Winning the Sussex-Huawei Locomotion-Transportation Recognition Challenge

Vito Janko[1,2,*], Martin Gjoreski[1,2,*], Gašper Slapničar[1], Miha Mlakar[1], Nina Reščič[1,2], Jani Bizjak[1,2], Vid Drobnič[1], Matej Marinko[1], Nejc Mlakar[1], Matjaž Gams[1,2], Mitja Luštrek[1,2]

[1] Jožef Stefan Institute, Department of Intelligent Systems, Ljubljana, Slovenia

[2] Jožef Stefan Postgraduate School, Ljubljana, Slovenia

* The first two authors should be regarded as joint first authors.

Abstract The Sussex-Huawei Locomotion-Transportation Recognition Challenge presented a unique opportunity to the activity-recognition community to test their approaches on a large, real-life benchmark dataset with activities different from those typically being recognized. The goal of the challenge was to recognize eight locomotion activities (Still, Walk, Run, Bike, Car, Bus, Train, Subway). This chapter describes the submissions winning the first and second place. They both start with data preprocessing, including a normalization of the phone orientation. Then, a wide set of hand-crafted domain features in both frequency and time domain are computed and their quality evaluated. The second-place submission feeds the best features into an XGBoost machine-learning model with optimized hyper-parameters, achieving the accuracy of 90.2%. The first-place submission builds an ensemble of models, including deep learning models, and finally refines the ensemble's predictions by smoothing with a Hidden Markov model. Its accuracy on an internal test set was 96.0%.

Keywords Activity recognition, machine learning, deep learning, ensembles, HMM, competition

Introduction

Smart devices have become an indispensable part of our lives. Smartphones, smart watches and other wearables accompany us everywhere. The grand vision of ubiquitous computing is that these devices will know as much as possible about our context in order to provide the best possible service, and contribute to our safety, health, comfort and overall quality of life.

What we are doing at any given moment is a key element of our context, which is why activity recognition (AR) is intensely researched. Most research on AR is focused on our bodies, dealing with activities such as walking, sitting and lying. However, since we spend a lot of time in vehicles – transportation studies show that the average commute time is up to 80 minutes a day [17] and we also travel for other purposes – this is probably an area that deserves more attention. If the grand vision of ubiquitous computing is to be realized, our devices should know not only whether we are walking or sitting, but also whether riding a train or driving a car.

The Sussex-Huawei Locomotion-Transportation (SHL) dataset consists of seven months of recordings of smartphone sensors during eight modes of locomotion. It was collected to develop methods for AR, traffic analysis, localization, sensor fusion and other problems. These methods can support mobile services such as travel and traffic advice, adapting phone operation to the mode of locomotion (notifications, volume, Wi-Fi and GPS ...), using this mode in games, for music selection and a myriad of other purposes application developers will think of. Most importantly for this chapter, the dataset provides an excellent challenge for ubiquitous computing researchers – to apply existing and new methods for locomotion AR on more data than most of them are able to collect on their own and of course to take on the SHL Challenge.

This chapter describes two approaches to locomotion AR that placed first and second at the SHL Challenge. The first one is classical, based on over a decade of experience in the AR field. It starts with preprocessing the orientation data, continues with extracting a large number of expertly crafted features and selecting the best of them, and finishes with feeding the features into a classification model with tuned hyper-parameters. The second approach builds multiple models, some of which are trained with deep learning algorithms. The outputs of the models are combined into an ensemble, and the final predictions are smoothed with a Hidden Markov model (HMM). In designing both approaches, we applied the principle of multiple knowledge [13]: used different "viewpoints" (feature categories, models) and sensibly combined them (via feature selection, in an ensemble), with the expectation that the result will be superior to using single (high-quality) viewpoints. The two approaches were originally described in two HASCA workshop papers [14, 25]. They are described in this chapter as a single approach with several improvements, particularly regarding smoothing, and post-competition commentary is added.

Related Work

The AR domain has been thoroughly explored in the past using body-worn sensors, ambient sensors and combinations of the two. Here we focus only on body-worn sensors, since smartphones and smart watches are most frequently used for AR today.

The most frequent AR task is classifying activities related to movement, e.g., walking, running, standing still and cycling [4]. Different approaches using standard machine learning (ML) and feature extractions have been used and tested on various datasets [2, 3, 9, 10, 11, 12]. However, deep learning attempts are becoming increasingly prevalent [5]. Surprisingly, this domain is still not dominated by deep learning, unlike computer vision and some other domains, most likely because deep learning in AR is not clearly superior to classical ML.

Several attempts have been made in the past towards classification of just one activity, or distinguishing between activities related to one domain (e.g. transportation) [6, 7, 8]. However, the SHL Challenge seems to be more ambitious, trying to classify a wide variety of activities both human movement- and transportation-related. Therefore, the main and most important related work to this chapter are other approaches submitted to this challenge, some of which are included in this book.

Dataset

The SHL Dataset used in this work is publicly available and thoroughly described [1]. The subset used for the SHL Challenge was recorded with a Huawei Mate 9 smartphone carried inside the front right pocket (not fixed orientation) by a single participant over a period of four months for 5–8 hours per day.

