Data Analytics for the Smart Society: Human Activity Recognition

Giovanni Bignardi

May 29, 2017

1 Brief overview

The train partition is composed by 8 labelled sequences, each one represents the records for a human being. Each sequence is composed by 5-dimensional signals corresponding to the person performing a certain set of activities (running, walking, standing, sitting, lying) in seminaturalistic conditions. The first 2 dimensions concern data from an Accelerometer: the z-axis and the modulus of the (x,y) axes. The following three refer to a 3-axes gyroscope. The lenght of a sequence representing a person varies from one to another, in a range from approximately sixteen thousand observations, to nineteen thousand. Each observation (sampled with a 16Hz frequency) is labelled with a number from 1 to 5:

1. walking
2. running
3. standing
4. sitting
5. lying

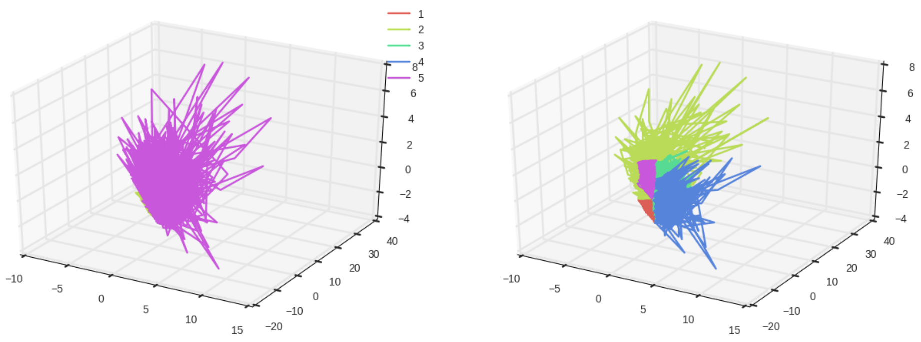
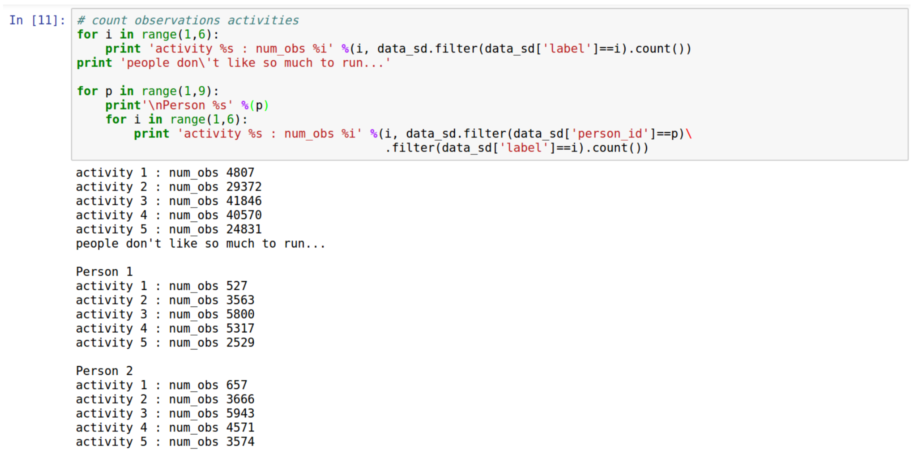


Figure 1: Here a representation of the data reduced in 3 dimensions

The first is a plot showing data coloured according to the activities. There was no way to avoid the overlapping. The second plot use some k-means clustering to group the data but with no appreciable results.

