Cognitive Load Detection from Wrist-Band Sensors

Xiling Li
xl32@uw.edu
University of Washington
Tacoma, WA

Martine De Cock[1]
mdecock@uw.edu
University of Washington
Tacoma, WA

ABSTRACT

In this work, we use machine learning (ML) to detect the cognitive load of a user based on sensor data from a smart wrist-band, sampled during 30 seconds. The data is provided by a challenge at the UbiTtention 2020 workshop of UbiComp 2020; in this paper we describe UW's participation (team Lynx). The defining characteristic of our approach is in the custom features that we extract from the time series. While we do not have any labeled instances for the test users, the fact that we do have multiple time series for each test user, allows us to extract features that measure how much individual time series deviate from the user's average. We combine this extracted information with other time series' features from the literature. We further use feature selection based on Gini impurity and state-of-the-art techniques for training ML models such as Logistic Regression, (Boosted) Decision Trees, Random Forests, and Support Vector Machines, yielding $\sim$ 63% accuracy by 6-fold cross-validation.

CCS CONCEPTS

• Human-centered computing → Ubiquitous and mobile computing; **• Computing methodologies → Feature selection**; **Classification and regression trees**; **Bagging**.

KEYWORDS

cognitive load, attention management, binary classification, time series analysis, feature engineering, machine learning

ACM Reference Format:
Xiling Li and Martine De Cock[1]. 2020. Cognitive Load Detection from Wrist-Band Sensors. In *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers (UbiComp/ISWC '20 Adjunct), September 12–16, 2020, Virtual Event, Mexico.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3410530.3414428

1 INTRODUCTION

The seemingly never-ending stream of notifications received through mobile devices can be detrimental to one's productivity, and in some situations even dangerous. The more aware devices become of the user's current cognitive load, the better these devices can adapt their interactions with the user, and avoid unfortunate disruptions.

In the field of Ubiquitous Computing, researchers started to study these phenomena in the context of attention management [1]. Cognitive load (CL), which is originally a psychological term to describe how much working memory resources are used, is used as a metric to enable the design of intelligent devices that adapt their interactions with the user. To this end, it is important for the device to be aware of the user's current CL.

Our focus in this paper is on the use of machine learning (ML) to infer the CL of a user from physiological data collected through sensors in wearable activity trackers, such as smart wrist-bands. These wearable devices are already popular, and the users' physiological data can be measured through them in a non-invasive manner, making smart wrist-bands viable for wide deployment of CL tracking. We use an open-source dataset provided by the UbiTtention 2020 challenge at UbiComp 2020[1]. For each user, the dataset contains sensors measurements for 4 categories of physiological features, sampled at 1 Hz rate during 30 second time windows. The goal is to infer from these measurements whether the user is experiencing CL or not. To this end, we train binary classifiers based on Logistic Regression, (Boosted) Decision Trees, Random Forests, and Support Vector Machines. We find that, while ML models trained directly on the raw sensor measurements are better than random guessing, there are substantial further accuracy improvements to be gained through feature engineering.

After presenting the dataset and problem description in Section 2, in Section 3 we give an overview of the feature transformation, feature extraction, and feature selection techniques that we included in this study. Section 3 also contains a description of the ML model training algorithms, and how we combined them with the feature engineering techniques in a variety of different end-to-end pipelines for CL inference, experimental results of which are detailed in Section 4. The results that we obtain are at par with results published previously in the literature for similar tasks [8, 13].

2 PROBLEM DESCRIPTION AND DATASET

Cognitive load (CL) is related to physiological responses of the human body [13], hence it is reasonable to try to infer CL from physiological data collected from smart wrist-bands. The UbiTtention2020 dataset [7] contains data for 18 train users and 5 test users. As an illustration, the data for one of the users is visualized in Fig. 1. The data for this particular user consists of 41 times series (instances), each sampled during a 30 seconds time window at a 1 Hz rate. 4 different kinds of physiological responses are measured,

[1] M. De Cock is a Guest Professor at Ghent University, Dept. of Applied Mathematics, Computer Science and Statistics.