The data comes from a variety of sensors in the smartphone: accelerometer, gyroscope, magnetometer, linear accelerometer, gravity, orientation (expressed with quaternions) and barometer; all sampled with the frequency of 100 Hz. The data is labeled with eight classes: Still, Walk, Run, Bike, Car, Bus and Subway. A class label was applied to each sensor sample. In aggregate, around 266 hours of labeled data was provided. The goal of the competition was to train a model on this data, and then use it to classify an additional 100 hours of unlabeled test data.

The provided data was split into 1-minute intervals, which were then shuffled -- with the original order for the labeled data provided as part of the dataset description. Since consecutive intervals can display substantial similarities (e.g., the phone may be in exactly the same orientation), it is essential that the dataset is placed in the original order before it is split into train, validation and test set. If this is not done, one of consecutive intervals is often placed in the train and the other in the validation/test set, making some irrelevant features (e.g., specific phone orientation) appear valuable for the prediction and thus resulting in overfitting.

We used 50% of the labeled data for training, 25% for validation and 25% for testing. After the parameters of the most accurate ML model were fixed, a different distribution – 90% for training and 10% for testing – was used to get an insight on how the behavior of the model changes if given more training data. For training of the final model, all the data was used. In all cases, we verified that the activity distribution remained similar in all listed splits.

Features

In order to apply most ML methods constituting our approach, the data had to be preprocessed and some features describing each signal extracted.

Preprocessing

First, the data was downsampled from 100 Hz to 50 Hz. This significantly reduced the computational load of the subsequent calculations, while not significantly affecting the classification accuracy. The additional practical benefit of the downsapling lies in reduced consumption of the sensor battery, should the system be used in a real-time setting.

Second, "virtual" sensor streams were calculated based on the real ones. These sensor streams had the same frequency and were used in the same manner as the original data streams for calculating features. They can be grouped in three categories:

- *Magnitudes*. Sensors with three axes had the magnitude ($m = \sqrt{x^2 + y^2 + z^2}$) calculated, which was used in addition to the sensor streams of individual axes.
- *De-rotated sensors.* These were computed by de-rotating acceleration and magnetometer data from body (phone) coordinate system to the North-East-Down (NED) coordinate system by multiplying them with the rotation matrix R_{NB} created with quarternions $[q_w, q_x, q_y, q_z]$ as given below. This was done in order to obtain orientation-independent sensor information, which is important to avoid overfitting to specific orientations of the phone. However, since orientation information can also be relevant, both groups of sensor streams were retained.

$$R_{NB} = \begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_y) \\ 2(q_x q_y + q_w q_z) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_w q_x) \\ 2(q_x q_z - q_w q_y) & 2(q_y q_z + q_w q_x) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_N = R_{NB} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_B$$

- *Roll, pitch and yaw.* Finally, quarternions were used to compute Euler angles – roll, pitch and yaw. Orientation is usually presented with quarternions to avoid the *gimbal lock* point of singularity, however, Euler angles are better for extracting features since each of them has a clear real-world meaning, whereas queternion components are only really meaningful when taken all together. Because of that, we only used Euler angles in the subsequent steps.

$$roll = \arcsin(2(q_w q_y + q_z q_x))$$

$$pitch = \arctan\left(\frac{2(q_w q_x + q_y q_z)}{1 - 2(q_x q_x + q_y q_y)}\right)$$

$$yaw = \arctan\left(\frac{2(q_w q_z + q_x q_y)}{1 - 2(q_y q_y + q_z q_z)}\right)$$

In the end, counting all separate sensor axes, we worked with 30 different data streams.

Feature extraction

Features were extracted from 1-minute windows of data. This window size was chosen as it was the largest possible given the limitations imposed by the nature of the competition, and we achieved the highest classification accuracy using it. Since transitions between activities are rare, long windows did not have a significant negative impact on the performance. Labels were calculated for each window as the most frequent label in that window.

Nearly all features were extracted from each individual data stream. Three categories of features were computed. First, domain features that have proven themselves in our previous work in similar domains [22, 23] including in a previously won competition [4]. These features are described in the work by Cvetković et al. [23].

Second, we tried to generate all features using the tsfresh library [19]. While the library is capable of generating a large number of features, is seemed too slow given the size of our dataset. Consequently, we only generated some features that seemed interesting and were not included in other categories – namely minimum, maximum, autocorrelation, number of samples above/below the mean and the average difference between two sequential data samples.

Finally, we calculated some features from the frequency domain, which describe the periodicity of the signals. These features were calculated using power spectral density (PSD), which is based on the fast Fourier transform (FFT). PSD characterizes the frequency content of a given signal and can be estimated using several techniques. The simplest one is to use a periodogram, which is obtained by taking the squared-magnitude of the FFT components. An alternative to periodogram is the Welch's method, which is also widely used and commonly considered superior to periodogram. It differs from a traditional periodogram in the fact that it computes the average of the periodograms of multiple overlapping segments of the signal to reduce the variance of the PSD. In our work, we opted to use the Welch's method to obtain the PSD.