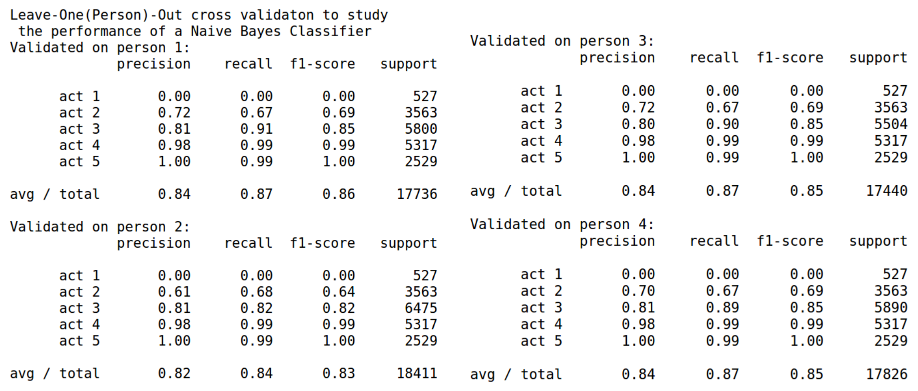
Before beginning to present the models, another information worth to be mentioned is in which proportion the activities appear in the entire dataset.



```
In [11]: # count observations activities
         for i in range(1,6):
             print 'activity %s : num_obs %i' %(i, data_sd.filter(data_sd['label']==i).count())
         print 'people don\'t like so much to run...'

         for p in range(1,9):
             print'\nPerson %s' %(p)
             for i in range(1,6):
                 print 'activity %s : num_obs %i' %(i, data_sd.filter(data_sd['person_id']==p)\
                                                      .filter(data_sd['label']==i).count())

         activity 1 : num_obs 4807
         activity 2 : num_obs 29372
         activity 3 : num_obs 41846
         activity 4 : num_obs 40570
         activity 5 : num_obs 24831
         people don't like so much to run...

         Person 1
         activity 1 : num_obs 527
         activity 2 : num_obs 3563
         activity 3 : num_obs 5800
         activity 4 : num_obs 5317
         activity 5 : num_obs 2529

         Person 2
         activity 1 : num_obs 657
         activity 2 : num_obs 3666
         activity 3 : num_obs 5943
         activity 4 : num_obs 4571
         activity 5 : num_obs 3574
```

Figure 2: tot for each activity over the entire dataset and following just for person 1 and for person 2. Find the rest of count in the attached ipython notebook 'HAR spark RForest'

The interesting thing is that the observations come from real moniotring and tracking records, and exactly because of that the periods of running activities are really few, confronted for example to those of standing or sitting.
Eventually this setting could bias the model under the cover and we would not notice it. What I am referring to can be better explained through the example of testing the accuracy of a Naive Bayes classifier with a leave-one(person)-out cross-validation. Throughout the iterations, the overhall accuracy obtained predicting on the lef-out person could seem high, but if we observe the results in deep, i.e. the accuracy obtained for each activity, suddenly it shows up that the algorithm completely fails in recognizing the activity of running.



```
Leave-One(Person)-Out cross validaton to study
 the performance of a Naive Bayes Classifier
Validated on person 1:
             precision    recall  f1-score   support

      act 1       0.00      0.00      0.00       527
      act 2       0.72      0.67      0.69      3563
      act 3       0.81      0.91      0.85      5800
      act 4       0.98      0.99      0.99      5317
      act 5       1.00      0.99      1.00      2529

avg / total       0.84      0.87      0.86     17736

Validated on person 2:
             precision    recall  f1-score   support

      act 1       0.00      0.00      0.00       527
      act 2       0.61      0.68      0.64      3563
      act 3       0.81      0.82      0.82      6475
      act 4       0.98      0.99      0.99      5317
      act 5       1.00      0.99      1.00      2529

avg / total       0.82      0.84      0.83     18411

Validated on person 3:
             precision    recall  f1-score   support

      act 1       0.00      0.00      0.00       527
      act 2       0.72      0.67      0.69      3563
      act 3       0.80      0.90      0.85      5504
      act 4       0.98      0.99      0.99      5317
      act 5       1.00      0.99      1.00      2529

avg / total       0.84      0.87      0.85     17440

Validated on person 4:
             precision    recall  f1-score   support

      act 1       0.00      0.00      0.00       527
      act 2       0.70      0.67      0.69      3563
      act 3       0.81      0.89      0.85      5890
      act 4       0.98      0.99      0.99      5317
      act 5       1.00      0.99      1.00      2529

avg / total       0.84      0.87      0.85     17826
```

Figure 3: the remaining scores can be found in the attached ipynb 'HAR sklearn'

