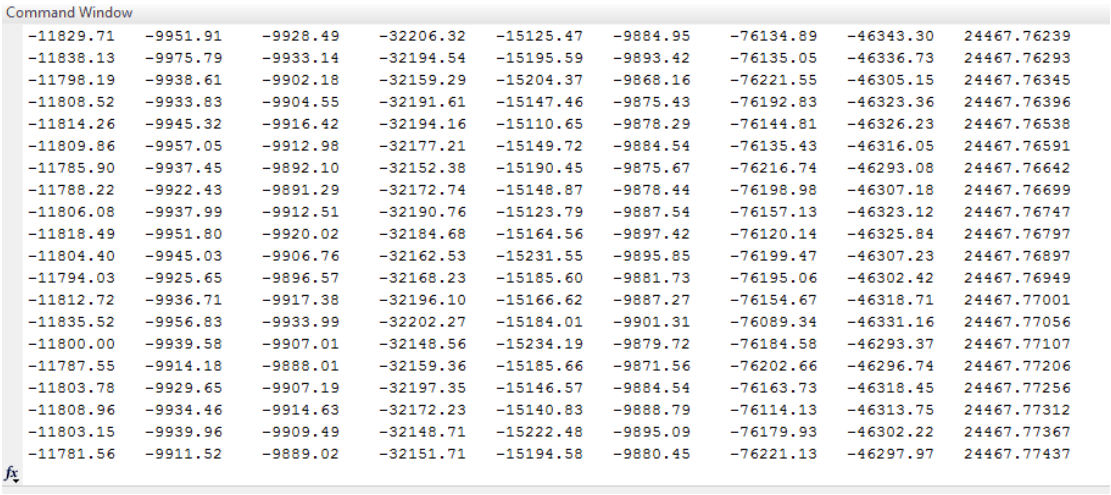


Figure 10. Receiving data by the LSL: Data from different layers appears from the first to the eighth column, whereas the last column displays the time in milliseconds.

When the user start streaming data of the experiment (we take care of the ECG & GSR signals and the events) the raw signals are presented on the upper graphic window of the application and been continuously updated with new data. In parallel, the system counting the size of the signal that been accumulated. If a segment size reached (about 8 seconds), the system opening new thread for processing the segment data, extracting features and making predictions of its cognitive load and stress levels. The processed ECG & GSR signals are updating on the lower graphic window and the system's predictions are being presented on the cognitive workload level and stress level graphic widgets.

Meantime the system in parallel also following the duration time of the different experiment sessions for making predictions of the cognitive load and stress levels of the hole sessions. When calculating a segment prediction, if it's marked to be a part of a block session the results are being added to lists of the block segment's predictions. When event of end of block session is received from the stream, the system predict the current block session cognitive overload and stress level by taking the median of predictions on its prediction list and the result with the duration of the block are being presented on the lower graphic window.