Using the PSD is only suitable when the signal is clearly periodic. As we chose to use a 1-minute window, any periodic pattern in the signals is successfully captured, as shown in Fig. 1.

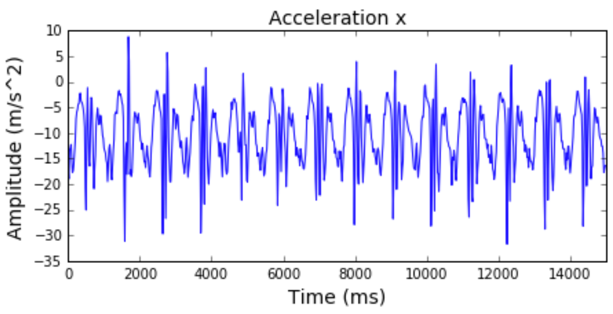

Fig. 1. Periodic pattern in a 15-second accelerometer segment during walking.

We implemented the frequency-domain features as given in related work [24]. Some were slightly modified or expanded in accordance with our expert knowledge. The following features were computed.

- *Three largest magnitudes.* Three peaks with the largest magnitude from the PSD were considered. These tell us the dominant frequencies in the signal. Both the magnitude values and the frequencies (in Hz) were taken as features.
- *Energy.* Calculated as the sum of the squared FFT component magnitudes. The energy was then normalized by dividing it with the window length.

$$energy = \frac{1}{N}\sum_{n=0}^{N-1}|x(n)|^2$$

where *x(n)* is the *n*-th FFT component and *N* is the window length.

- *Entropy.* Calculated as the information entropy of the normalized FFT component magnitudes. It helps discriminating between activities with similar energy features.

$$entropy = -\sum_{n=0}^{N-1} x(n)\log(x(n))$$

- *Binned distribution.* A normalized histogram, which is essentially the distribution of the FFT magnitudes. First, the PSD is split into 10 equal-sized bins ranging from 0 Hz to 25 Hz. Then, the fraction of magnitudes falling into each bin is calculated.
- *Skweness and kurtosis.* Calculated on the distribution-like PSD. Skewness and kurtosis describe the shape of the PSD. More precisely, skewness tells us about the symmetry of the distribution while kurtosis tells us about its flatness, as shown in Fig. 2.

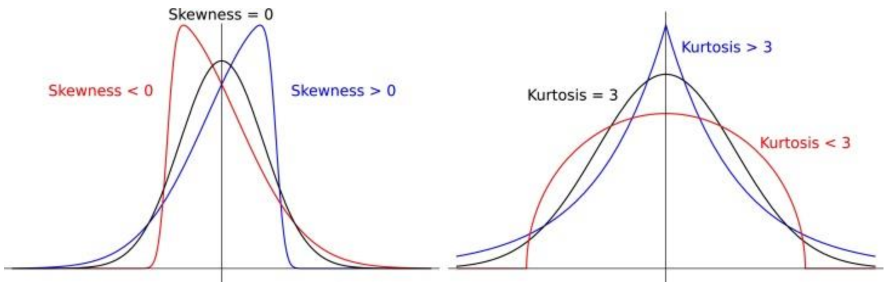

Fig. 2. Distributions with different skewness and kurtosis.

In total, 1696 features were computed and used in the subsequent steps.

Feature selection

Since a relatively high number of features was computed, feature selection was used to remove redundant and noisy ones in order to reduce overfitting and speed up the training process. Our feature selection consisted of three steps.

In the first step, the mutual information between each feature and the label was estimated [21], where larger mutual information means higher dependency between the feature and the label.

After the features were sorted according to mutual information, correlated features were removed based on the Pearson correlation coefficient [20]. This showed that roughly half of the features are redundant, which was expected because different data streams contained similar information. Since calculating the correlation of all feature pairs was computationally too expensive, only 100 features were taken at a time, starting with those with the highest mutual information with the label. Correlation was then calculated for each these pairs. If the correlation was higher than a certain threshold (experimentally determined as 0.8), the feature with the lower mutual information was discarded. After that, the next 100 features were added and the correlation between each pair was calculated again.

In the final step, features were selected using a greedy "wrapper" algorithm. A random forest (RF) model was first built on the train set using only the best scoring feature. The model was used to predict labels for the validation set and the prediction accuracy was calculated. Then the second-best feature was added and the model was built again. If the accuracy on the validation set was higher than without using this feature, the feature was kept. This procedure was repeated for all features. This strict selection initially led to overfitting to the validation set (the accuracy was much higher compared to the test set, on which the final model was tested), so the condition for keeping a feature was made less strict: the feature was kept if the accuracy did not decrease by more than an experimentally set threshold. Using this rule, overfitting to the validation set was reduced.

The best-performing features came roughly evenly distributed from each of the three categories, with those coming from the de-rotated magnetometer being the most significant, followed by those from accelerometer and gyroscope. The magnetometer came as a surprise, since it is the accelerometer that usually takes the spotlight in similar problems. It might be explained by the fact that the magnetometer is especially useful for transportation classification.