The result is not upsetting: if we consider the small support of activity 1 in each person records and the theory behind Naive Bayes, like the assumption that every feature is independent, we could figure out why the model does never recognize an observation as running. But the reason for inserting this output

is that it warns to not forget considering the influence of different proportions among activities. Checking how the single activities are predicted ensures to avoid choosing biased models, which are succesfull with the main labels but perform badly on those with low-numbers support.
Finally, in their paper[1] Avci, Bosch, Marin-Perianu and Havinga propose two intermediary steps, concerning a proceure of record segmentation first, and a subsequent feature extraction. This could be either an efficient dimensionality reduction (if the number of features extracted is less than the segment length) or a data abstraction aimed to feed the algorithms with a faithful, but easier-to-digest, representation of the original data.
Naturally, to be properly conduced it is necessary to possess a certain scientific awareness of the statistics you are extracting, which cannot be limited to the classic time-domain features (verified: I was able in Spark to segment data and extract mean, variance and spectral entropy but this did not improve the results, if confronted on an approach on raw data). The models need higher-level features such as those reported in the paper: Signal Magnitude Area, Inter-axis Correlation, Wavelet Coefficients, Spectral Energy. Beyond the need of a scientific awareness, which can be temporarely integrated with the help of scientific computing packages, what I am concerned about is the segmentations step, because its implementation entails a loss of frequency in the data. Here I am not challenging neither the results obtained in the paper nor the approach of any other eminent researcher in the field, but I thought it would have been interesting to understand if it was possible to achieve accurate results directly operating on raw data.

[1] A.Avci, S.Bosch, M.Marin-Perianu, R.Marin-Perianu and P.Havinga, "Activity Recognition Using Inertial Sensing for Healthcare, Wellbeing and Sports Applications: A Survey", University of Twente, The Netherlands

2 Hidden Markov Models

A straightforward approach is to build a Hidden Markov Model for each one of the 5 activities. Therefore, at the end of the training procedure, I will have 5 different models. To predict the label for the observations in the test set, we could simply compute the log-likelihood of each activity's HMM on the 5-dimensional signal, and assign that observation to the activity with the max HMM log-likelihood.

2.1 Find the proper number of hidden states for each model

Given that the activities are different, the sequence of 5-dim signals recorded while a certain activity is performed probably have a different shape than signals belonging to another action. For example I could intuitively expect a wave with larger oscillation for an activity as running, with respect to the others. According to this intuition I woul probably need an higher number of hidden states to describe it. Moreover I recall that we are studying people, not robot, hence I have to consider how each person "do activities" in a unique way; I expect divergencies in nearly every activity-related meausre: the rythm, how they balance their bodies, how they breath, even if sometimes it is just a slight divergence. Because of that, when I train an HMM on a set of people (especially when it is a reduced as our train partition with 8 individuals) I guess it is important to assume they can be either very different from each other or overhall similar except for an odd person, or whatever different combination. Therefore the hidden-states tuning has to be conduced with a certain care: I deploy a LOO cross-validation approach, adapted in this case as a Leave-one-(person)-out. Precisely, what does the function in the ipython notebook do?

- I can provide it with the label of an action (1,2,3,4,5 in this case) and the number of hidden states I want to test (if I give for example, 15, it means every int from 1 to 15).
- I use the function loo from the scikitlearn.crossvalidation library to automatically compute leave-lone-out cross-validation indices for the 8 people in the train dataset.
- From now on, it is a structure of iterations into iterations:
- For each hidden-state I want to try:
- For each pair of indices (ex: train [0,1,2,3,4,5,6],validation[7]) yielded by the leave-one-out function:
  - I split the original train dataset: 7 people as the (sub)train partition and I keep one out for validation.
  - From each person's record in the (sub)train partition I extract only the singals belonging to the action under study and I concatenate them into a unique array.
  - On this array I fit a HMM with Gaussian emission, **diagonal covariance matrix**, and the tried number of hidden states.

  - I compute the log-likelihood of that HMM on the person representing the validation partition (obviously this was as well subsetted only for the signals belonging to the analyzed action). This log-likelihood is stored.
  - At the end of the cross-validation for hidden-states = i , I will have 8 different log-likelihood scores, each one yielded by the HMM fitted on 7 people and validated on the left-out person.
  - I compute the average out of them and consider it a reasonable benchmark for the success of an HMM with hidden-states = i for the prediction task on the action under consideration.

- Once I have repeated the procedure for all the different hidden-states I desired to test, I can confront the (average) log-likelihoods yielded by each one of them during the LOO cross-validation, and choose the number of hidden-states that returns the maximum score.

The important aspect of this approach is that, using the Leave-one-out approach, the log-likelihood scores I obtain from each iteration are very sensitive to the peculiarity of the person I'm testing on. This will be evidenced soon commenting my results.