UbiComp/ISWC '20 Adjunct, September 12–16, 2020, Virtual Event, Mexico
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-8076-8/20/09…$15.00
https://doi.org/10.1145/3410530.3414428

[1]https://www.ubittention.org/2020/

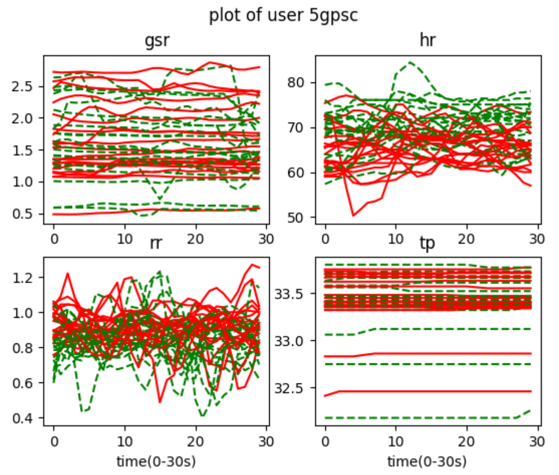

Figure 1: Visualization of data for user with id *5gpsc*, Red solid lines are for instances with label 0 (CL) and green dashed lines are for instances with label 1 (non-CL, i.e. rest).

which explains why there are 4 different pictures in Fig. 1. The measurements for instance i of user u ($i = 1, \ldots, 41$ for the user in Fig. 1) form a numerical vector $\mathcal{D}_u(i)$ of length 120 that itself is composed of 4 vectors of length 30, one for each kind of physiological data:

$$\mathcal{D}_u(i) = \langle \mathcal{G}_u(i), \mathcal{H}_u(i), \mathcal{R}_u(i), \mathcal{T}_u(i) \rangle$$

where

- $\mathcal{G}_u(i)$: galvanic skin response (gsr)
- $\mathcal{H}_u(i)$: heart rate (hr)
- $\mathcal{R}_u(i)$: rr intervals (rr)
- $\mathcal{T}_u(i)$: skin temperature (temp)

All $\mathcal{G}_u(i)$ vectors for the user with id *5gpsc*, $i = 1 \ldots 41$, are depicted in the top left picture in Fig. 1, all $\mathcal{H}_u(i)$ vectors in the top right picture etc. Red solid lines are for instances with label 0 (CL) and green dashed lines are for instances with label 1 (non-CL, i.e. rest).

As mentioned above, there are 18 such users in the train data. The number of time series (instances) per user varies between 14 and 41, making up a total of 632 instances in the train data. 315 of those instances have label 0 and 317 instances have label 1, hence the train data is well balanced.

The test data consists of similar time series information for 5 users. For each user in the test data, the number of available time series varies between 38 and 39. The difference between the train data and the test data is that the time series in the test data are not labeled. The inference problem addressed in this paper can hence be formalized as: given a set $\mathbb{D}_u = \{\mathcal{D}_u(i) \mid i = 1, \ldots, n_u\}$ of n_u instances for test user u, assign a label 0 or 1 to each $\mathcal{D}_u(i)$ in $\mathbb{D}_u$. Note that it is not immediately possible to train personalized ML models for the test users in a supervised fashion, since *no* labeled instances are provided at all for any of the test users. The fact that multiple time series need to be labeled per test user does allow us to extract statistical information per test user which can be used as "per user" features for the classifier (see Section 3.1.2).

Since we have no labels available for the users in the test data (they are kept hidden by the UbiTtention 2020 organizers for the purpose of the competition), we can not evaluate the accuracy of the methods proposed in this paper on the test data. Instead, we evaluate our methods with 6-fold cross-validation (CV) on the 18 train users. For each fold, we use train data from 15 users and we use the data from the remaining 3 users as validation data.

3 METHODOLOGY

3.1 Feature Engineering

In the remainder of this paper, we refer to the feature vectors $\mathcal{D}_u(i)$ from Section 2 as RAW feature vectors. In ML, substantial improvements in accuracy are often achieved by preprocessing raw feature vectors before model training. In this paper we evaluate the effect of a variety of feature engineering methods which draw on 3 main techniques: transformation, feature extraction, and feature selection. The transformation methods that we use, namely fast Fourier transform (FFT) and sliding mean filter (SMF), are specific to preprocessing of time series data, while feature extraction and feature selection are well known approaches that are very widely applicable. Feature extraction (FE) aims at constructing new features from existing features, while feature selection (FS) aims at identifying subsets of features that are highly relevant for the inference problem at hand [9]. ML models trained on a set consisting of extracted and selected features are expected to perform as well as or even better than ML models trained on raw data.