Machine learning

In line with the principle of multiple knowledge mentioned in the introduction, we built an ensemble of ML models followed by a HMM (Fig. 3). The ensemble consists of ten base models: nine models that take features as inputs, and one deep neural network (DNN) that takes spectrograms as inputs. Of the nine feature-based models, one is also a DNN, while the other eight were trained with different classical ML algorithms. The output class probabilities from the base models are fed into a Meta model, which outputs a class prediction. Finally, the class prediction of the Meta model is corrected by the HMM, which smooths the predictions. The details of each step are presented in the following subsections.

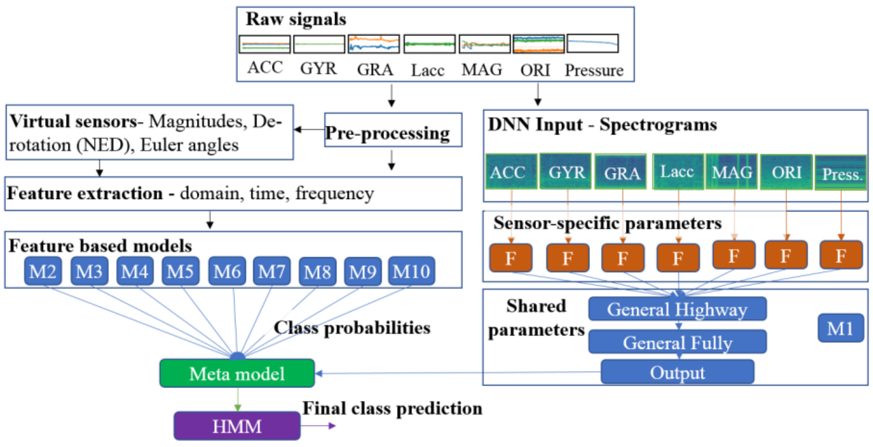

Fig. 3. Our machine-learning architecture.

Base models of the ensemble

The feature-based models are built using features extracted from the sensor data as described in the previous section. After the feature extraction, the data was fed to the following nine ML algorithms: DNN, Random Forest, Gradient Boosting, Extreme Gradient Boosting, SVM, AdaBoosting, KNN, Gaussian Naïve Byes and Decision Tree. The algorithms were used as implemented in the scikit-learn python library. The models' hyperparameters were tuned using 2-fold randomized parameter search. For the feature-based DNN, we experimented with different architectures, and the best performing was the architecture with 2 fully connected dense layers with 256 and 128 neurons.

For the spectrogram-based DNN, we again experimented with different architectures, including convolutional neural networks (CNNs) and long-short term memory networks (LSTMs). The final architecture is depicted in the right half of Fig. 1. For each category of raw sensor data, i.e., 3D acceleration, 3D gyroscopes, 3D linear acceleration, 4D orientation, 3D magnetometer and pressure data, a spectrogram representation is calculated using the Fourier transformation. The spectrograms are represented as vectors with dimensions $P \times T \times N$. P represents the number of spectral bands; T represents the time for which the spectral power is calculated; N represents the number of axes for the specific sensor type (e.g., the accelerometer has three axes, the orientation sensor has four axes and the pressure sensor has only one axis). The vectors are used as input to a fully connected DNN. The first layer of the network is a sensor-specific layer, which learns sensor-specific parameters. There are 128 neurons for each sensor type, thus there are in total 896 (7×128) neurons in the sensor-specific layer. The output of the sensor-specific layer is merged using a shared Highway layer, which is followed by a fully connected layer with 1024 neurons. The output of the model is obtained from the final layer with a softmax activation function yielding a class probability distribution.

To avoid overfitting, L2 regularization and dropout methods were used for all DNN models. The dropout probability was set to 0.3. The training was fully supervised, by back propagating the gradients through all layers. The parameters were optimized by minimizing the cross-entropy loss function using the Adam optimizer. The models were trained with a learning rate of 10^{-4}. The batch size was set to 2000.

Meta model of the ensemble

The Meta model takes as inputs the class probabilities output of each of the ten base models. We evaluated Meta models built with seven ML algorithms: Random Forest, Gradient Boosting, SVM, AdaBoosting, KNN, Gaussian Naïve Byes and Decision Tree. Each Meta model was trained on the validation set and evaluated on the test set. The models' hyperparameters were tuned using 2-fold randomized parameter search on the models' training data. The best-performing model on the test data was picked as the final Meta model.

HMM smoothing

Classification accuracy can be improved by considering the probability of a classified sequence. For example, the classified sequence: Train, Bus, Train, Train, Train, makes little practical sense, particularly in a short time interval, e.g., a couple of seconds. A misclassification of the "Bus" instance is much more probable than the user switching from a bus to a train and back in that time.

In order to find and correct this kind of mistakes, we employed the HMM method. This method assumes that there are some hidden internal states (in our case activities) that emit some signal at each time step (in our case classifications). The parameters of such a system are both the probabilities of transitions and emissions. Both can easily be inferred from the dataset – all we need are transition probabilities between each pair of activities and the confusion matrix of the model. Having the parameters of the system, a sequence of sequential classifications is given to the HMM method, which returns the most likely sequence of internal states – activities.

