A Feature Ranking and Selection Algorithm for Machine Learning-Based Step Counters

Stef Vandermeeren, Samuel Van de Velde, Herwig Bruneel, and Heidi Steendam, *Senior Member, IEEE*

***Abstract*— Although ultra wideband (UWB) positioning is considered as one of the most promising solutions for indoor positioning due to its high positioning accuracy, the accuracy in situations with a large number of users will be reduced because the time between two UWB position updates can be very high. To obtain a position estimate in between these updates, we can combine the UWB positioning with a different technology, e.g., an inertial measurement unit (IMU) that captures data from an accelerometer, gyroscope, and magnetometer. Previous research using the IMU outputs for location-based services employs the periodic behaviour of the accelerometer signal to count steps. However, most of these algorithms require extensive manual tuning of multiple parameters to obtain satisfactory accuracy. To overcome these practical issues, step counting algorithms using machine learning (ML) principles can be developed. In this paper, we consider accelerometer-based step counters using ML. As the performance and complexity of such algorithms depend on the features used in the training and inference phase, proper selection of the employed features is important. Therefore, in this paper, we propose a novel feature selection algorithm, where we first rank the features based on their Bhattacharyya coefficients and then systematically construct a subset of these ranked features. In this paper, we compare three ranking approaches and apply our algorithm to different ML algorithms employing an experimental set. Although the performance of the evaluated combinations slightly varies for different ML algorithms, their performance is comparable to state-of-the-art step counters, without the need to tune parameters manually.**

***Index Terms*— Step counter, feature selection, machine learning, accelerometer.**

I. Introduction

In the last decade, much research effort was devoted to accurate indoor positioning technologies to be used in various location-based services (LBS) that require a precise location estimate, e.g. navigation in shopping centres, hospitals or airports, and asset tracking. A promising technology is ultra wideband (UWB) [1]–[4], which is able to achieve an accuracy of the order of tens of centimetres. However, a problem with UWB-based positioning is that only one user at a time can range with an anchor, so that if a large number of users want to know their location, the time interval between two UWB updates can be large. To obtain an estimate of the location between these updates, other technologies can be used. A promising approach is to use the sensors that are available in mobile devices, such as the inertial measurement unit (IMU) and combine the measurements of the IMU and UWB to obtain a hybrid positioning algorithm using sensor fusion. Depending on the level of cooperation between the IMU and UWB sensors, the complexity of such a sensor fusion algorithm can be very high. To obtain an approach with acceptable complexity, the cooperation between the IMU and UWB sensors can be kept low: the UWB position measurement is used as prior information to the IMU positioning algorithm, and until a new UWB measurement is available, the IMU takes over to track the user.

Manuscript received January 12, 2018; accepted February 1, 2018. Date of publication February 16, 2018; date of current version March 22, 2018. This work was supported in part by the Flemish Fund for Scientific Research (FWO), in part by the Interuniversity Attraction Poles Program initiated by the Belgian Science Policy Office, and in part by the EOS through the Belgian Research Councils FWO and FNRS under Grant 30452698. The associate editor coordinating the review of this paper and approving it for publication was Dr. Arindam Basu. *(Corresponding author: Stef Vandermeeren.)*

The authors are with the Department of Telecommunications and Information Processing, Ghent University, 9000 Ghent, Belgium (e-mail: stef.vandermeeren@ugent.be; samuel@pozyx.io; herwig.bruneel@ugent.be; heidi.steendam@ugent.be).

Digital Object Identifier 10.1109/JSEN.2018.2807246

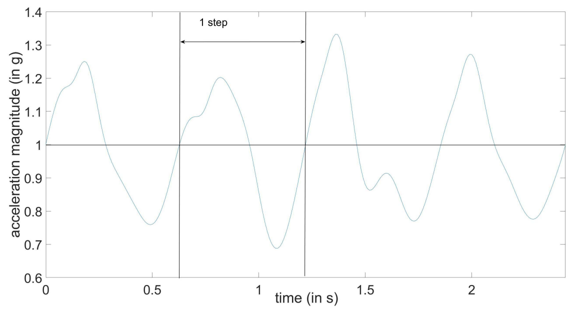


Fig. 1. Snippet of the acceleration magnitude (in g).

The IMU is able to measure the motion and orientation of a mobile device and can be used e.g. in a pedometer to count the number of steps taken by a user, or in a People Dead Reckoning application [5]–[7], which is an indoor localization technique that combines the number of detected steps with their length and orientation to track a user. In these applications, the IMU signal is evaluated to detect the number of steps taken by a walking user, which is obtained by employing the temporal variation in the acceleration vector. This temporal variation is caused by the differences in acceleration when the heel makes contact with the floor, when the leg makes a swing, or when the user stands still. The resulting acceleration signal for a walking user exhibits a periodic pattern, as illustrated in Figure 1. This figure shows a snippet[1] of the

[1]In this paper we will use the term *snippet* for a short fragment, i.e. with a length of a few seconds, of the measured acceleration.

magnitude of the acceleration vector corresponding to four steps. The repetitive nature of the signal lends itself well to extract and count steps.

In the literature, several ad hoc algorithms to count the number of steps using this repetitive nature of the acceleration can be found. Most of them operate in the time domain, by detecting the peaks or zero crossings in the measured signals, e.g. [5], [8]–[11]. However, due to noise and irregularities in the measured signals — successive steps may lead to strong differences in the measured acceleration — algorithms that only count peaks or zero crossings in general will result in large deviations between the number of detected steps and the real number of steps. To improve the performance, additional constraints were added to the algorithms, such as, only detecting peaks above an amplitude threshold, or setting temporal thresholds to avoid that noise, resulting in multiple zero crossings in a short time interval, is translated into multiple steps. A comparison between several ad hoc step counters was made in [12]. The authors observed that the median error rate, i.e. the median of the absolute differences between the real and detected number of steps, was less than 3% for all cases, and that the algorithms were more likely to undercount than overcount. Moreover, all considered algorithms required extensive tuning of different parameters, which limits the practical use of the algorithms, as the optimal parameter settings will be different for every person and situation. Because of the practical limitations of these ad hoc algorithms, other algorithms, based on machine learning, are now being considered and already resulted in a few commercially available pedometers, e.g. Fitbit and Jawbone. These step counters all adopt the supervised learning principle, where the algorithm must learn to fit the input data to an output, so that the algorithm is able to predict the output when new data is entered. To this end, the algorithm is provided with examples of input data together with the desired output during the training phase. Several supervised learning algorithms exist, but up to now, only a few have been considered for step detection and counting. In [13], a step counter was designed using neural networks taking as input data the magnitude of the acceleration signal, while in [14], a decision tree based pedometer was proposed that uses as input several features from the accelerometer and gyroscope signal. These *features* are scalar numbers extracted from the measured data, and the machine learning algorithm combines the information included in the features to determine the number of steps. The median error rates of these algorithms were -0.5% and -6.8%, respectively, indicating they slightly underestimate the number of steps. Further in [15], a support vector machine (SVM) was used to decide whether a measured accelerometer signal corresponds to one or more steps, or to a situation where the smartphone measures acceleration that is not caused by stepping, i.e. they focus on how to detect false steps, but the algorithm is not used to count the number of steps. Although some research can be found regarding machine learning step counters, no comparison between different machine-learning-based step counters is available.