For feature selection, we use the filter approach based on Gini impurity (Section 3.1.5). This means that for each feature we compute a score, and we retain the features with the "best" score. For feature transformation and extraction, we use a variety of different approaches, many of which are well known in the literature (Section 3.1.1 to 3.1.4).

Table 1 contains an overview of all features. The RAW features were described in Section 2. Below we provide more details about the other features. Since the features that we consider vary quite a lot in numeric scale, e.g. gsr data < 10 while temp data > 50, we normalize all extracted feature values before model training in Section 3.2. To this end we use z-score normalization, which means that from each feature value we subtract the mean of that feature, and we divide the result by the standard deviation of the feature.

3.1.1 Fast Fourier Transform based Features (FFT)

FFT is an method to convert a signal from the time domain into the frequency domain, by computing the discrete Fourier transform (DFT) of a sequence [15]. Our use of FFT based features is inspired by the work of Baldominor et al.[2], who use physiological time series data collected from ubiquitous devices (smart phone/wristband) for activity recognition, which, like our CL detection task, is also a classification problem.

In our approach, for each instance $\mathcal{D}_u(i)$, we transform the 4 raw feature vectors $\mathcal{G}_u(i)$, $\mathcal{H}_u(i)$, $\mathcal{R}_u(i)$, and $\mathcal{T}_u(i)$, and subsequently extract the same statistical information features from the transformed vectors as Baldominos et al. [2]. For each instance, this results in 4 features of each kind, i.e. for gsr, for hr, for rr, and for temp. These features are marked as FFT features in Table 1.

As in [2], we also extract the mean and standard deviation from each raw feature vector. While these are computed directly on the raw data and do not involve an FFT step, we mark them as FFT

Table 1: Overview of features

Features	Description	Method	#
raw	Unprocessed sensor measurements	RAW	120
mean	Mean of raw data	FFT [2]	4
stdDev	Std dev of raw data	FFT [2]	4
median	Median of FFT data	FFT [2]	4
lowQuant	Lower quantile of FFT data	FFT [2]	4
upQuant	Upper quantile of FFT data	FFT [2]	4
skew	Skewness of FFT data	FFT [2]	4
kurtosis	Kurtosis of FFT data	FFT [2]	4
diffMean	Mean diff of inst and user	PUF	4
gMean	Mean of SMF data	SMF [13]	1
gStd	Std dev of SMF data	SMF [13]	1
gLowQuant	Lower quantile of SMF data	SMF [13]	1
gUpQuant	Upper quantile of SMF data	SMF [13]	1
gQuantDev	Quantile deviation of SMF data	SMF [13]	1
gSum	Sum of SMF data	SMF [13]	1
stdDevDiff	Std dev diff between adjacent rr data	HRV [8]	1
sqrtMean	Sqrt of mean of square of diff between adjacent rr data	HRV [8]	1
perctDiff	% of diff between adjacent rr data	HRV [8]	29

anyway in Table 1 for simplicity: all features marked as FFT in Table 1 coincide with the feature set used in [2]. Unlike in [2], we compute FFT directly on each of $\mathcal{G}_u(i)$, $\mathcal{H}_u(i)$, $\mathcal{R}_u(i)$, and $\mathcal{T}_u(i)$ as a whole instead of using a sliding window on each segment of data. In our experiments, we used the fftpack package of scipy [16].

3.1.2 *Per-User Features (PUF)*

The features in Section 3.1.1 are extracted from each instance $\mathcal{D}_u(i)$ in isolation. In addition to this, we propose the extraction of per-user features that measure how the "Mean of raw data" for a given instance i for user u differs from the average mean across all instances j for that user. For instance, letting $\overline{\mathcal{G}_u(j)}$ denote the mean value of $\mathcal{G}_u(j)$ (which in itself is one of the FFT features), we compute a "difference of mean" feature as follows:

$$\text{diffMean}(\mathcal{G}_u(i)) = \overline{\mathcal{G}_u(i)} - \frac{1}{n_u}\sum_{j=1}^{n_u}\overline{\mathcal{G}_u(j)} \tag{1}$$

where u is the index of the user and i is the index of the instance for which the feature value is computed. Note that, while the second term on the right hand side of Formula 1 relies on information about all instances for user u, the diffMean feature values are still computed for each instance, and may differ from one instance to the next. Indeed, the diffMean of each instance describes how far the instance lies from the overall mean of the user in each category. Computing diffMean for all 4 categories of physiological data yields the 4 PUF features in Table 1.