There are two possible scenarios where the HMM method can be used. In the first case (see Fig. 4) the whole classified sequence is known in advance. In this case, the HMM method can be used directly. In a real-life setting, this corresponds to reporting the classified activities to the user with a delay (as the HMM method uses instances classified after the current one). Different delays were tested to determine the relation between the sequence length and the method's usefulness.

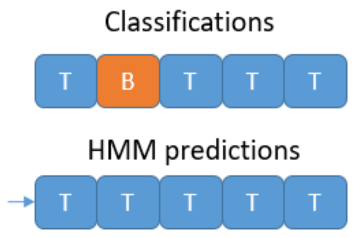

Fig. 4. Sequential classified windows, compared to HMM predictions. T = Train, B = Bus.

In the second case (see Fig. 5), we have the entire history of classifications, but we cannot see the future ones. In a real-life setting, this corresponds to reporting activities to the user as they happen. This can be implemented by using the HMM to predict only the last element of the sequence, while iteratively lengthening it.

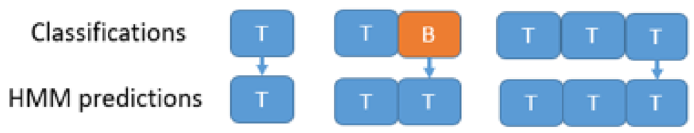

Fig. 5. Iterative HMM predictions. Sequence is iteratively lengthened as more of the history becomes known. Only the last instance in the sequence in changed after each step.

The HMM method requires instances to be ordered in the correct temporal sequence. This was not a problem for the internal test set, where ordering was provided. However, for the final test data the correct sequence of instances was unknown. To overcome this problem, we devised an algorithm that could automatically sort the one-minute intervals in the correct order. This was possible due to the fact that the last sensor reading of one window is very similar to the first sensor reading in the subsequent window, assuming the correct ordering. The algorithm iteratively searched for the next window that matched the above criteria. Doing so, it almost perfectly reconstructed the original ordering allowing us to use the HMM smoothing on the final prediction.

Single-model alternative

The previously described methodology – using an ensemble and HMM smoothing – is quite complex and thus risky. We could not ignore the possibility of a human error, and it was also possible that the labeled data available for internal testing prior to the submission of the results to the SHL Challenge was different from the competition evaluation data. Because of that we decided to submit two entries, one of which used a single model and no smoothing.

The single-model alternative used an Extreme Gradient Boosting model (XGB) [18], which performed best of the base models in the ensemble, and is also often the best-performing algorithm in various ML competitions. XGB is an upgraded version of the gradient boosting algorithm. The implementation of XGB offers several advanced features for model tuning, computing environments (e.g., parallelization across several CPU cores, distributed computing for large models, cache optimization, etc.) and algorithm enhancement (e.g., handling missing values, optimal usage of memory resources, etc.). It is capable of performing the three main forms of gradient boosting (Gradient Boosting (GB), Stochastic GB and Regularized GB) and it is robust enough to support fine tuning and addition of regularization parameters. According to the author, the main difference is that XGB uses a more regularized model formalization to control overfitting, which gives it better performance.

In order to obtain best results, XGB requires careful hyper-parameter tuning. For comparison we first trained the XGB model with default parameter values and then improved the model through the tuning of its parameters. XGB has three major groups of parameters:

- *General parameters* that relate to the boosting algorithm that we use, commonly a tree or a linear model. For this problem, we used the tree model.
- *Booster parameters* that depend on which booster is chosen. We chose the tree booster, so the parameters in this section will be tree-specific parameters.
- *Learning task parameters* that decide on the learning scenario, for example, which evaluation metric should be optimized. Regression tasks may use different parameters than classification tasks.

Since XGB has more than 30 hyper-parameters, the parameter tuning process could not be done in one step, so we did it iteratively. This means that we optimized one or two parameters at a time while keeping the other parameters at default values. After finding the optimal value for the selected parameter(s), we fixed this value and started optimizing another parameter. For this iterative process, we started with the more important parameters as follows:

- We chose a relatively high learning rate. Learning rate defines the amount of "correction" we make at each step (each boosting round is correcting the errors of the previous round). So having a lower learning rate makes our model more robust to over-fitting and usually gets better results. But with a lower learning rate, we need more boosting rounds, which takes more time to train the model. Since we needed to fit a lot of parameters we started with a higher learning rate.
- We tuned tree-specific boosting parameters (max_depth, min_child_weight, gamma, subsample, colsample_bytree) for the chosen learning rate and number of trees.
- We tuned regularization parameters for boosting (lambda, alpha), which can help reduce model complexity and enhance performance.
- We lowered the learning rate to obtain the optimal parameter values.

We started by setting the initial values for the parameters that were going to be optimized. We chose:

- max_depth = 5
- min_child_weight = 1
- gamma = 0
- subsample, colsample_bytree = 0.8
- reg_alpha = 0.005

Note that these were just initial estimates and were tuned later. We took the default learning rate of 0.1 and checked for the optimal number of trees using the cv function of XGB which performs cross-validation at each boosting iteration and thus returns the optimal number of trees required. The obtained number of trees was 140.