The complexity of both the training phase and the inference phase of supervised learning algorithms rises with the number of used features. Therefore, a low-complexity algorithm preferably employs as few as possible features. However, mindlessly reducing the number of features can strongly affect the performance of the algorithms. Hence, selecting which features to be used is a crucial step. In the literature, the features for machine learning based step counters are selected in an ad hoc way. To the authors' best knowledge, the optimal selection of features for step counters has not been considered yet. To meet these concerns, we propose in this paper a novel algorithm to select the features. Although our algorithm is applied to the feature selection problem in step counters, the algorithm can easily be extended to select features for other applications.

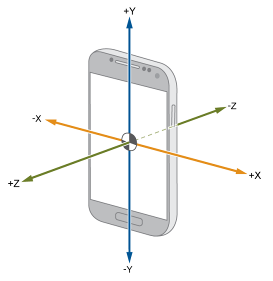


Fig. 2. Coordinate system smartphone.

The rest of the paper is organised as follows. In Section II, we define a set of features from which our method will select the best features. Then, we explain how we use the Bhattacharyya coefficients to rank the features, and in Section III, we describe the method to select the final feature set. In Section IV, we evaluate the performance of the resulting algorithms and the conclusions will be given in Section V.

II. Feature Ranking

A. Feature Extraction From the Accelerometer Signal

Common to all supervised learning algorithms is that they require as input a set of features. In this paper, we restrict our attention to data in the form of snippets, captured by the accelerometer contained in a smartphone, and for each snippet we extract a set of features. The accelerometer outputs three signals, corresponding to the acceleration in the x, y and z direction (see Figure 2). We assume that the smartphone is handheld in texting position so that the z-axis is approximately aligned with gravity, i.e. the magnitude of the acceleration in the z-direction is larger than in the x- and y-direction. We considered this orientation of the smartphone for convenience of the processing of the experimental data during the acquisition phase. However, this assumption is not critical for the proper action of the proposed feature selection algorithm as long as sufficient snippets corresponding to different smartphone orientations are included in the training phase. The signals are sampled at $100Hz$. We define the start and end points of a step as the instants at which the acceleration magnitude crosses $1g$, i.e. the gravitational force, with a positive slope as indicated in Figure 1. Due to measurement noise in accelerometers, several crossings may occur for each step. To prevent that noise will cause multiple crossings with 1g during a step,

TABLE I
PROPERTIES OF THE USED DATA SET

number of snippets	1538
total time (s)	3055
number of snippets for training	1230
number of snippets for testing	308
number of steps	4564
0 steps (%)	6.83
1 steps (%)	1.30
2 steps (%)	17.17
3 steps (%)	41.02
4 steps (%)	30.36
5 steps (%)	3.32

we first smoothen the measured acceleration sequences with a 3rd order low-pass Butterworth filter. The resulting sequence of samples

$$\boldsymbol{a}_i = (a_{x,i}, a_{y,i}, a_{z,i}),$$

where $a_{\alpha,i}$ is the i^{th} filtered sample of the α component of the acceleration, $\alpha \in \{x, y, z\}$, is subdivided in shorter fragments, i.e. snippets, in order to contain a few steps.

We select the duration of the snippets between one and three seconds, and the snippets start and end with a $1g$ magnitude crossing with positive slope. To train and test our algorithm, we collected data corresponding to approximately 51 minutes of walking. This data was converted in 1538 snippets, where for each snippet, we manually determined the number of steps contained in the snippet. This approach resulted in snippets containing zero to five steps. The set of 1538 snippets is randomly subdivided in two sets, i.e. a training set S_{train}, that is used to build a model to predict the number of steps in a snippet, and a test set S_{test}, that is used to evaluate the performance of the trained model. In Table I, the properties of the collected data set are shown.

The supervised learning algorithm predicts the output, i.e. how many steps are contained in a snippet, by combining the information contained in the features. To determine which features are most suitable for step detection, we calculated for each snippet in our experimental data set a large number (i.e. 128) of features, including but not limited to the minimum, maximum, mean, variance, and energy of all acceleration components and the acceleration magnitude. For each snippet, we arrange the features in a vector $\boldsymbol{x}_i$, where i is the snippet index, $i \in [1, |S_{train}|_C]$ with $|S_{train}|_C$ the cardinality of the training set S_{train}. In the following sections, we will describe an algorithm to select out of this set of 128 features a subset of features resulting in the best performance. For some of the supervised learning algorithms, some pre-processing is required on the features. More specifically, in algorithms that are sensitive to scaling of features, e.g. SVM, features that can take values from a larger range will be considered as more important. To avoid this problem, features are often normalised, meaning that each feature is scaled so that the values of each feature approximately have the same range. In this paper, this is achieved with

$$x'_{i,j} = \frac{x_{i,j} - \mu_{x,j}}{\sigma_{x,j}}, \tag{1}$$

where $x_{i,j}$ is the j^{th} component of the feature vector $\boldsymbol{x}_i$ of snippet i, with $j \in [1, |\boldsymbol{x}_i|_C]^2$ and $i \in [1, |S_{train}|_C]$. Further $\mu_{x,j}$ and $\sigma_{x,j}$ are the mean and standard deviation of the j^{th} feature taken over all snippets in the training data set S_{train} and $\boldsymbol{x}'_i$ is the normalised feature vector for snippet i. If new measurements are carried out, the features derived from these snippets will be normalised using the mean and standard deviation that were determined with the training set. In the remainder of this paper, we will restrict our attention to normalised features.

B. Bhattacharyya Coefficient

In a supervised-learning-based step counter, the algorithm must decide to which *class* a snippet belongs, i.e. how many steps are contained in the snippet. Because different snippets correspond to different numbers of steps in different situations, the features can be modelled as random variables. To decide which of the 128 features calculated in Section II-A are suitable for step counting, we need to compare the conditional distributions[3] of the feature, assuming the snippet contains a given number of steps, i.e. the distribution of the feature conditioned on the class. If the conditional distributions of a feature for the different classes show noticeable differences, the feature can contribute to the decision process, while a feature whose conditional distribution is essentially independent of the class, is not suitable.

A measure for the similarity between distributions is the Bhattacharyya coefficient [16]:

$$d_{Bhat,j}(n, n') = \sqrt{1 - \frac{1}{\sqrt{\overline{H}_{j,n}\overline{H}_{j,n'}}}\frac{1}{N}\sum_{i=1}^{N}\sqrt{H_{j,n}(i)H_{j,n'}(i)}}, \tag{2}$$

where n and n' correspond to the classes (or the number of steps in a snippet) of which the conditional distributions need to be compared, j corresponds to the index of the feature for which the conditional distributions are compared, $H_{j,n}(i)$ is the value of the histogram of class n in the i^{th} bin, $\overline{H}_{j,n} = \frac{1}{N}\sum_{i=1}^{N} H_{j,n}(i)$ is the average of the conditional distribution of the feature within class n, and N is the number of bins in the histograms for both classes n and n'. From (2), it is clear that if the two histograms do not overlap, the coefficient $d_{Bhat,j}(n, n')$ between classes n and n' will be equal to one, while if the histograms completely overlap, $d_{Bhat,j}(n, n')$ will be zero. Hence, we expect that the best features will be features with high Bhattacharyya coefficients between the classes. For each of the 128 features, we determined the conditional distribution of the feature within each class, and computed the Bhattacharyya coefficient (2). An example is illustrated in Table II for two features, i.e. one feature with low similarities, thus a good feature, and one with high similarities, thus a bad feature. Further for the two features of Table II, the histograms for the classes of three and four steps are

[2] In this paper $|\boldsymbol{x}_i|_C = 128$.