The maximum number of hidden states
I have previously said that a low number of hidden-states could be an excessive generalization for more complex activities like running or walking, which would require a more complex representation (more hidden states). On the other hand, I have noticed that limiting the threshold for the maximum number of hidden states could save from situations of overfitting. Let's start from what I have achieved setting a high threshold (20) for the maximum number of hidden states that my tuning function is allowed to try during the cross-validation of the HMM.

- 1) Running: 13 hidden states
- 2) Walking: 17 hidden states
- 3) Standing: 17 hidden states
- 4) Sitting: 2 hidden states
- 5) Lying: 2 hidden states

While setting a more restrained threshold (10, just twice the size of signal's dimensions) the best number of hidden-states returned for each action is:

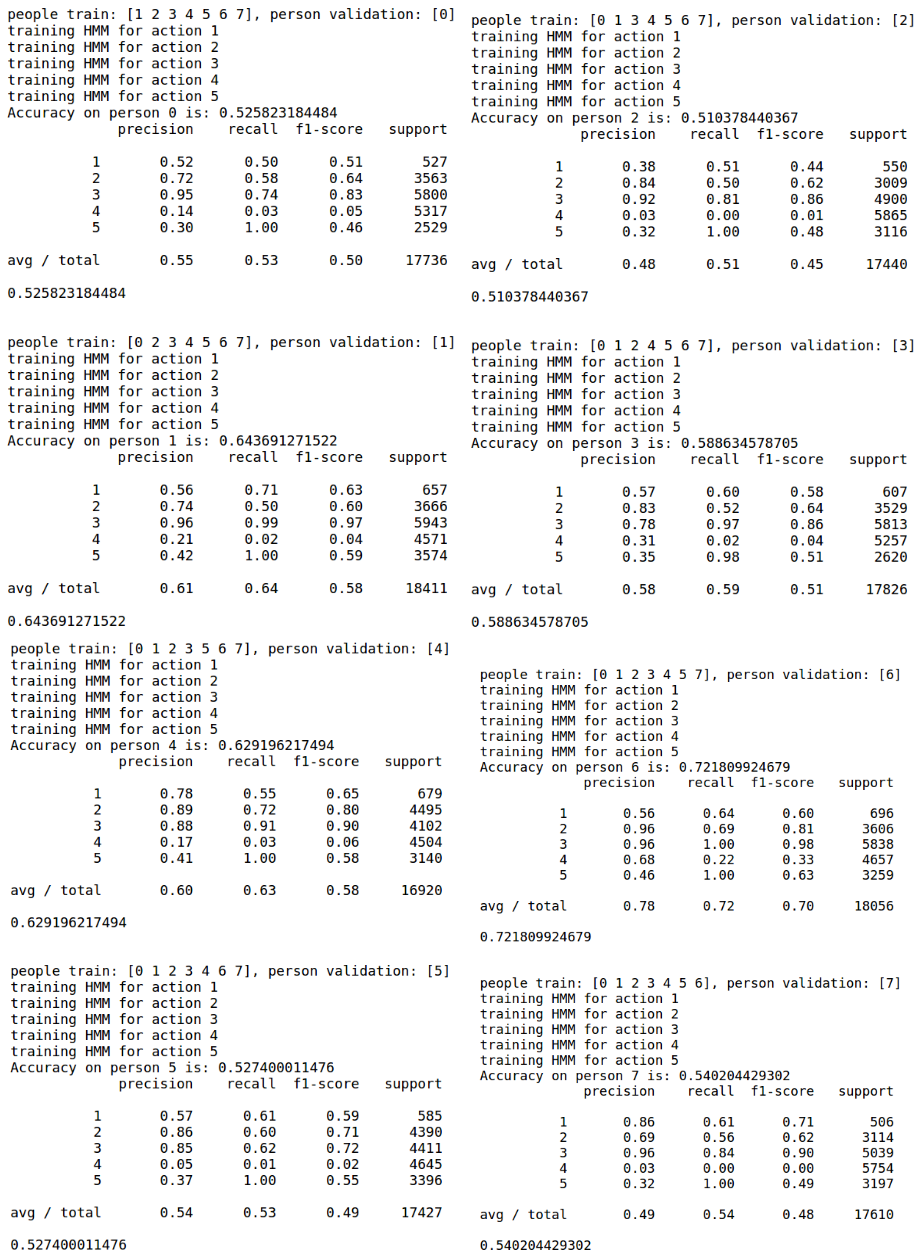
- 1) Running: 3 hidden states
- 2) Walking: 9 hidden states
- 3) Standing: 2 hidden states
- 4) Sitting: 2 hidden states
- 5) Lying: 2 hidden states