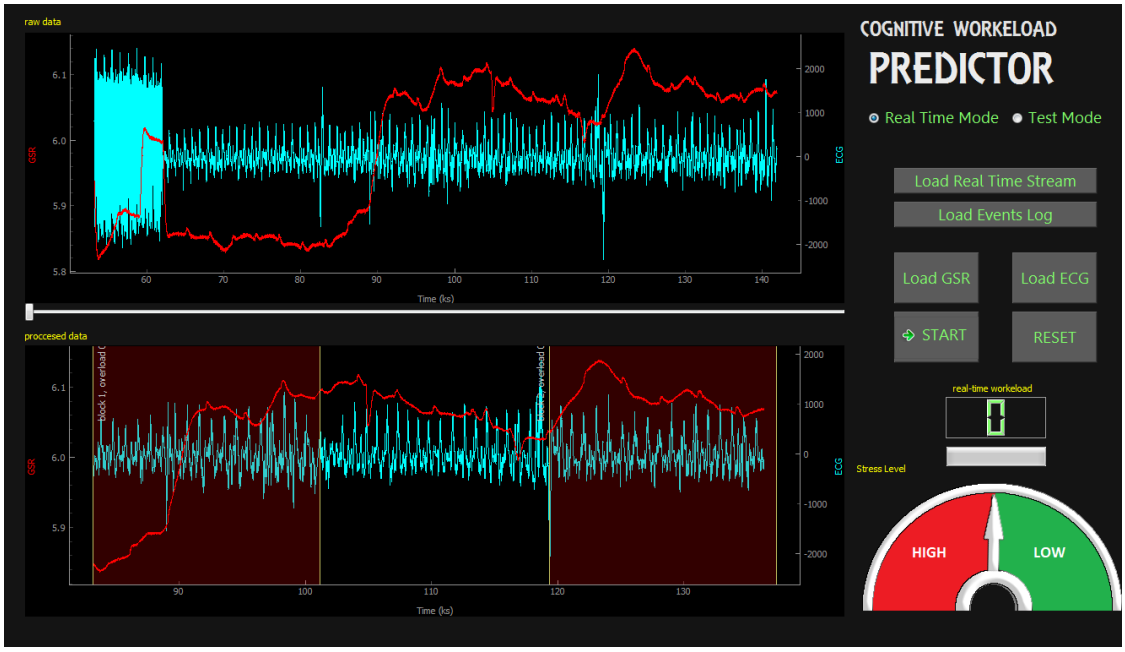


Figure 11. A real time session

The system stops its process when end of session event is received from the data stream. In anytime the user may reset the system by clicking on the reset button.

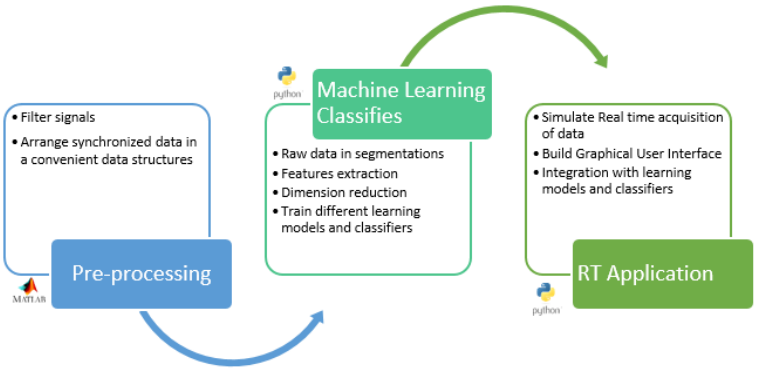
6 Conclusions and Future Work

Modeling cognitive load level is a complex mission as it is hard to define, it depends on many different factors (such as individual differences) and there is no "right way" to measure it.
An important limitation of our project, and of physiological classification problems as a whole, is the difficulty to obtain accurate ground truth labels. Machine-learning supervised learning algorithms rely on correct labels; however, the process of creating data-set containing physiological signals for different mental and/or psychological states is usually based on an external experiment that might not lead subjects to the desired state.
For future work, we think it is important to implement additional filtering and normalization to the data in order to improve generalizability of the classifiers. In addition, we think out feature extraction could be improved.
Regarding the low accuracy we achieved in the RNN models, we think that adding more data from additional subjects and applying a dimensionality reduction to raw data will improve their performance.
Furthermore, we believe that in the learning stage, adding features that are more performance dependent, such as time response will help to achieve better results.
Additionally, in future research it will be interesting to use clustering methods on the data, instead of classifying it by labels, and see whether we accomplish better results that are less individually depended.
Finally, this project exposed us to an existing and challenging research field that combines knowledge from different domains in Neuroscience, Psychology and Computer Science.

7 Appendix A: High Level Design Explanation

In this chapter, we will present a high-level design explanation. Our project's code is written in Matlab and Python and is divided to three logical parts: Pre-processing, machine-learning classifiers and RT Application. Each part will be described further in this section.

Figure 11: Our project's main work content



7.1 Pre Processing

This part was written in Matlab. It contains two main classes, one main script, and several functions.

Classes:

1. Block – containing all the relevant information of a single block of the experiment, including - condition (stress/no-stress), ring size, time block started, etc.
2. Subject: with the fields:
    a. Channels: contains the raw data of both signals of one subject (ecg, gsr) corresponding to the blocks order.
    b. Blocks: array of the 26 blocks (made from blocks objects that have all the fields, which can be used for extracting different labels type: level/ ring size etc.).