[3] These conditional distributions are determined based on the full experimental data set containing all 1538 snippets.

TABLE II

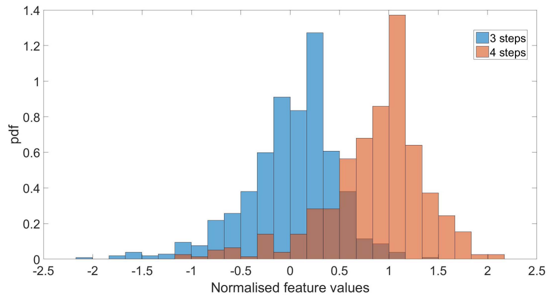
BHATTACHARYYA COEFFICIENT $d_{Bhat,j}(n,n'), n, n' \in [0,5]$ FOR TWO FEATURES (a) A GOOD FEATURE AND (b) A BAD FEATURE

class (steps)	**0**	**1**	**2**	**3**	**4**	**5**
0	0	0.83	1	1	1	1
1	0.83	0	0.89	0.95	0.99	1
2	1	0.89	0	0.73	0.90	0.93
3	1	0.95	0.73	0	0.64	0.81
4	1	0.99	0.90	0.64	0	0.56
5	1	1	0.93	0.81	0.56	0

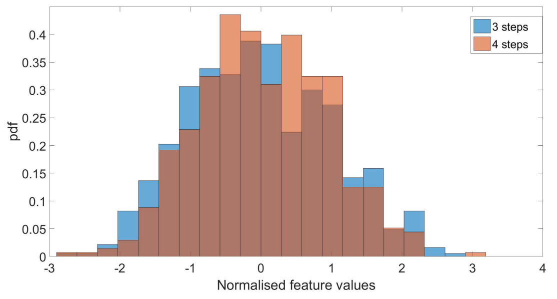
(a)

class (steps)	**0**	**1**	**2**	**3**	**4**	**5**
0	0	0.51	0.31	0.31	0.31	0.57
1	0.51	0	0.46	0.45	0.43	0.53
2	0.31	0.46	0	0.11	0.17	0.42
3	0.31	0.45	0.11	0	0.12	0.39
4	0.31	0.43	0.17	0.12	0	0.40
5	0.57	0.53	0.42	0.39	0.40	0

(b)



(a)



(b)

Fig. 3. Two normalized histograms for snippets with three and four steps: (a) shows an example of a feature that could be used to distinguish three and four steps, (b) gives an example of a feature that is not useful.

shown in Figure 3. As can be observed, the histograms of feature (b) highly overlap, resulting in $d_{Bhat,b}(3,4) = 0.12$, while for feature (a), where the distributions are clearly distinct, $d_{Bhat,a}(3,4) = 0.64$. For all other combinations of classes, $d_{Bhat}(n,n')$ for feature (b) is much lower than for feature (a). Hence, feature (b) will be less suited to discriminate the number of steps, and thus for our step counting algorithms.

C. Ranking the Features

In our investigations, we considered a large number of features, from which we want to select the best ones. However, because of this large set, optimally selecting the features in a brute force way, by evaluating all possible combinations, is NP hard. Therefore, we have to resort to suboptimal selection algorithms. The approach we will use is to first rank the features based on their Bhattacharyya coefficients and then use the ranked features as input for a systematic selection algorithm. In this section, we focus on the ranking of the features. A first hurdle to take is that for each feature, multiple Bhattacharyya coefficients are available, i.e in a system with $M+1$ classes, we have $\frac{M(M+1)}{2}$ distinct Bhattacharyya coefficients to consider. Hence, ranking the features with respect to these coefficients is not straightforward. To meet this problem, we first reduce the number of Bhattacharyya coefficients per feature, by assuming that if we are able to discriminate between the classes with n and $n+1$ steps, $n \in [0, M-1]$, we can also discriminate between classes with higher step difference. The validity of this assumption is illustrated in the example of Table II: the smallest values of the Bhattacharyya coefficient, corresponding to low discrimination ability between the classes, generally occur when there is only one step difference. Therefore, in the remainder of this paper, we restrict our attention to the evaluation of the Bhattacharyya coefficients between classes with n and $n+1$ steps, i.e. $d_{Bhat,j}(n,n+1), n \in [0, M-1]$. A second issue is the large number of features to be tested. In order to reduce the complexity of feature selection, we first note that features with high Bhattacharyya coefficients are more likely to give good results in our step counting algorithms. Therefore, before we start selecting features, we rank the features using one of the following three ranking approaches, and limit the number of features to be further processed.

- **Approach 1:** In this first approach, for each value of n, we first rank the features in descending order of their Bhattacharyya coefficients $d_{Bhat,j}(n,n+1)$.

$$\boldsymbol{p}_n = \underset{j \in 1:|\boldsymbol{x}_i|_C}{argsort}(d_{Bhat,j}(n,n+1)), \tag{3}$$

where $|\boldsymbol{x}_i|_C$ is the number of features and $argsort(u_j)$ is the function that returns the indices of the vector $\boldsymbol{u}$ that would sort the array, i.e. the vector $\boldsymbol{p}_n$ contains the indices of the features with the Bhattacharyya coefficients $d_{Bhat,j}(n,n+1)$ sorted in descending order:

$$d_{Bhat,p_n(l)}(n,n+1) \geq d_{Bhat,p_n(l+1)}(n,n+1), \tag{4}$$

for $l \in [1, |\boldsymbol{x}_i|_C - 1]$. Hence, we obtain M vectors $\boldsymbol{p}_n$, $n \in [0, M-1]$. In the next step, we limit the number of elements in $\boldsymbol{p}_n$ to K, i.e. $\boldsymbol{p}_n^{[K]} = \boldsymbol{p}_n(1:K)$: for each n, only the K features with the highest Bhattacharyya coefficients, i.e. the highest ranked features, are kept for further processing. We arrange the vectors $\boldsymbol{p}_n^{[K]}$ in a $M \times K$ matrix:

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{p}_0^{[K]} \\ \vdots \\ \boldsymbol{p}_{M-1}^{[K]} \end{bmatrix} \tag{5}$$

and convert this matrix $\boldsymbol{P}$ into a vector $\boldsymbol{v}$ by reading out the matrix column-wise, i.e. $\boldsymbol{v} = \boldsymbol{P}(:)$, or

$$\boldsymbol{v} = \big[p_0(1) \cdots p_{M-1}(1), \\ p_{0,2} \cdots p_{M-1}(2) \cdots p_0(K) \cdots p_{M-1}(K)\big]. \quad (6)$$

If a feature occurs in more than one vector $\boldsymbol{p}_n^{[K]}$, because it is suitable to distinguish between more than two classes, its index occurs more than once in the vector $\boldsymbol{v}$. We only keep the first occurrence, and remove the duplicates from $\boldsymbol{v}$ to obtain the final ranked feature set $\boldsymbol{\Omega}_{r,1}$. This set $\boldsymbol{\Omega}_{r,1}$ contains maximally $K' = M \cdot K$ different features.