3.1.3 *Sliding Mean Filter Method based Features (SMF)*

Pejović et al. [13] use SMF as a smoothing method to remove noise of wristband sensor time series data. They apply this specifically to galvanic skin response data (gsr) and subsequently extract statistical information features from the smoothed data. We include 6 features of this kind in our study, as shown in Table 1: gMean, gStd, gLowQuant, gUpQuant, gQuantDev, gSum.

3.1.4 *Heart Rate Variability Features (HRV)*

HRV is a method used by Gjoreski et al. [8]. We include the "adjacent rr" features from [8] in our study, as these showed promise in our preliminary experiments. Recall that each $\mathcal{R}_u(i)$ vector consists of a time series of 30 raw rr feature values, i.e. 29 pairs of adjacent rr values. stdDefDiff and sqrtMean in Table 1 capture statistical information computed based on the difference between adjacent rr values. We also compute 29 perctDiff features.

3.1.5 *Gini Impurity Method (GINI)*

Gini impurity (GINI) is a well known criterion for picking the best feature as a new node for each iteration of decision tree learning in the CART algorithm [4]. GINI can be used in a similar fashion as a scoring criterion in the "filter approach" to feature selection. The GINI value (G) of a set S of labeled examples is computed as follows:

$$G(S) = \sum_{i=1}^{N} p_i \cdot (1 - p_i) = \sum_{i=1}^{N}(p_i - p_i^2) = 1 - \sum_{i=1}^{N} p_i^2 \tag{2}$$

where N is the number of classes ($N = 2$ in our case) and p_i is the probability of a randomly chosen element from S to belong to class i. A numeric feature f, like the ones we have in our study, is considered "good" if it can reduce the GINI impurity a lot for a suitable threshold α in the range of feature values of f. To measure this, one constructs $S_1 = \{x \mid x \in S \wedge f(x) \leq \alpha\}$ and $S_2 = \{x \mid x \in S \wedge f(x) > \alpha\}$, and computes

$$G_{split}(S, f, \alpha) = \frac{|S_1|}{|S|} \cdot G(S_1) + \frac{|S_2|}{|S|} \cdot G(S_2) \tag{3}$$

An optimal threshold α for f can be found by sorting the data S based on the feature f, identifying instances that differ in class labels, and generating candidate thresholds midway between the corresponding values of f. See Mitchell [10] for more details.

After computing the GINI value for each feature, we sort all features in ascending order based on GINI values and select the top k features. In our experiments, we set k to be approximately one tenth of all features because of relatively high validation accuracy compared with other ks.

3.2 Training Methods

In the training step, we use traditional ML algorithms which we briefly recall below. Unless specified otherwise, we use the default hyperparameter settings of the sklearn library [12]. Each ML algorithm takes as input a set of labeled training examples $((x_1, x_2, \ldots, x_m), y)$. The feature values $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ are real numbers corresponding to a subset of the features in Table 1, while the class label y is 0 (CL) or 1 (non-CL). The goal is to learn a function from the data that maps previously unseen feature values to a corresponding class label.

Logistic Regression. A LR classifier corresponds to a function of the form $g(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$. In this expression, σ is the logistic function $\sigma(z) = 1/(1 + e^{-z})$ and $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ is the input feature vector. The weight vector $\mathbf{w} = (w_1, w_2, \ldots, w_m)$ and the bias b are model parameters that are learned during an iterative training process. If $g(\mathbf{x}) < 0.5$, instance $\mathbf{x}$ is classified as label 0, and label 1 otherwise.

(Boosted) Decision Trees. A DT is a tree structure in which each internal node tests the value of a particular feature against a corresponding threshold and branches according to the result. Each leaf node specifies one of the classes. The result of the classification is the class associated with the leaf reached from traversing the tree, starting from the root. During training, a DT is grown in a top-down manner. For each internal node, a feature f is chosen that, after splitting, will help reduce the "impurity" the most in the set of training examples that have reached that node. In Section 3.1.5, we briefly sketched how this works in CART algorithm [4]. A similar technique is used in the C4.5 algorithm [14], in which entropy is used instead of Gini impurity. The entropy of a set of examples S is defined as $H(S) = -\sum_{i=1}^{N} p_i \cdot \log_2(p_i)$ where N is the number of classes ($N = 2$ in our case) and p_i the probability of a randomly chosen element from S to be class i. This is the technique we use for DT model training in Section 4.