The first parameters tuned were max_depth and min_child_weight as they were expected to have the highest impact on the model outcome. To start with, we set wider possible ranges and then we performed another iteration for smaller ranges around the best values obtained in the first step. The optimal obtained values were max_depth = 6 and min_child_weight = 1.

In the next step, we focused on tuning the gamma value. We used the parameters already tuned and searched for the optimal gamma value. The search showed that the initial value of gamma = 0 was the most suitable one.

Next, we tuned the parameters subsample and colsample_bytree. The optimal obtained values were subsample = 0.65 and colsample_bytree = 0.75.

Next, we applied regularization to reduce overfitting, even though the gamma parameter already substantially controls the model complexity. The best found value was reg_alpha = 0.0001.

Finally, we reduced the learning rate and increased the number of trees. We used learning_rate = 0.01 and number_of_trees = 5000. With this step, we added a significant boost in performance and the effect of parameter tuning became clearer. For comparison, the accuracy obtained with default parameter values was 88.3% and after parameter tuning we obtained the result of 90.2%.

Experimental results

The first step in developing our approach was to decide on the temporal length of the windows to classify. Different window lengths were tested in order to determine the optimal one. Window length of 1 minute was proven the best, as shown by the highest accuracy in Table 1. Only frequency-domain features were used in this experiment to train a RF model, as these were the fastest to compute and evaluate.

Table 1. Accuracy using frequency-domain features computed on windows of length 5 seconds, 30 seconds and 60 seconds. Random Forest was used to train and evaluate the model.

Window length	Accuracy [%]
5 seconds	67.1
30 seconds	72.5
60 seconds	74.9

All subsequent results were obtained by using the 1-minute-majority labels. Per-sample accuracy was also computed, but it was consistently around 0.5 percentage point lower, which is insignificant compared to the accuracy gains when using larger windows. This can be attributed to long-on-average activities, due to which activities very rarely changed in the middle of a given window.

Accuracies of the RF model used in feature selection are given in Table 2. One can note that the accuracy on the test set increases with each step. The accuracy on the test set is slightly higher than on the validation set in the first three lines, presumably because the test instances were easier to classify on average. In the last line ("Wrapper Strict"), the accuracy on the test set drops, suggesting that we overfit to the validation data. The difference between "Wrapper" and "Wrapper strict" is in the threshold to select a feature. The threshold was higher in "Wrapper Strict", so fewer features were selected – apparently such that did not generalize very well beyond the validation set. The same experiment was conducted with the XGB model, obtaining similar results: overall, the accuracy was higher, but the increase after each feature selection step was retained. The "Wrapper" feature set was thus chosen for the final model.

Table 2. The number of features kept and accuracy of the RF model after each step of feature selection on both validation and test sets.

	Features	Accuracy [%]	
		Validation	Test
All Features	1696	78.9	82.2
Correlation removed	816	80.7	83.0
Wrapper	359	82.0	83.6
Wrapper (Strict)	90	83.5	82.6

After choosing the "Wrapper" feature set, we proceeded to test different machine learning algorithms on the validation set. The resulting models were used as base models in our ensemble (with the exception of XGB, which was also used on its own). Table 3 summarizes the experimental results. The first column represents the accuracies of the 10 base models + the majority classifier. The middle column represents the accuracies of the complete ensemble using Meta models trained with different machine-learning algorithms. The rightmost column represents the accuracies of the ensemble with its predictions smoothed by the HMM method.

HMM-Past considers only the past data, HMM-2 and HMM-6 provide the output after a 2 or 6 time slots, while HMM-All had all the data as input.

Table 3. Accuracy on the internal test data.

Test Accuracy					
Base models		**Meta models**		**HMM models**	
Majority	16.0%	RF-Meta	92.0%	HMM-Past	94.0%
RF	84.8%	SVM-Meta	90.6%	HMM-2	95.0%
SVM	87.1%	**GB-Meta**	**92.2%**	HMM-6	95.5%
GB	89.5%	ADA-Meta	68.5%	**HMM-All**	**96.0%**
ADA	60.0%	KNN-Meta	90.8%		
KNN	81.5%	NB-Meta	85.9%		
NB	76.2%	DT-Meta	87.5%		
DT	74.1%				
XGB	**90.2%**				
DNN-Feat.	89.4%				
DNN-Spec.	81.8%				

For the base models it can be seen that the highest accuracy of 90.2% is achieved by XGB. This model was used for the SHL Challenge submission that placed second. The spectrogram-based DNN (DNN-Spec.) had 7.6 percentage points lower accuracy compared to the feature-based DNN (DNN-Feat.), suggesting that the selection of high-quality features importantly contributed to our success. For the ensemble using Meta models, it can be seen that the Meta model built with the Gradient Boosting algorithm (GB) had the highest accuracy of 92.2%.

Finally, the results using the HMM method show that the HMM significantly increases the accuracy up to 96%. When working with past data only, this benefit is halved, but it is still present. The small accuracy difference between HMM-6 and HMM-All indicates that a couple of time slots are sufficient to smooth the data. Also, HMM-Past has the lowest achieved accuracy compared to the other three HMM variations, indicating that classifying with some delay is better than classifying immediately. HMM-All was used for the SHL Challenge submission that placed first. Fig. 6 presents the normalized confusion matrix for this model. The normalization is performed per row (i.e., the sum in each row is 100). From the confusion matrix it can be seen that the most problematic activities are Train and Subway.