Walking appears to be the most complex action in the group an it always requires the highest number of hidden-states. Initially I thought that running would have been the one asking for a complex representation of its signals. We cannot say that walking is more peculiar than running, because the latter is just an acceleration of the former and eventually enhances the characteristics. An intuitive answer I gave me is that it was the high pace frequency itself of the running activity that prevail over the other peculiarities and allows for a "simpler translation" (even just 3 hidden-states when I lower the threshold for the maximum) of the signals. Instead for walking we have to rely on several hidden-states to represent the complexity.
I see the results concerning standing as controversial: letting possible to choose an high number of hidden-states, the LOO cross-validation returns 17 as best option, but it seems not reasonable that 'standing' needs a complex representation as walking. Indeed, if I look at the other stationary activities in the group (sitting and lying), just two hidden states, the minimum, are enough for the representation in both the case, with or without limiting the maximum threshold. Indeed, when the max number of possible hidden-states, standing is simplified to 2.
At this point it is worth to compare how the models yielded by the two approaches work.
For each action, I extract observation belonging to it from the whole train dataset and, on them, I fit an HMM, with the optimal number of hidden-states, again gaussian emission and diagonal covariance matrix. Finally I have 5 models. To test accuracy in a rapid way, I try to predict the labels of the same train dataset the 5 models have been trained on. I recall that the prediction is performed in this way: for the single observations I compute the log-likelihoods for each of the 5 HMMs and I assign the label according to the model that returned the maximum score.
Note: This is not a valid approach to test the accuracy; the observations used for training are **the same** I try to predict. Exactly because of that I expected an accuracy strictly near to 1, but...

```
Using optimal hidden-states yielded by the LOOcv with max number = 20
Walking 17, Running 14, Standing 17, Sitting 2, Lying 2
Accuracy on person 1 is: 0.904544429409
Accuracy on person 2 is: 0.910596925751
Accuracy on person 3 is: 0.867431192661
Accuracy on person 4 is: 0.845450465612
Accuracy on person 5 is: 0.884869976359
Accuracy on person 6 is: 0.643828541918
Accuracy on person 7 is: 0.919361984936
Accuracy on person 8 is: 0.932481544577

Using optimal hidden-states yielded by the LOOcv with max number = 10
Walking 9, Running 3, Standing 2, Sitting 2, Lying 2
Accuracy on person 1 is: 0.864738385205
Accuracy on person 2 is: 0.832763022106
Accuracy on person 3 is: 0.839793577982
Accuracy on person 4 is: 0.784864804219
Accuracy on person 5 is: 0.80780141844
```

Accuracy on person 6 is: 0.757732254548
Accuracy on person 7 is: 0.844096145326
Accuracy on person 8 is: 0.872231686542