In addition to single DT models, we use boosting-based ensemble models trained with the AdaBoost algorithm (AB) [6]. An ensemble is a set of DTs used together to classify new instances. The DTs in a boosted model are trained in sequence; the training process for each DT gives more importance to instances that were misclassified by DTs trained earlier in the sequence. While boosted DT models are computationally more expensive to train than single DTs, they usually yield higher accuracy, as we also see in our experiments.

Random Forests. A RF is an ensemble of DTs that are each trained on a bootstrap replicate of the original data. To create a replicate of a data set S with n instances, one samples n times, with replacement. In addition, during the DT growing process, the best feature for each node is chosen from a randomly selected subset of the features instead of from all features. This process of *bagging* (for instances) and *subspace sampling* (for features) makes the DTs in the ensemble sufficiently different from each other, allowing RFs to yield higher accuracy than individual DTs [3].

Support Vector Machines. A SVM [5] with a linear kernel, as we use in this study, corresponds to a linear decision boundary $h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$. The parameters $\mathbf{w}$ and b are learned during an optimization process aimed at finding a "maximal margin" hyperplane that separates the two classes and that is as far away as possible from each training example. The tolerance for training examples to be on the wrong side of the decision boundary is controlled by a penalty parameter C; in our experiments $C = 1$.

Voting-based Ensemble. Voting-based Ensemble is a method to combine models by classifying an instance based on the majority of labels inferred by all ML models [11]. This method may improve accuracy when some single models perform not very well.

We did not include deep learning methods in our study because of the small size of the dataset. Baldominos et al. [2] compared the use of convolutional neural networks (CNNs) with traditional ML algorithms for classification of time series data from smartphone/wristband and observed that the CNN was outperformed by traditional methods.

3.3 Approaches

We combine the ML algorithms from Section 3.2 with the feature engineering methods from Section 3.1 in different end-to-end pipelines which we categorize in 5 general approaches:

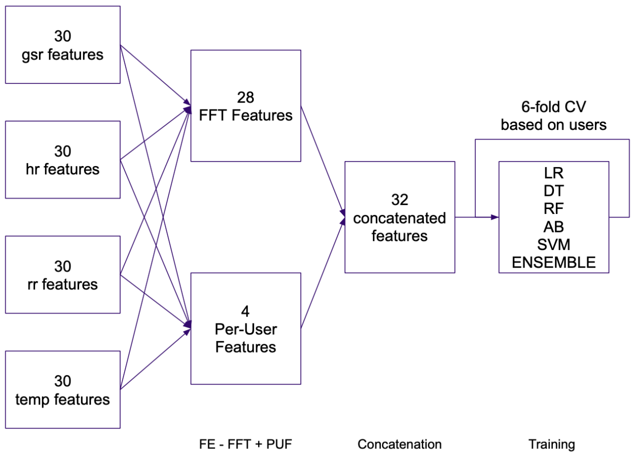

Figure 2: Approach on extracted data (FFT + PUF)

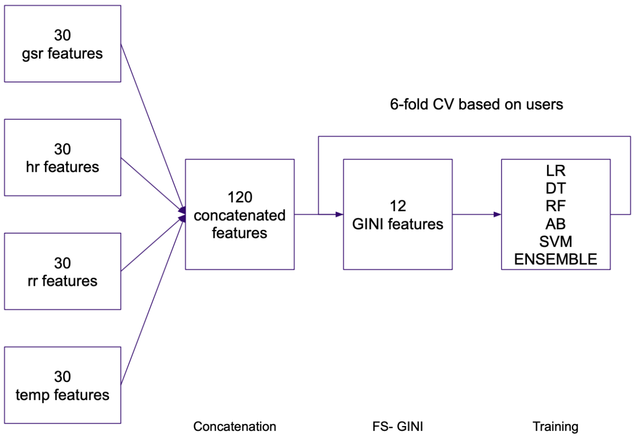

Figure 3: Approach on selected data (FS(RAW))