	Still	Walk	Run	Bike	Car	Bus	Train	Subway
Still	95.3	1.1	0.0	0.2	0.3	0.3	2.0	0.8
Walk	0.3	98.7	0.2	0.6	0.0	0.2	0.0	0.0
Run	0.0	1.1	98.9	0.0	0.0	0.0	0.0	0.0
Bike	0.0	0.5	0.0	99.3	0.0	0.0	0.2	0.0
Car	0.5	0.2	0.0	0.0	99.4	0.0	0.0	0.0
Bus	0.9	0.2	0.2	0.0	0.0	98.8	0.0	0.0
Train	2.0	0.2	0.0	0.0	0.0	0.0	82.0	15.9
Subway	0.0	0.0	0.0	0.0	0.0	0.0	4.8	95.2

Fig. 6. Normalized confusion matrix on the internal test data for the model with highest accuracy, the HMM-All model.

For the submission that won the SHL Challenge, we used the predictions of the HMM-all model. For the submission ranked as second at the challenge, we trained the models on all the available labeled data and used them to evaluate the unlabeled competition evaluation data. To get an insight in the behavior of the "fully trained" model, we also trained an XGB model on 90% of the data and classified the remaining 10%, achieving the accuracy of 93.7%. We compared its classifications with the ones made with the model that was trained on only 50% of the data and found them very similar (they matched in 97.2% of cases).

Since the used dataset is large and the approach complex, it is worth mentioning the computational complexity. A workstation with a four-core 3.3 GHz CPU, 16 GiB of RAM and nVIDIA GeForce GTX1070 graphic card was used. The preprocessing and feature extraction required ~6 hours, the model training required ~30 minutes, and the model testing required ~1 minute on the internal test data (once the features were extracted). Additional 3 hours were required to classify the competition evaluation data, mostly due to computationally expensive process of calculating the features.

Conclusion and discussion

The SHL Dataset presents a uniquely large and sensor-rich dataset from real life. It provides an open platform for creating and testing various AR and other algorithms. By containing activities not common in AR, such as "Train" and "Subway", it opens new challenges for the AR community. The SHL Challenge was an effective way to jumpstart the research on the dataset, yielding the first solutions to the basic locomotion AR problem.

We approached the SHL Challenge systematically, first by paying attention to the organization of the work. The group of 11 people consisting of senior and junior researchers as well as a few of students was split into two teams. The first team – JSI-Classic – concentrated on processing the input data and applying one best classical ML method [14], finally placing second in the competition. The second team – JSI-Deep – focused on deep learning, ensembles and smoothing, placing first.

At the beginning, the teams did not share information in order to encourage original thinking and maximize the benefits of multiple knowledge [13]. Only in the last weeks did they start exchanging ideas, data and software, so that both teams converged to the best solutions within their strategy. The strategy of JSI-Classic was to use a classical and safe approach, while the strategy of JSI-Deep was to use the maximally sophisticated and complex approach.

The high accuracy reported in this chapter can be attributed both to careful preprocessing, feature extraction and selection, as well as to successful use of multiple knowledge, implemented in the form of an ensemble, and on successful smoothing using the HMM method. If we assume 70% accuracy reported in the SHL Dataset authors' paper [1] as the baseline, the contribution of preprocessing, feature extraction and selection, and selection of a good ML model amounts to roughly 20 percentage points, corresponding to the accuracy of around 90% achieved by the JSI-Classic team [14]. This can be broken down into 12 percentage points for the preprocessing and feature extraction, 1–2 percentage points for the feature selection, 5 percentage points for using XGB over the commonly used RF, and 2 percentage points for tuning its hyperparameters.

The additional 2 percentage points due to the ensemble and 4 percentage points due to HMM smoothing, corresponding to the accuracy of 96% achieved by the JSI-Deep team, do not seem much compared to the 20 percentage points of the JSI-Classic approach, but are still rather significant, having in mind that 100% is the absolute limit. The DNNs deserve special comment, since they are the name-sake of the JSI-Deep team and not commonly used for AR. The spectrogram-based model had a 7.6 percentage points lower accuracy compared to the feature-based model. This indicates that the features contain more information than spectrograms, which seems reasonable, since they contain additional specialized time-domain information, i.e., hand-crafted features that are based on the sensor's amplitudes. The spectrograms, on the other hand, only contain information about the change in the frequency bands over time. The DNN model using features was comparable to the best classical models, demonstrating that deep and classical approaches are comparable in this case.

The competition results were reported in average F-score, and we achieved the F-score of 0.94 which closely matched the F-score we achieved in our internal testing (0.96). Looking at the results of other competitors, one can see a large discrepancy between the results reported on their internal test sets and the results provided by the organizers on the competition evaluation dataset. The easiest explanation is that many competitors did not restore the labeled dataset to the original order, allowing them to heavily overfit to the training data as explained in the dataset description. Based on this, we can conclude that a key achievement for success was avoiding overfitting, although all the elements of our approach were needed to overcome the closest competitors.