- **Approach 2:** Similarly as in Approach 1, we sort for each n the features in descending order of their Bhattacharyya coefficients $d_{Bhat,j}(n, n+1)$ to obtain the vectors $\boldsymbol{p}_n$ (3). Then, we define for each n a score vector $\boldsymbol{s}_n$ that contains for each feature j its position in the ranked list $\boldsymbol{p}_n$:

$$s_{n,j} = l | [p_n(l) = j], \quad (7)$$

i.e. the feature j with the highest $d_{Bhat,j}(n, n+1)$ gets the score $'1'$, while for the feature with the lowest $d_{Bhat,j}(n, n+1)$, the score is $'|\boldsymbol{x}_i|_C'$. In the next step, we average for each feature the score over the different n:

$$\overline{s}_j = \frac{1}{M} \sum_{n=0}^{M-1} s_{n,j} \quad (8)$$

and sort the scores in ascending order to obtain the vector $\boldsymbol{p}'$ of feature indices

$$\boldsymbol{p}' = \underset{j=1:|\boldsymbol{x}_i|_C}{argsort}(\overline{s}_j). \quad (9)$$

Finally, we limit the number of features to be further processed to K' and obtain the final ranked feature set $\boldsymbol{\Omega}_{r,2}$ of the K' highest ranked features as

$$\boldsymbol{\Omega}_{r,2} = \boldsymbol{p}'(1 : K'). \quad (10)$$

- **Approach 3:** In this last approach, we first compute for each feature the average Bhattacharyya coefficient $\overline{d}_{Bhat,j}$ over the different n:

$$\overline{d}_{Bhat,j} = \frac{1}{M} \sum_{n=0}^{M-1} d_{Bhat,j}(n, n+1). \quad (11)$$

Then, we sort the features in descending order of average Bhattacharyya coefficient:

$$\boldsymbol{p}'' = \underset{j=1:|\boldsymbol{x}_i|_C}{argsort}(\overline{d}_{Bhat,j}) \quad (12)$$

and finally, we limit the number of features to be further processed by K', resulting in the final ranked feature set $\boldsymbol{\Omega}_{r,3}$ with the highest ranked features:

$$\boldsymbol{\Omega}_{r,3} = \boldsymbol{p}''(1 : K'). \quad (13)$$

The three approaches all have advantages and drawbacks. For example, a feature that is reasonably good in distinguishing multiple classes, but is not ranked in the top ten in one of the vectors $\boldsymbol{p}_n$, will not be considered in Approach 1, while it may be included in the other two approaches because its average score is good. On the other hand, Approach 1 will contain a feature that is extremely good in discriminating between one set of classes, but is weak in distinguishing between other classes, while the two other approaches might neglect this feature as its average score is bad. In Section IV, we will compare the obtained performance for the three ranking approaches.

III. Feature Selection

A. The Algorithm

In the previous section, we introduced three approaches to rank the features, each resulting in a ranked set $\boldsymbol{\Omega}_{r,i}$, $i = \{1, 2, 3\}$, of at most K' features. In this section, we describe an algorithm that starts from one of the ranked sets $\boldsymbol{\Omega}_{r,i}$ to select a subset $\boldsymbol{\Omega}_{f,i} \subset \boldsymbol{\Omega}_{r,i}$ of features that results in optimal performance with a limited number of features. As this selection algorithm can be used irrespective of the used ranking approach, we drop the index i in this section to simplify notations.

In this feature selection algorithm, we will sequentially update the final set $\boldsymbol{\Omega}_f$ of features. To evaluate the performance of the selected subset $\boldsymbol{\Omega}_f$, we randomly select 80% of the experimental data set (Table I), i.e. $|S_{train}|_C = 1230$ snippets, to train the supervised algorithm, while the other $|S_{test}|_C = 308$ snippets are used to evaluate the performance of the selected feature set $\boldsymbol{\Omega}_f$. To minimize the influence of the selected training set, we repeat the construction of $\boldsymbol{\Omega}_f$ 100 times using different randomly selected training sets. The overall performance, i.e. the snippet accuracy $P_{s,f}$, which equals the number of snippets with the correct number of estimated steps divided by the total number of snippets, is found by averaging the performance over the 100 runs. By definition $P_{s,f} \leq 1$ or expressed in percentage $P_{s,f} \leq 100\%$. The closer $P_{s,f}$ gets to 100%, the higher the accuracy of the step counting algorithm.

The algorithm to update the final feature set $\boldsymbol{\Omega}_f$ consists of three phases, i.e. the initialisation phase, the addition phase and the deletion phase.

1) **Initialisation phase:** First, the algorithm computes the features and their Bhattacharyya coefficients to be able to rank them with one of the approaches of Section II-C, resulting in the ranked set $\boldsymbol{\Omega}_r$. Then, it creates the final feature set $\boldsymbol{\Omega}_f$, which in this initial stage contains the first ranked feature only, i.e. the feature with the best score of the ranked set $\boldsymbol{\Omega}_r$, and it removes this feature from $\boldsymbol{\Omega}_r$. Employing the set $\boldsymbol{\Omega}_f$, we train the supervised learning algorithm with S_{train}, and evaluate the accuracy of the trained model with S_{test}.
2) **Addition phase:** In the next phase, new features from the ranked feature set $\boldsymbol{\Omega}_r$ are tested sequentially, and added to $\boldsymbol{\Omega}_f$ if they improve the snippet accuracy significantly, i.e. if $P_{s,temp} > P_{s,f} + threshold$, where $P_{s,temp}$ is the snippet accuracy obtained after training the model with the temporary feature set

$$\boldsymbol{\Omega}_{temp} = \boldsymbol{\Omega}_f \cup \{x_{new}\},$$

and x_{new} the new feature that is evaluated. We distinguish two cases for x_{new}: **1)** The temporary feature x_{new}

did result in $P_{s,temp} > P_{s,f} + threshold$. The temporary feature is retained in the set $\mathbf{\Omega}_f$, i.e.

$$P_{s,f} \leftarrow P_{s,temp} \quad (14)$$
$$\mathbf{\Omega}_f \leftarrow \mathbf{\Omega}_{temp} \quad (15)$$
$$\mathbf{\Omega}_r \leftarrow \mathbf{\Omega}_r \setminus \{x_{new}\}. \quad (16)$$

Next, we restart testing the feature with the highest rank in $\mathbf{\Omega}_r$ that was not added to $\mathbf{\Omega}_f$ in a previous stage. This look-back phase is introduced because a feature that was not selected in a previous phase, as e.g. it is highly correlated with one feature or a combination of features that is already present in $\mathbf{\Omega}_f$ and therefore did not have noticeable influence on the performance, might have added value in combination with a feature that has a lower ranking. In our tests, on average 14.8% of the features that were added to the final set, were not added the first time they were considered in the addition phase. **2)** The previously tested feature was not included in $\mathbf{\Omega}_f$: the algorithm proceeds with the next feature in $\mathbf{\Omega}_r$ with a lower rank than the previously tested feature, i.e. the look-forward phase. Every time the algorithm adds a feature x_{new} to $\mathbf{\Omega}_f$, it removes the feature from $\mathbf{\Omega}_r$.