(1) **Approach on raw features (RAW).** In this approach (Fig. 3 without FS block), we concatenate features of all physiological data along columns to get $\mathcal{D}_u$ (120 features) for each user u. Then we train the ML models without feature engineering.
(2) **Approach on extracted features (FE).** In this approach (Fig. 2 as example of FFT+PUF), we use FE only. We apply 3 combinations of FE: FFT+PUF/SMF+ HRV/FFT+PUF+SMF+HRV. Then we train the ML models on the concatenated data extracted by those combinations with 32/37/69 features. For comparison, we train models on RAW+FFT+PUF+SMF+HRV with 189 features as well.
(3) **Approach on selected features (FS(RAW)).** In this approach (Fig. 3), we use FS only. We do FS on $\mathcal{D}_u$ with GINI to select one tenth of the RAW features with smallest GINI values for each fold (approx. 9 from rr data and 3 from hr data are selected).
(4) **Approach on extracted and selected features (FS(RAW) + FE).** In this approach (Fig. 4 as example of FS(RAW)+FFT+PUF), we combine both FE and FS separately. As in approach

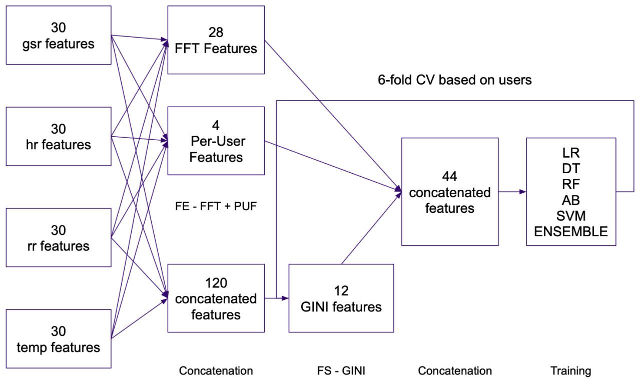

Figure 4: Approach on extracted and selected data FS(RAW) + FFT + PUF

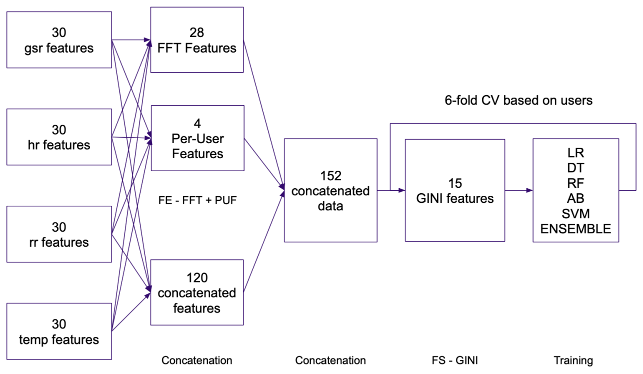

Figure 5: Approach on extracted-then-selected data FS(RAW + FFT + PUF)

2, we extract features with 3 combinations. Like approach 3, FS for each fold with GINI selects 12 features from $\mathcal{D}_u$. Then we concatenate selected data with extracted data of all combinations to get FS(RAW)+FFT+PUF / FS(RAW)+SMF+HRV / FS(RAW)+FFT+PUF+SMF+HRV with 44/49/81 features. Finally, we train the ML models on those concatenated data.

(5) **Approach on extracted-then-selected features (FS(RAW + FE)).** In this approach (Fig. 5 as example of FS(RAW+FFT+PUF)), we apply FE and then FS. As in approach 2, we extract features with 3 combinations. Then we concatenate extracted data of all combinations with $\mathcal{D}_u$. After that, we do FS for each fold on those concatenated data to get FS(RAW+FFT+PUF) / FS(RAW+SMF+HRV) / FS(RAW+FFT+PUF+SMF+HRV) with 15/15/18 features (selected features are mostly from extracted data). Finally, we train the ML models on those selected data.

4 RESULTS

Table 2 contains the accuracy results of all approaches from Section 3.3, measured with 6-fold CV with respect to users. In each fold we used data from 15 users for training, and data from 3 users for testing. The RF and AB models are ensembles of 20 trees each. The best results of each kind of model architecture from Section 3.2 across the 5 approaches from Section 3.3 is highlighted in bold, identifying approach 4 as the best one.

The importance of feature extraction, i.e. the construction of features that represent the data well, is very clear from Table 2. The approaches that do not involve feature extraction, namely approach 1 and 3, are clearly outperformed by the other approaches in terms of accuracy. Comparing the results of approach 3 with approach 1, one can also observe that applying feature selection to the raw features did not systematically help. This does not mean that feature selection has no merit: comparing the last row of approach 2 with the last row of approach 4, one can see that applying feature selection to the raw data before combining it with the other features improved the accuracy across all model architectures. We observed a similar phenomenon for the combination of the raw features with subsets of the extracted features, i.e. first applying FS to RAW gives better results than not applying FS to RAW for all combinations in approach 4; we omitted results for some of the combinations without FS for conciseness, focusing on the better performing techniques instead.