I hardly obtain a score above 92%. The first case is extreme, the models completely fail to predict person 6 and this is an important indicator. Recall that before I was concerned that in the records we can encounter individuals with peculiar difference from others. From this point of view it is risky to use high number of hidden-states, because they capture a more precise shape for the action and they could fail for 'weird' inividuals as person 6. I have justified the use of LOO cross-validation to take into account this when choosing the optimal number of hidden-states; despite that my models fail. I thought it was a signal for my LOOcv not working properly but, even allowing for a retrained maximum number of h-s, person 6 is the one with the lowest accuracy achieved. It is still worht to say that the models with a lower overhall number of hidden-states yield generalized representations, that loose accuracy on most people, but exactly thanks to the more 'approximate' view, they do better with odd individuals. Another advantage I have to declare for working with a limited maximum number of hidden-states is that the time for fitting are reduced and I can allow for a larger number of iteration during the training of the model.
But now it is time to check how the 5 HMM predict the single activities for each Left-out-person. I report the results for the ensemble of HMMs with respectively 1:3, 2:9, 3:2, 4:2, 5:2 hidden-states. This model is faster in training and just slightly different in accuracy from the one with a higher number of hidden-states for the actions running and walking.



```
people train: [1 2 3 4 5 6 7], person validation: [0]
training HMM for action 1
training HMM for action 2
training HMM for action 3
training HMM for action 4
training HMM for action 5
Accuracy on person 0 is: 0.525823184484
             precision    recall  f1-score   support

          1       0.52      0.50      0.51       527
          2       0.72      0.58      0.64      3563
          3       0.95      0.74      0.83      5800
          4       0.14      0.03      0.05      5317
          5       0.30      1.00      0.46      2529

avg / total       0.55      0.53      0.50     17736

0.525823184484
```

```
people train: [0 1 3 4 5 6 7], person validation: [2]
training HMM for action 1
training HMM for action 2
training HMM for action 3
training HMM for action 4
training HMM for action 5
Accuracy on person 2 is: 0.510378440367
             precision    recall  f1-score   support

          1       0.38      0.51      0.44       550
          2       0.84      0.50      0.62      3009
          3       0.92      0.81      0.86      4900
          4       0.03      0.00      0.01      5865
          5       0.32      1.00      0.48      3116

avg / total       0.48      0.51      0.45     17440

0.510378440367
```

```
people train: [0 2 3 4 5 6 7], person validation: [1]
training HMM for action 1
training HMM for action 2
training HMM for action 3
training HMM for action 4
training HMM for action 5
Accuracy on person 1 is: 0.643691271522
             precision    recall  f1-score   support

          1       0.56      0.71      0.63       657
          2       0.74      0.50      0.60      3666
          3       0.96      0.99      0.97      5943
          4       0.21      0.02      0.04      4571
          5       0.42      1.00      0.59      3574

avg / total       0.61      0.64      0.58     18411

0.643691271522
```

```
people train: [0 1 2 4 5 6 7], person validation: [3]
training HMM for action 1
training HMM for action 2
training HMM for action 3
training HMM for action 4
training HMM for action 5
Accuracy on person 3 is: 0.588634578705
             precision    recall  f1-score   support

          1       0.57      0.60      0.58       607
          2       0.83      0.52      0.64      3529
          3       0.78      0.97      0.86      5813
          4       0.31      0.02      0.04      5257
          5       0.35      0.98      0.51      2620

avg / total       0.58      0.59      0.51     17826

0.588634578705
```

```
people train: [0 1 2 3 5 6 7], person validation: [4]
training HMM for action 1
training HMM for action 2
training HMM for action 3
training HMM for action 4
training HMM for action 5
Accuracy on person 4 is: 0.629196217494
             precision    recall  f1-score   support

          1       0.78      0.55      0.65       679
          2       0.89      0.72      0.80      4495
          3       0.88      0.91      0.90      4102
          4       0.17      0.03      0.06      4504
          5       0.41      1.00      0.58      3140

avg / total       0.60      0.63      0.58     16920

0.629196217494
```

```
people train: [0 1 2 3 4 5 7], person validation: [6]
training HMM for action 1
training HMM for action 2
training HMM for action 3
training HMM for action 4
training HMM for action 5
Accuracy on person 6 is: 0.721809924679
             precision    recall  f1-score   support

          1       0.56      0.64      0.60       696
          2       0.96      0.69      0.81      3606
          3       0.96      1.00      0.98      5838
          4       0.68      0.22      0.33      4657
          5       0.46      1.00      0.63      3259

avg / total       0.78      0.72      0.70     18056

0.721809924679
```

```
people train: [0 1 2 3 4 6 7], person validation: [5]
training HMM for action 1
training HMM for action 2
training HMM for action 3
training HMM for action 4
training HMM for action 5
Accuracy on person 5 is: 0.527400011476
             precision    recall  f1-score   support

          1       0.57      0.61      0.59       585
          2       0.86      0.60      0.71      4390
          3       0.85      0.62      0.72      4411
          4       0.05      0.01      0.02      4645
          5       0.37      1.00      0.55      3396

avg / total       0.54      0.53      0.49     17427

0.527400011476
```

```
people train: [0 1 2 3 4 5 6], person validation: [7]
training HMM for action 1
training HMM for action 2
training HMM for action 3
training HMM for action 4
training HMM for action 5
Accuracy on person 7 is: 0.540204429302
             precision    recall  f1-score   support

          1       0.86      0.61      0.71       506
          2       0.69      0.56      0.62      3114
          3       0.96      0.84      0.90      5039
          4       0.03      0.00      0.00      5754
          5       0.32      1.00      0.49      3197

avg / total       0.49      0.54      0.48     17610

0.540204429302
```