3) **Deletion phase:** During the addition phase, all features in $\mathbf{\Omega}_r$ were tested and added to $\mathbf{\Omega}_f$ if they resulted in an improved $P_{s,f}$. In the deletion phase, we test if the removal of a feature in $\mathbf{\Omega}_f$ degrades $P_{s,f}$, to check if we can reduce the number of features in the final set without affecting the performance. To this end, we start by setting

$$\mathbf{\Omega}_{temp} = \mathbf{\Omega}_f \setminus \{x_{del}\},$$

where x_{del} is the first feature that was added to the final set, and determine the snippet accuracy $P_{s,temp}$ using $\mathbf{\Omega}_{temp}$ to train the machine learning algorithm.
Again we can distinguish two cases. **1)** $P_{s,temp} < P_{s,f}$, which means that the removal of x_{del} degraded the performance. Hence, we must keep this feature, i.e. $\mathbf{\Omega}_f$ does not change and we set x_{del} equal to the next feature in $\mathbf{\Omega}_f$ (look-forward phase). **2)** $P_{s,temp} \geq P_{s,f}$, implying x_{del} can safely be removed without affecting the performance. In this case:

$$P_{s,f} \leftarrow P_{s,temp} \quad (17)$$
$$\mathbf{\Omega}_f \leftarrow \mathbf{\Omega}_{temp} \quad (18)$$
$$\mathbf{\Omega}_r \leftarrow \mathbf{\Omega}_r \cup \{x_{new}\} \quad (19)$$

i.e. we remove x_{del} from $\mathbf{\Omega}_f$, add x_{del} to $\mathbf{\Omega}_r$, and set x_{del} equal to the feature of $\mathbf{\Omega}_f$ that was added the longest ago (look-back phase). In our tests, on average 21.0% of the features that were selected in the addition phase, were removed in the deletion phase. The reason why this feature can be deleted is because a combination of other features that were added to $\mathbf{\Omega}_f$ in the addition phase contain similar information so that this feature is obsolete. In contrast to the addition phase, we do not use a threshold in the deletion phase. This procedure is continued until the last feature in $\mathbf{\Omega}_f$ is evaluated and not removed from $\mathbf{\Omega}_f$.

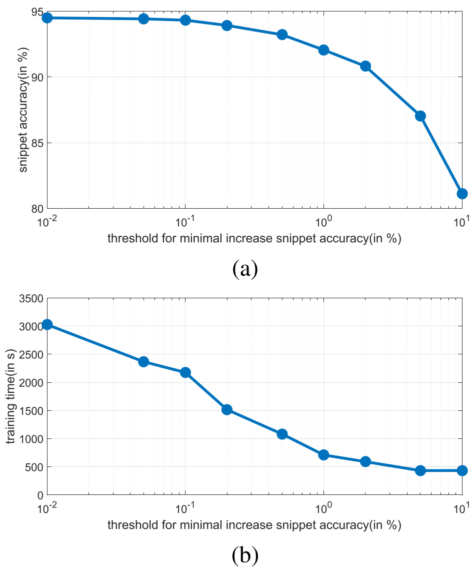


Fig. 4. **a)** Snippet accuracy (in %) and **b)** training time (in seconds) as function of the threshold in the addition phase.

If none of the features from $\mathbf{\Omega}_f$ was removed, we stop the selection algorithm. In the other case, we go back to the addition phase followed by a deletion phase. This approach yields optimal performance in the sense that adding another feature from $\mathbf{\Omega}_r$ or removing a feature from $\mathbf{\Omega}_f$ will not improve the accuracy. We noticed that in most cases, features were removed from the final set $\mathbf{\Omega}_f$ in the first deletion phase only and that on average only 2.1% of the features were added in the second addition phase or later. This means that typically the addition and deletion phase need to be executed only twice. After the final feature set is determined using our feature selection algorithm, this feature set Ω_f will be used to train a model from the entire experimental data set $S_{train} \cup S_{test}$ with the machine learning algorithm that was used to find these features. This model can then be used to estimate the number of steps in snippets derived from new data in the inference phase.

In the addition phase we introduced a threshold. As the performance of the set Ω_f is determined using the test sets S_{test} extracted from our experimental data set, allowing the addition of features with limited impact on the performance might result in a final feature set that is overtrained on our experimental data set. Although such an overtrained feature set performs well for the considered data set, it probably will result in a degraded performance when new data needs to be evaluated in the inference phase. Besides preventing overtraining, with this threshold, we can also reduce the training time. In Figure 4, we show the snippet accuracy and training time as a function of this threshold. These results were obtained by averaging the snippet accuracy and training time of our feature selection algorithm over four different machine learning algorithms (i.e. Decision Tree, Linear SVM, RBF SVM and the Quadratic Discriminant Analysis algorithm) for several values of this threshold. As can be observed, for a

TABLE III

FEATURES IN $\boldsymbol{\Omega}_{f,1} \cup \boldsymbol{\Omega}_{f,2} \cup \boldsymbol{\Omega}_{f,3}$, WHERE $\boldsymbol{\Omega}_{f,i}$ IS THE FINAL FEATURE SET FOR RANKING APPROACH i, WHEN THE FEATURE SELECTION ALGORITHM IS USED ON THE RBF SVM ALGORITHM

Feature number	***Feature description***	***Feature symbol***
1	Standard deviation of a_z	σ_{a_z}
2	Mean absolute deviation of a_z	mad_{a_z}
3	Number of samples of $\|\mathbf{a}\|$ above threshold	$th_{\|\boldsymbol{a}\|}$
4	Number of maxima in $\|\mathbf{a}\|$	$n_{max,\|\boldsymbol{a}\|}$
5	Mean of maxima in $\|\mathbf{a}\|$	$\mu_{max,\|\boldsymbol{a}\|}$
6	Standard deviation of maxima(peaks) $\|\mathbf{a}\|$	$\sigma_{max_{\|\boldsymbol{a}\|}}$
7	Frequency at maximum of FFT of a_y	f_{a_y}
8	Frequency at maximum of FFT of a_z	f_{a_z}
9	Maximum of FFT of a_z	max_{f,a_z}
10	Standard deviation of FFT of a_z	σ_{f,a_z}
11	Kurtosis of FFT of a_z	kur_{f,a_z}
12	Skewness of FFT of a_z	$skew_{f,a_z}$
13	Energy of FFT of a_z between 1 and 5Hz	E^{1-5}_{f,a_z}
14	Energy of FFT of a_z	E_{f,a_z}
15	Energy of FFT of $\|\mathbf{a}\|$	$E_{f,\|\boldsymbol{a}\|}$
16	Signal magnitude area of FFT of acceleration	SMA_f

threshold of 0.2%, the training time is halved, while the snippet accuracy only slightly decreases with approximately 0.5%. Larger thresholds only yield limited gain in training time at the cost of larger performance degradation. As a trade-off between accuracy and training time, we set in this paper the threshold to 0.2%.