Comparing approach 4 and 5 shows that feature selection as a filtering step across all features (not just the RAW ones) in advance of model training, as done in approach 5, mostly hurts the accuracy. This is likely because each of the ML algorithms from Section 3.2 have a built-in feature selection technique based on entropy (DT, RF, AB) or a feature weighing technique (LR, SVM) that is applied dynamically and/or looks at weighted combinations of features, while the feature selection technique based on GINI from Section 3.1.5 is a filter approach that measures the importance of each feature in isolation only.

We now turn our attention to some of the best performing models, in particular the SVM model with FFT+PUF+SMF+HRV features of approach 2, and the LR model with FS(RAW)+SMF+HRV features of approach 4. Per-user results of these approaches, as provided in Table 3, show that both approaches have fairly similar misclassification rates, and that there are users on which both models do well (e.g. *c24ur*) while other users are challenging for both models (*7swyk*, *iz2ps*).

5 CONCLUSION AND FUTURE WORK

We used a variety of feature engineering techniques combined with traditional machine learning algorithms to infer from wrist-band data whether a user is experiencing high cognitive load (CL). We obtained our best results, i.e. an accuracy of ∼ 63%, with a Logistic Regression model trained on 4 categories of physiological data, namely galvanic skin response, heart rate, rr intervals, and skin temperature data. While ∼ 63% is substantially better than random guessing, it may not be sufficiently accurate for deployment.

There are several possibilities for improvement. First of all, collecting a larger training dataset would lead to higher accuracy of the models trained in this paper, and it would also enable the use of "data hungry" deep learning architectures. Second, the set-up of the UbiTtention2020 challenge "forced" us to treat the CL inference problem for each user as a "cold start" problem (zero-shot learning), because there are no labeled training instances given at all for the 5 users in the test data. To mimic this set-up as closely

Table 2: Accuracy results of proposed approaches (6-fold CV based on users)

Approach	Features	LR	DT	RF	AB	SVM	Ensemble
1	RAW	50.90%	54.22%	56.36%	53.86%	55.12%	54.91%
2	FFT+PUF	62.42%	55.82%	58.28%	58.72%	61.80%	60.84%
	SMF+HRV	62.26%	54.15%	58.08%	56.47%	60.63%	60.29%
	FFT+PUF+SMF+HRV	59.47%	52.63%	59.30%	57.91%	62.32%	61.47%
	RAW+FFT+PUF+SMF+HRV	60.16%	56.00%	57.41%	56.24%	61.65%	62.29%
3	FS(RAW)	52.50%	53.24%	53.99%	52.63%	52.47%	54.38%
4	FS(RAW)+FFT+PUF	62.65%	**57.92%**	**59.83%**	**58.74%**	62.08%	61.80%
	FS(RAW)+SMF+HRV	**63.30%**	53.33%	58.81%	55.93%	**62.54%**	62.34%
	FS(RAW)+FFT+PUF+SMF+HRV	61.82%	57.79%	57.89%	57.82%	62.26%	**62.93%**
5	FS(RAW+FFT+PUF)	60.27%	52.15%	58.96%	56.66%	60.25%	59.51%
	FS(RAW+SMF+HRV)	59.70%	54.91%	59.54%	55.21%	59.71%	59.18%
	FS(RAW+FFT+PUF+SMF+HRV)	60.26%	56.25%	58.47%	56.13%	59.95%	58.83%

Table 3: Per user results for SVM with FFT+PUF+SMF+HRV and for LR with FS(RAW)+SMF+HRV