The main module of this section is a script called load_data, this scripts calls several functions and is responsible for the following:

1. Loads all subject's raw data files according to an Excel file. The Excel file determinates which subjects to upload (a binary matrix). For each subject that its data is uploaded the script is responsible for:
    - Down samples GSR, Filter ECG and GSR raw data and synchronizes between them.
    - Creates a Subject structure with all the relevant fields.
    - Creates segments of raw data from both signals according to global parameters of segment length and window overlap.
    - Creates an array of labels for each segment (according to the four options of classifying as described in section XX).
    - Saves raw data and labels in a CSV file.

7.2 Machine learning classifies

This part is written in Python. It contains three main modules:

- RNN_classifiers – implementation of all RNN models.
- Feature_exraction – implementation of feature extraction
- Other_classifiers - implementation of all other classifiers: SVM, random forest, gradient boosting, MLP neural network.

We used the following packages: Tensorflow, Seaborn, Pandas, Tsfresh, Numpy, Scipy, Sklearn, Pickle.

7.3 Real-Time Predictor Application

Designing a system capable of monitoring its user's mental workload can evaluate the suitability of its interface and interactions for user's current cognitive status and stress state, and properly change them when necessary.

In this section, the real-time application prediction will be described shortly.

The predictor application has two running modes: Real time mode and Off-line test mode and predicts cognitive load level (in a scale of 1-5) and binary stress state.

The prediction is based on ECG and GSR signals with two trained models from the experiment: Random Forest classifier and Basic Neural Network (MLP) for the CL 5-level prediction and for stress binary state prediction, respectively.

7.3.1 Software Architecture:

The application acquires a stream of data, do some processing (filtering, features extraction, etc.), display the cleared data signals and the predictions on the screen, optionally in real-time. For real-time implementation, we used the QTimer class to make repeated calls.

Real-time prediction and visualization is a key component of our program. To satisfy our requirements we needed a fast and portable graphics library. Since we implemented the GUI in PyQT, we also required that the graphics library should be embeddable in this framework. We used PyQtGraph [16]. It is a pure Python implementation, with a focus on speed, portability and a rich set of features. PyQtGraph is designed to do real-time plotting and interactive image analysis. It is built on top of PyQt4/PySide, giving easy integration and full compatibility with the Qt framework. This allows using tools like Qt Designer to design the GUI. Using Qt Designer and the examples provided with the PyQtGraph library, it is easy to configure and customize the widgets. PyQtGraph is also built on top of NumPy, facilitating and improving the performance of the manipulation of numerical data.

Reading the live stream data was simulated by using prepared modified stream file. (With ecg, gsr signals and epoch events). After accumulating stream data in length of the time window segment used to train the models (8.8 sec), processing the data stage starts including: filtering and extracting features with features optimization and dimension reduction with PCA. After

processing, starting to classify the data segment. When finishing accumulating data of a complete block session: calculating the final prediction of the current block based on all the time window segments' predictions calculated from the block –the median.