B. Discussion

To illustrate the feature selection algorithm from Section III-A, we apply it to our step counting problem. In this example, we select $K = 10, K' = 50, M = 5$ and unless stated otherwise, we use a radial basis function support vector machine (RBF SVM) with classification as learning method. Table III gives an overview of the features that were selected for the final feature set, for one or more of the ranking approaches of section II-C. Features $1-6$ are extracted from the time-domain signal, while features $7-16$ are obtained from the frequency domain signal. Notice that almost all features in this table are derived from the z-component a_z of the acceleration, or the magnitude $|\boldsymbol{a}|$. This can be explained by the fact that the smartphone was approximately horizontally handheld during the measurements, so that the up-and-down movement due to a step is mainly present in the z-component. The signal magnitude area (SMA) of the FFT of the acceleration magnitude is defined as:

$$SMA_f = \frac{\sum_{j=1}^{F} |a_{f,x}(j)| + |a_{f,y}(j)| + |a_{f,z}(j)|}{F},$$

where $a_{f,\alpha}(j), \alpha \in \{x, y, z\}$, is the j^{th} sample of the Fourier transform of the α component of the acceleration, and F is the number of samples in the Fourier transform of the acceleration components. Feature 3, i.e. $th_{|\boldsymbol{a}|}$, corresponds to the number of samples in the snippet where the amplitude $|\boldsymbol{a}|$ is above a threshold. To determine the optimal range of this threshold, we apply our feature selection algorithm to the machine learning algorithms from Table IV, and for each ranking approach, we average the resulting snippet accuracy over the different machine learning algorithms. In Figure 5, we show the resulting average snippet accuracy for the three ranking approaches as function of the threshold of $th_{|\boldsymbol{a}|}$. Further, we show the snippet accuracy averaged over the three ranking approaches. Although the snippet accuracy slightly

TABLE IV

SUPERVISED LEARNING ALGORITHMS

Linear SVM classification
RBF SVM classification
RBF SVM regression
Decision Tree
Random Forest
Neural Networks
k-Nearest Neighbours
Adaboost
Quadratic Discriminant Analysis

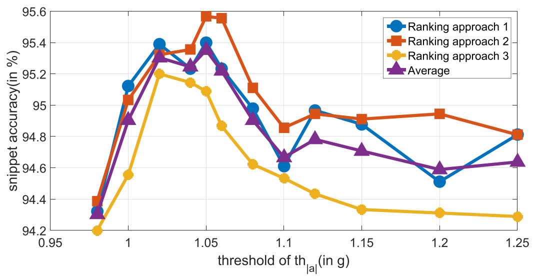


Fig. 5. Snippet accuracy averaged over different machine learning algorithms, for the three ranking approaches and the average snippet accuracy over the three ranking approaches as function of the threshold of $th_{|\boldsymbol{a}|}$.

TABLE V

FINAL FEATURE SET $\mathbf{\Omega}_{f,i}$ FOR THE THREE APPROACHES WHEN THE FEATURE SELECTION ALGORITHM IS USED ON THE RBF SVM ALGORITHM

$\Omega_{f,1}$	2	3	4	6	8	13	14	
$\Omega_{f,2}$	1	2	3	7	8	10	15	16
$\Omega_{f,3}$	3	4	5	8	9	11	12	16

TABLE VI

APPROACH 1: FEATURES FROM $\mathbf{\Omega}_{f,1}$ FOR THE RBF-SVM ALGORITHM THAT BELONG TO THE TEN BEST FEATURES TO DISTINGUISH BETWEEN n AND $n+1$ STEPS (BHATTACHARYYA COEFFICIENT IS MENTIONED BETWEEN BRACKETS)

0↔1	3(1)	6(1)	13(1)	14(1)
1↔2	13(0.97)	2(0.96)	14(0.96)	
2↔3	3(0.80)	4(0.61)		
3↔4	3(0.76)	4(0.64)	8(0.39)	
4↔5	8(0.91)	4(0.77)	3(0.60)	

differs for the three ranking approaches, it is clear that the optimal range of the threshold is between $1.02g$ and $1.07g$. This is in line with the results from [12], where an optimal threshold of $1.07g$ was found. In Figure 5, we observe that a threshold of $1.05g$ results in the highest snippet accuracy. Hence, in the remainder of the paper, we use this value for the threshold in $th_{|\boldsymbol{a}|}$.

The list of features, given in Table III, contains the union $\mathbf{\Omega}_{f,1} \cup \mathbf{\Omega}_{f,2} \cup \mathbf{\Omega}_{f,3}$ of the final feature sets $\mathbf{\Omega}_{f,i}$, where $\mathbf{\Omega}_{f,i}$ is obtained by using as input for the feature selection algorithm the ranked feature set $\mathbf{\Omega}_{r,i}$ obtained with ranking approach i, for the RBF SVM classification algorithm. Let us now compare the different $\mathbf{\Omega}_{f,i}$. The selected features for each ranking approach are listed in Table V. As can be observed, the number of features in the three final sets $\mathbf{\Omega}_{f,i}$, $i = \{1, 2, 3\}$, is similar. The table reveals there is only a limited overlap between the three sets: only two features, i.e. features 3 and 8, appear in all three sets, while three appear in two sets, but the majority of the features are selected for only one of the final sets. Although 12 out of the 16 features listed in Table III occur in the three ranked sets $\mathbf{\Omega}_{r,i}$, $i = \{1, 2, 3\}$, our selection algorithm picks different features. This can be explained by noting that in the list of features from Table III, some features are closely related, e.g. features 14 and 15. It can be observed that feature 14 was selected for Approach 1, while Approach 2 preferred feature 15. Hence, the selection algorithm is able to recognize similar features and picks one feature out of the subset of similar features, while the others are not selected unless they introduce a performance gain. Which feature is selected, depends on its ranking and thus on the used ranking approach. Therefore, although the overlap between the final feature sets $\mathbf{\Omega}_{f,i}$ seems limited, we expect that the effect on the performance will be small as several of the features are more or less interchangeable.

To obtain a high accuracy, the features that are selected for the final feature set must be able to discriminate between the different classes. Hence, we take a closer look at the features selected for Approach 1, and look which feature is helpful in the decision process between two classes. Table VI gives an overview of the features from the final set that belong to the top ten features for different n, i.e. that belong to $\boldsymbol{p}_n^{[K]}$, together with their Bhattacharyya coefficients. As can be observed, features 3 and 4 are not only able to discriminate between two and three steps, but also between three and four, and four and five steps, while these features are less helpful to distinguish between no steps and one step, and one step and two steps. The table suggests that it will be easier to discriminate between no steps and one step, and one step and two steps, as the features in these rows have very high Bhattacharyya coefficients, while the coefficients are much lower for the other three rows. However, a high Bhattacharyya coefficient does not always guarantee a high accuracy. This is illustrated in Tables VII and VIII, where the confusion matrices are shown. These confusion matrices show how the errors are distributed over the classes, for Approaches 1 and 2 using RBF SVM with classification, respectively. For example, for Approach 1, our algorithm predicted for 99.9% of the snippets with zero steps that the snippet had zero steps, while for 0.1% of the snippets with zero steps, the algorithm predicted one step. Although we expected that it would be easier to distinguish between zero and one, and one and two steps because of the high Bhattacharyya coefficients of the features, we notice from the confusion matrices of both approaches that the snippets with one step are most often assigned the wrong number of steps, i.e. the accuracy for class 1 is 79.6% for Approach 1 and only 77.9% for Approach 2. Yet we find that the overall snippet accuracy $P_{s,f}$ for both approaches is high: $P_{s,f} = 97.2\%$ and $P_{s,f} = 97.7\%$ for Approach 1 and 2 respectively. A reason for this behaviour is that in our data set only 1.30% of the snippets contained one step (this only happens at the beginning or end of a walk), so that the supervised learning algorithm had too few snippets to train on. We expect that when more snippets with one step are used