fold	id	SVM with FFT+PUF+SMF+HRV					LR with FS(RAW)+SMF+HRV				
		TP	TN	FP	FN	misclassification rate	TP	TN	FP	FN	misclassification rate
1	5gpsc	16	14	6	5	26.83%	18	12	8	3	26.83%
	wjxci	5	17	2	14	42.11%	8	16	3	11	36.84%
	rc1in	13	11	8	6	36.84%	10	9	10	9	50.00%
2	ibvx8	12	9	8	6	40.00%	12	8	9	6	42.86%
	yljm5	12	10	10	7	43.59%	13	16	4	6	25.64%
	hpbxa	1	6	0	7	50.00%	2	6	0	6	42.86%
3	**7swyk**	1	16	1	18	**52.78%**	2	16	1	17	**50.00%**
	e4gay	15	14	5	4	23.68%	12	15	4	7	28.95%
	ef5rq	8	14	6	10	42.11%	11	12	8	7	39.47%
4	8a1ep	5	8	2	4	31.58%	5	7	3	4	36.84%
	f3j25	6	16	4	13	43.59%	9	13	7	10	43.59%
	tn4vl	11	11	6	7	37.14%	8	12	5	10	42.86%
5	2nxs5	17	11	9	2	28.21%	18	10	10	1	28.21%
	1mpau	14	12	7	5	31.58%	10	16	3	9	31.58%
	iz2ps	5	9	10	14	**64.86%**	5	10	9	14	**60.53%**
6	b7mrd	8	11	4	8	38.71%	11	11	4	5	29.03%
	c24ur	16	16	4	3	**17.95%**	16	14	6	3	**23.08%**
	dkhty	13	12	6	6	32.43%	13	14	4	6	27.03%

as possible, we evaluated our approaches with 6-fold CV on the training data, setting the data of 3 users aside for validation in each fold. We observed higher accuracies in a 10-fold CV setup across all 632 instances in which we held out 63 instances for validation in each fold, allowing user overlap between train and validation data, indicating that having *some* labeled training instances available for a user (i.e. few-shot learning) improves inference accuracy on new, unseen instances for that same user. Finally, one can combine the inference techniques used in this paper with other information that is indicative of a user's CL, such as additional features extracted from wrist-band measurements [8, 13] or other environmental sensors.

REFERENCES

[1] Christoph Anderson, Isabel Hübener, Ann-Kathrin Seipp, Sandra Ohly, Klaus David, and Veljko Pejovic. 2018. A Survey of Attention Management Systems in Ubiquitous Computing Environments. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 2 (2018).

[2] Alejandro Baldominos, Alejandro Cervantes, Yago Sáez, and Pedro Isasi. 2019. A Comparison of Machine Learning and Deep Learning Techniques for Activity Recognition using Mobile Devices. *Sensors* 19 (2019), 521.

[3] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[4] Leo Breiman, Jerome Friedman, Charles Stone, and Richard Olshen. 1984. *Classification and Regression Trees.* Taylor & Francis.

[5] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Machine Learning* 20, 3 (1995), 273–297.

[6] Yoav Freund and Robert E Schapire. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. System Sci.* 55, 1 (1997), 119 – 139.

[7] Martin Gjoreski, Tine Kolenik, Timotej Knez, Mitja Luštrek, Matjaž Gams, Hristijan Gjoreski, and Veljko Pejović. 2020. Datasets for Cognitive Load Inference Using Wearable Sensors and Psychological Traits. *Applied Sciences* 10, 11 (May 2020), 3843.

[8] Martin Gjoreski, Mitja Luštrek, and Veljko Pejović. 2018. My Watch Says I'm Busy: Inferring Cognitive Load with Low-Cost Wearables. In *ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers (UbiComp '18).* 1234–1240.

[9] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. 2014. A survey of feature selection and feature extraction techniques in machine learning. *Proceedings of 2014 Science and Information Conference* (2014), 372–378.

[10] Tom M. Mitchell. 1997. *Machine Learning.* McGraw-Hill.

[11] Xiaoyan Mu, Jiangfeng Lu, Paul Watta, and Mohamad H. Hassoun. 2009. Weighted voting-based ensemble classifiers with application to human face recognition and voice recognition. In *2009 International Joint Conference on Neural Networks.* 2168–2171.

[12] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[13] Velijko Pejović, Martin Gjoreski, Christoph Anderson, Klaus David, and Mitja Luštrek. 2020. Toward Cognitive Load Inference for Attention Management in Ubiquitous Systems. *IEEE Pervasive Computing* 19, 2 (2020), 35–45.

[14] John Ross Quinlan. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[15] Charles Van Loan. 1992. *Computational Frameworks for the Fast Fourier Transform.* Society for Industrial and Applied Mathematics. https://epubs.siam.org/doi/abs/10.1137/1.9781611970999

[16] Pauli Virtanen, Ralf Gommers, and et al. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272.