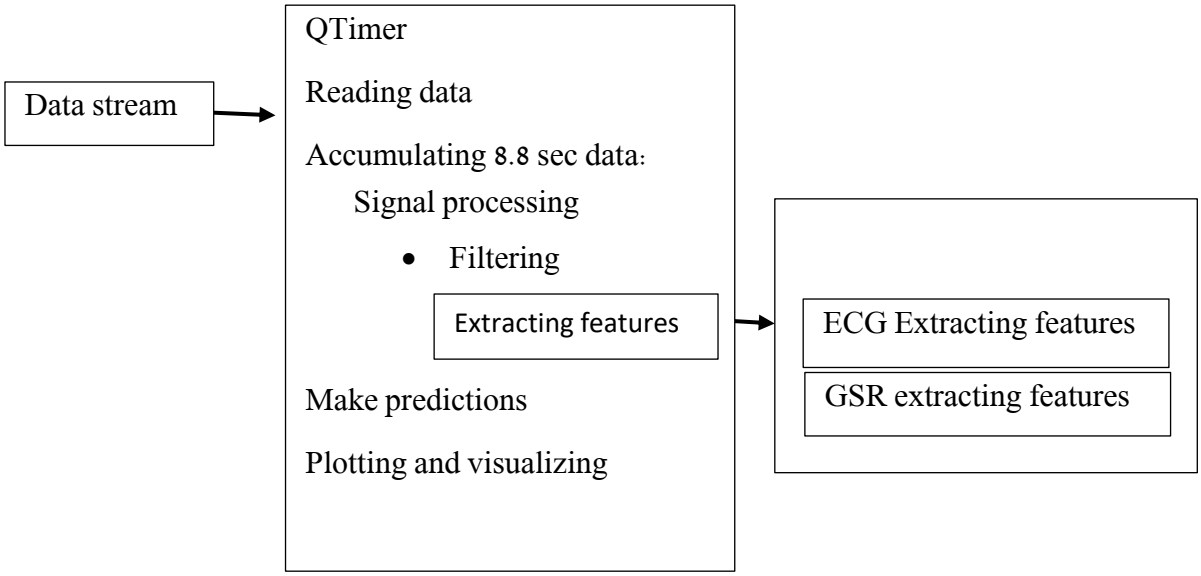


Figure 12: Diagram of the software architecture

8 References

1. Sweller, J.: Cognitive load theory, learning difficulty, and instructional design. Learning and instruction, vol. 4, issue 4, 295-312 (1994)
2. Engstrom, J., Johansson, E., Ostlund, J.: Effects of visual and cognitive load in real and simulated motorway driving. Transportation Research Part F: Traffic Psychology and Behaviour, vol. 8, issue 2, pp. 97-120 (2005)
3. Wilson, G. F.: An analysis of mental workload in pilots during flight using multiple psychophysiological measures. The International Journal of Aviation, vol. 12, pp. 3-18 (2002)
4. Dickerson, S. S., Kemeny, M. E.: Acute Stressors and Cortisol Responses: a theoretical integration and synthesis of laboratory research. Psychological Bulletin, vol. 130, issue 3 pp. 355-391 (2004)
5. Setz, C., Arnrich, B., Schumm, J., La Marca, R., Tröster, G., Ehlert, U.: Discriminating stress from cognitive load using a wearable EDA device. IEEE Transactions on Information Technology in Biomedicine, vol. 14, issue 2, pp. 410-417 (2010)
6. Heine, T., Lenis, G., Reichensperger, P., Beran, T., Doessel, O., & Deml, B. (2017). Electrocardiographic features for the measurement of drivers' mental workload. Applied ergonomics, 61, 31-43.
7. 5. Healey, J., Picard, R.: SmartCar: detecting driver stress. International Conference on Pattern Recognition, vol. 4, pp. 218 - 221. (2000)
8. Nourbakhsh, N., Wang, Y., Chen, F., Calvo, R. A.: Using galvanic skin response for cognitive load measurement in arithmetic and reading tasks. Proceedings of the 24th Australian Computer-Human Interaction Conference. ACM, pp. 420-423 (2012)
9. 1920 Niedermeyer, Ernst and 1935 Lopes da Silva, F. H. Electroencephalography : basic principles, clinical applications, and related fields. Philadelphia ; London : LippincottWilliams Wilkins, 5th ed edition, 2005. Includes bibliographical references and index.
10. Malik M, Bigger JT, Camm AJ, Kleiger RE, Malliani A, Moss AJ, Schwartz PJ: Heart rate variability: Standards of measurement, physiological interpretation, and clinical use. Eur Heart J 1996, 17: 354-381
11. Zheng, B. S., Murugappan, M., & Yaacob, S. (2013, April). FCM clustering of emotional stress using ECG features. In *Communications and Signal Processing (ICCSP), 2013 International Conference on* (pp. 305-309). IEEE.
12. Liu, M., Fan, D., Zhang, X., & Gong, X. (2016, November). Human emotion recognition based on galvanic skin response signal feature selection and svm. In *Smart City and Systems Engineering (ICSCSE), International Conference on* (pp. 157-160). IEEE.
13. Belogiannis, T. (2012). Individual Stress Diagnosis from Skin Conductance sensor signals.
14. Benjamini, Y. and Yekutieli, D. (2001). The control of the false discovery rate in multiple testing under dependency. Annals of statistics, 1165–1188

15. . Subramanian, J., & Simon, R. (2013). Overfitting in prediction models–is it a problem only in high dimensions?. *Contemporary clinical trials*, *36*(2), 636-641.
16. L. Campagnola. PyQtGraph. Scientific Graphics and GUI Library for Python, http://www.pyqtgraph.org/