TABLE VII

CONFUSION MATRIX FOR APPROACH 1 USING RBF SVM

Actual \ **Predicted**	**0**	**1**	**2**	**3**	**4**	**5**
0	99.9	0.1	0	0	0	0
1	0	79.6	18.8	1.6	0	0
2	0	1.3	92.0	6.6	0.1	0
3	0	0	1.0	97.9	1.1	0
4	0	0	0.1	0.9	98.5	0.5
5	0	0	0	1.9	4.0	94.1

TABLE VIII

CONFUSION MATRIX FOR APPROACH 2 USING RBF SVM

Actual \ **Predicted**	**0**	**1**	**2**	**3**	**4**	**5**
0	99.8	0.2	0	0	0	0
1	10.5	77.9	10.4	1.2	0	0
2	0	0.7	93.8	5.1	0.4	0
3	0	0.1	1.1	97.6	1.2	0
4	0	0	0.2	0.9	98.8	0.1
5	0	0	0	1.9	3.6	94.5

TABLE IX
ACCURACY COMPARISON OF DIFFERENT MACHINE LEARNING ALGORITHMS

	Benchmark		Approach 1		Approach 2		Approach 3	
$Algorithm$	$Snippet\ accuracy(\%)$	$Step\ error(\%)$	$Snippet\ accuracy(\%)$	$Step\ error(\%)$	$Snippet\ accuracy(\%)$	$Step\ error(\%)$	$Snippet\ accuracy(\%)$	$Step\ error(\%)$
Linear SVM classification	94.7	-0.12	92.6	-0.34	92.9	-0.36	91.6	-0.45
RBF SVM classification	93.1	-0.51	96.8	0.30	97.1	0.11	96.9	0.13
RBF SVM regression	93.7	-0.26	97.2	0.17	97.7	0.31	97.2	0.37
Decision Tree	90.3	0.01	93.0	-0.01	92.9	-0.15	92.5	-0.08
Random Forest	93.3	-0.61	95.6	-0.06	95.7	-0.05	95.6	-0.01
Neural Networks	94.4	-0.06	95.4	0.10	95.8	0.12	94.8	0.14
k-Nearest Neighbours	79.5	1.31	94.6	0.82	94.6	0.57	94.1	0.40
Adaboost	96.6	-0.22	96.8	-0.14	97.0	-0.05	96.9	-0.16
Quadratic Discriminant Analysis	78.0	3.8	96.7	0.33	96.2	0.59	96.2	0.25

in the training phase, this accuracy will improve. However, although the accuracy for one step is quite low, the overall snippet accuracy $P_{s,f}$ for both approaches is still high because so few snippets with one step are present.

IV. RESULTS

In this section, we compare the performance of machine-learning-based step counters employing the proposed systematic feature selection algorithm from Section III-A. We apply the selection algorithm to the nine supervised-learning algorithms listed in Table IV. All but one of these algorithms are classification techniques, i.e. the output of the algorithm is an integer value (the number of steps), and one algorithm is a regression technique. Regression algorithms output a real number, which we round to the nearest integer to obtain the prediction of the number of steps. For each of the considered machine learning algorithms, we rank the features using the three approaches from Section II-C, and determine the final feature sets $\boldsymbol{\Omega}_f$ using the selection algorithm from Section III-A. First, we define the performance metrics that are used to compare the different step counting algorithms. Next, we investigate which combination of ranking approach and machine learning algorithm results in the step counter with the highest snippet accuracy. Finally, we also compare our best algorithm with state-of-the-art step counters.

In this paper, we characterise the performance of our algorithm with two metrics. The first one is the snippet accuracy $P_{s,f}$, as employed in the feature selection algorithm in Section III-A; this is the ratio of the number of snippets for which the predicted number of steps was correct, and the total number of snippets. The second performance metric is the step error, which is defined as the ratio of the difference between the estimated and the correct total number of steps summed over all snippets, and the correct total number of steps in all snippets. Although the two metrics are correlated, i.e. a high snippet accuracy will probably result in a low step error, situations may occur where a small step error corresponds to a lower snippet accuracy. This will happen when in one snippet, the number of steps is overestimated, while in another snippet it is underestimated. Although the snippet accuracy in such a case can be rather low, the step error will be small because the errors cancel out. In Table IX, the snippet accuracy and step error are shown for the different supervised learning algorithms combined with the feature selection algorithm from Section III-A using the three ranking approaches from Section II-C, as well as for a benchmark. As the benchmark, we consider for each supervised learning algorithm the case where all 128 features were used to classify the snippets. A negative step error in this table indicates the number of steps was undercounted, while a positive step error indicates it was overcounted. From Table IX, it follows that Approach 1 in combination with the RBF SVM regression algorithm results in the highest snippet accuracy, i.e. $P_{s,f} = 97.7\%$, and that Approach 3 combined with the linear SVM algorithm resulted in the lowest performance, i.e. $P_{s,f} = 91.6\%$. If we compare the three approaches with the benchmark, we can see that for the majority of the cases, our algorithm resulted in a better performance. Moreover, we notice that all combinations of machine learning algorithms and ranking approaches result in a low step error, implying that all considered combinations are suitable for applications where only the total number of steps is important. To find the best ranking approach, we average the snippet accuracy over the different machine learning algorithms. We obtain that Approach 2 has the best average performance, followed by Approach 1 and then Approach 3.

Finally in Table X we compare the RBF SVM regression algorithm trained with the features selected using the 2^{nd} ranking approach, with current state-of-the-art algorithms, i.e. the peak detection (PD) algorithm from [8], and the finite state machine (FSM) algorithm from [9]. To simplify the comparison between the three algorithms, we gathered a new validation set with in total 1000 steps. We trained the RBF SVM regression algorithm with the complete first data set of 1538 snippets (Table I), and evaluated the performance using the new validation set. For the other two algorithms, in a first stage, we used the parameter values mentioned in the respective papers. This approach resulted in 873 steps detected with the PD algorithm, 916 steps detected with the FSM algorithm, and 986 steps with our machine learning method (ML). The step errors are mentioned in the first row of Table X. We clearly see that our algorithm outperforms the other methods. The lower performance for the PD and FSM method can be attributed to the used parameter setup: we used the parameters from [8], [9], which were tuned to their test persons. To improve the performance of the PD and

TABLE X
COMPARISON MACHINE LEARNING ALGORITHM WITH AD HOC ALGORITHMS

Step error	PD [8]	FSM [9]	ML
Tuning from [8], [9]	-12.7%	-8.4%	-1.4%
Our tuning	-0.6%	0.2%	

FSM method, we tune the parameters using the validation set. Compared to the parameter settings of [8] and [9], our tuning noticeably improves the performance, as can be observed in the last row of Table X. We can conclude that the performance of the algorithm considered in this paper is similar to that of state-of-the-art algorithms, although the benefit of the method proposed in this paper is the absence of parameters to be tuned manually.