Winning the first and second place suggests that the approach described in this chapter represents the state of the art in (locomotion) AR, both in technical and organizational terms. As such, it might be of use to researchers in the AR community.

References

1. Gjoreski H, Ciliberto M, Wang L, Morales FJO, Mekki S, Valentin, Roggen S (2018) The University of Sussex-Huawei Locomotion and Transportation Dataset for Multimodal Analytics with Mobile Devices. IEEE Access 6:42592-42604. doi: 10.1109/ACCESS.2018.2858933
2. Hung WC et al (2014) Activity recognition with sensors on mobile devices. In: International Conference on Machine Learning and Cybernetics (ICMLC) 2014. Vol. 2. IEEE
3. Ronao CA, Cho S-B (2016) Human activity recognition with smartphone sensors using deep learning neural networks. Expert Systems with Applications 59:235-244. Doi:10.1016/j.eswa.2016.04.032
4. Kozina S, Gjoreski H, Gams M, Luštrek M (2013) Efficient Activity Recognition and Fall Detection Using Accelerometers. In: Botía J.A., Álvarez-García J.A., Fujinami K., Barsocchi P., Riedel T. (eds) Evaluating AAL Systems Through Competitive Benchmarking. EvAAL 2013. Communications in Computer and Information Science, vol 386. Springer, Berlin, Heidelberg
5. Gjoreski H et al (2016) Comparing deep and classical machine learning methods for human activity recognition using wrist accelerometer. In: Proceedings of the IJCAI 2016 Workshop on Deep Learning for Artificial Intelligence, . vol. 10. New York, NY, USA, 2016
6. Ravì D, Wong C, Lo BP, Yang G (2017) A Deep Learning Approach to on-Node Sensor Data Analytics for Mobile or Wearable Devices. IEEE Journal of Biomedical and Health Informatics 21:56-64.
7. Wang S, Chen C and Ma J. (2010) Accelerometer Based Transportation Mode Recognition on Mobile Phones. In: Asia-Pacific Conference on Wearable Computing Systems, Shenzhen, 2010, pp. 44-46. doi: 10.1109/APWCS.2010.18

8. Reddy et al. (2010) Using mobile phones to determine transportation modes. ACM Transactions on Sensor Networks (TOSN) 6(2):13.
9. Roggen D et al. (2010) Collecting complex activity data sets in highly rich networked sensor environments. In: Seventh International Conference on Networked Sensing Systems (INSS'10), Kassel, Germany, 2010.
10. Teng H et al. (2018) Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning. In: GigaScience 7(5). doi:10.1093/gigascience/giy037
11. Janko V et al. (2017) e-Gibalec: Mobile application to monitor and encourage physical activity in schoolchildren. Journal of Ambient Intelligence and Smart Environments 9(5):595-609.
12. Cvetković B et al. (2017) Real-time physical activity and mental stress management with a wristband and a smartphone. In: Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers. ACM, 2017.
13. Gams M (2001) Weak intelligence: through the principle and paradox of multiple knowledge. Nova Science
14. Janko V et al. (2018) A New Frontier for Activity Recognition -- The Sussex-Huawei Locomotion Challenge. In: Adjunct Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers, Singapore, Singapore, October 08 - 12, 2018
15. Wang L, Gjoreski H, Murao K, Okita T, Roggen D (2018) Summary of the Sussex-Huawei Locomotion-Transportation Recognition Challenge. In: Proceedings of the 6th International Workshop on Human Activity Sensing Corpus and Applications (HASCA2018). Singapore, Oct. 2018
16. Sussex-Huawei Locomotion Challenge Database (2018) http://www.shl-dataset.org/activity-recognition-challenge
17. Olsson LE, Gärling T, Ettema D et al. (2013) Social Indication Research 111: 255-263. doi:10.1007/s11205-012-0003-2
18. XGBoost (2018) http://xgboost.readthedocs.io/en/latest/. Accessed 13 Feb 2019
19. Tsfresh (2018) http://tsfresh.readthedocs.io/en/latest/. Accessed 13 Feb 2019
20. Pearson correlation coefficient. http://en.wikipedia.org/wiki/Pearson_correlation_coefficient.
    Accessed 13 Feb 2019
21. Mutual info score. http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html. Accessed 13 Feb 2019
22. Cvetković B, Janko V, Luštrek M (2015) Demo abstract: Activity recognition and human energyexpenditure estimation with a smartphone. In: 2015 IEEE InternationalConference on Pervasive Computing and Communication Workshops (PerCom Workshops), IEEE, 193–195
23. Cvetković B, Szeklicki R, Janko V, Lutomski P, Luštrek M. (2018) Real-time activity monitoring with awristband and a smartphone. Information Fusion 43:77–93
24. Su X, Tong H, Ji P (2014) Activity recognition with smartphone sensors. Tsinghua Science and Technology 19(3):235–249
25. Gjoreski M et al. (2018) Applying multiple knowledge to Sussex-Huawei locomotion challenge. In: Adjunct Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers, Singapore, Singapore, October 08 - 12, 2018