V. CONCLUSIONS

In this paper, we proposed a method to systematically build a feature set for a machine-learning-based step counter. We used the Bhattacharyya coefficient to identify the features that could potentially be useful for step counting, and proposed three ranking approaches for the features as well as a selection method, which uses these rankings, to determine the features to be used by the step counter. To illustrate the potential of the proposed algorithm, we compared different machine learning algorithms using the selected features in terms of accuracy. The accuracies obtained with the different combinations of machine learning algorithm and ranking approach are promising: compared to state-of-the-art algorithms that were tuned to our evaluation set, our algorithm achieved similar performance. We also showed that the performance of these state-of-the-art algorithms severely degrades when not properly tuned, while the machine learning algorithms considered in this paper can easily cope with changing situations, e.g. different persons, different carrying positions and different activities, by adding additional data to the training set.

REFERENCES

[1] H. Liu, H. Darabi, P. Banerjee, and J. Liu, "Survey of wireless indoor positioning techniques and systems," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 37, no. 6, pp. 1067–1080, Nov. 2007.

[2] Y. Gu, A. Lo, and I. Niemegeers, "A survey of indoor positioning systems for wireless personal networks," *IEEE Commun. Surveys Tuts.*, vol. 11, no. 1, pp. 13–32, 1st Quart., 2009.

[3] M. Kok, J. D. Hol, and T. B. Schön, "Indoor positioning using ultrawideband and inertial measurements," *IEEE Trans. Veh. Technol.*, vol. 64, no. 4, pp. 1293–1303, Apr. 2015.

[4] G. Du, P. Zhang, and D. Li, "Human–manipulator interface based on multisensory process via Kalman filters," *IEEE Trans. Ind. Electron.*, vol. 61, no. 10, pp. 5411–5418, Oct. 2014.

[5] T. Do-Xuan, V. Tran-Quang, T. Bui-Xuan, and V. Vu-Thanh, "Smartphone-based pedestrian dead reckoning and orientation as an indoor positioning system," in *Proc. Int. Conf. Adv. Technol. Commun. (ATC)*, Oct. 2014, pp. 303–308.

[6] H. Zhang, W. Yuan, Q. Shen, T. Li, and H. Chang, "A handheld inertial pedestrian navigation system with accurate step modes and device poses recognition," *IEEE Sensors J.*, vol. 15, no. 3, pp. 1421–1429, Mar. 2015.

[7] P. Kasebzadeh, C. Fritsche, G. Hendeby, F. Gunnarsson, and F. Gustafsson, "Improved pedestrian dead reckoning positioning with gait parameter learning," in *Proc. 19th Int. Conf. Inf. Fusion (FUSION)*, Jul. 2016, pp. 379–385.

[8] F. Li, C. Zhao, G. Ding, J. Gong, C. Liu, and F. Zhao, "A reliable and accurate indoor localization method using phone inertial sensors," in *Proc. ACM Conf. Ubiquitous Comput.*, 2012, pp. 421–430.

[9] M. Alzantot and M. Youssef, "UPTIME: Ubiquitous pedestrian tracking using mobile phones," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2012, pp. 3204–3209.

[10] H.-H. Lee, S. Choi, and M.-J. Lee, "Step detection robust against the dynamics of smartphones," *Sensors*, vol. 15, no. 10, pp. 27230–27250, 2015.

[11] E. M. Diaz and A. L. M. Gonzalez, "Step detector and step length estimator for an inertial pocket navigation system," in *Proc. Int. Conf. Indoor Positioning Indoor Navigat. (IPIN)*, Oct. 2014, pp. 105–110.

[12] A. Brajdic and R. Harle, "Walk detection and step counting on unconstrained smartphones," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2013, pp. 225–234.

[13] J. Kupke, T. Willemsen, F. Keller, and H. Sternberg, "Development of a step counter based on artificial neural networks," *J. Location Based Services*, vol. 10, no. 3, pp. 161–177, 2016. [Online]. Available: http://dx.doi.org/10.1080/17489725.2016.1196832

[14] J. Lin, L. L. H. Chan, and H. Yan, "A decision tree based pedometer and its implementation on the android platform," in *Proc. Comput. Sci. Inf. Technol.*, 2015, pp. 73–83.

[15] Y. Zhen-Jie, Z. Zhi-Peng, and X. Li-Qun, "An effective algorithm to detect abnormal step counting based on one-class SVM," in *Proc. IEEE 17th Int. Conf. Comput. Sci. Eng. (CSE)*, Dec. 2014, pp. 964–969.

[16] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bull. Calcutta Math. Soc.*, vol. 35, no. 1, pp. 99–109, 1943, Art. no. 14.

Stef Vandermeeren was born in Jette, Belgium, in 1992. He received the B.E. degree and the M.Sc. degree in electrical engineering from Ghent University, Ghent, in 2013 and 2015, respectively, where he is currently pursuing the Ph.D. degree with the Department of Telecommunications and Digital Information Processing. His research interests are in the general area of sensor fusion, indoor localization, and machine learning.

Samuel Van de Velde was born in Ghent, Belgium, in 1987. He received the B.E. degree from Vrije Universiteit Brussel, Brussels, Belgium, in 2008, and the M.Sc. degree in electrical engineering from Ghent University, Ghent, in 2010, where he is currently pursuing the Ph.D. degree with the Department of Telecommunications and Digital Information Processing. His research interests are in the general area of signal processing, optimization, algorithm design, and cooperative localization.

Herwig Bruneel was born in Zottegem, Belgium, in 1954. He received the master's degree in electrical engineering, the master's degree in computer science, and the Ph.D. degree in computer science from Ghent University, Belgium, in 1978, 1979, and 1984, respectively. He is a Full Professor with the Faculty of Engineering and Architecture and the Head of the Department of Telecommunications and Information Processing, Ghent University, where he also leads the SMACS Research Group.

He has co-authored the book *Discrete-Time Models for Communication Systems Including ATM* (Boston: Kluwer Academic Publishers, 1993) with B. G. Kim. His main personal research interests include stochastic modeling and analysis of communication systems and discrete-time queueing theory.

Since 2009, he has been holding a career-long Methusalem Grant from the Flemish Government at Ghent University, specifically on stochastic modeling and analysis of communication systems.

Heidi Steendam (M'01–SM'06) received the M.Sc. degree in electrical engineering and the Ph.D. degree in applied sciences from Ghent University, Ghent, Belgium, in 1995 and 2000, respectively. In 2015, she joined Monash University as a Visiting Professor. Since 1995, she has been with the Digital Communications Research Group, Department of Telecommunications and Information Processing, Faculty of Engineering, Ghent University, Belgium, first in the framework of various research projects and since 2002 as a Professor of Digital Communications.

She is the author of more than 150 scientific papers in international journals and conference proceedings, for which she received several best paper awards. Her main research interests are in statistical communication theory, carrier and symbol synchronization, bandwidth-efficient modulation and coding, cognitive radio and cooperative networks, positioning, and visible light communication.

Since 2002, she has been an Executive Committee Member of the IEEE Communications and Vehicular Technology Society Joint Chapter, Benelux Section, the Vice Chair since 2012, and the Chair since 2017. She was active in various international conferences as the Technical Program Committee Chair/Member and the Session Chair. In 2004 and 2011, she was the Conference Chair of the IEEE Symposium on Communications and Vehicular Technology in the Benelux. From 2012 to 2017, she was an Associate Editor of the IEEE TRANSACTIONS ON COMMUNICATIONS and the *EURASIP Journal on Wireless Communications and Networking*.