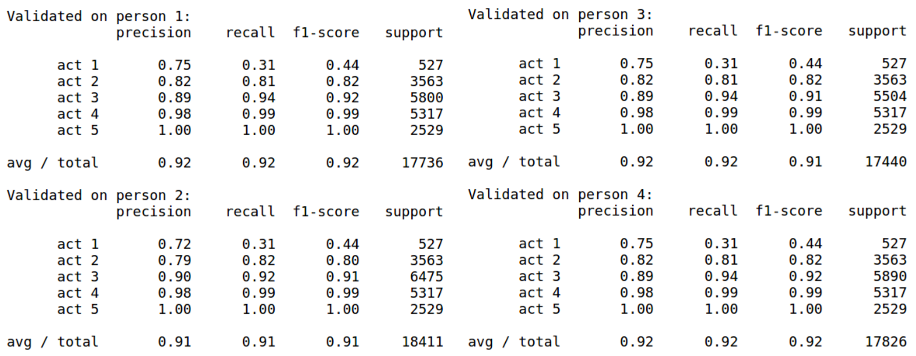
Figure 4: Classification accuracy of the HMMs throughout the stepsof the Leave-onePerson-out cross-validation

In the ipython notebook the reader can find all the accuracy scores for all the models but the conclusion is sadly the same: if we look in depth HMMs always fail in recognizing an activity. I suppose that this is due to the limited size of the dataset and i proceed with trying other approaches.

3 Other Models

3.1 Support Vector Machines

This kind of classificator could prove really effective when the data are condensed as in our situation: a kernel function maps the data into higher-dimensional spaces, until an hyperplane is found to correctely segregate them. But there are also some problems in practice application. The first is that here the classes we need to separate are not two, but five. The second trouble is that we shall select the best hyperparameters, which are both the parameters of the chosen kernel function (as gamma and coef0) and the kernel function itself (polynomial, radial basis function, sigmoid); to do this we need a grid search cross-validation and in computational terms it is very expensive.
I want to recall that, given that the kernel methods are based on distance, a normalization of the input data is always suggested before to begin the iterations, so that features appear in comparable ranges, otherwise the ones with the largest range will completely dominate in the computation of the kernel matrix.
The problem with this dataset is the time necessary for training the algorithm, considered the number of iterations in a grid-search I have thought to perform it on a single person's records, and hope that later the tuned hyper-parameter could perform well on the entire dataset. Moreover I have directly selected the rbf kernel, which in general is the most reasonable choice according to Hsu, Chang and Lin[2], and tune just its hyperparameters.



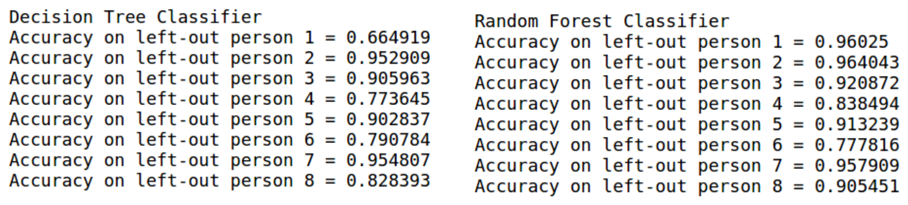
```
Validated on person 1:                                        Validated on person 3:
             precision    recall  f1-score   support                       precision    recall  f1-score   support

      act 1       0.75      0.31      0.44       527                act 1       0.75      0.31      0.44       527
      act 2       0.82      0.81      0.82      3563                act 2       0.82      0.81      0.82      3563
      act 3       0.89      0.94      0.92      5800                act 3       0.89      0.94      0.91      5504
      act 4       0.98      0.99      0.99      5317                act 4       0.98      0.99      0.99      5317
      act 5       1.00      1.00      1.00      2529                act 5       1.00      1.00      1.00      2529

avg / total       0.92      0.92      0.92     17736          avg / total       0.92      0.92      0.91     17440

Validated on person 2:                                        Validated on person 4:
             precision    recall  f1-score   support                       precision    recall  f1-score   support

      act 1       0.72      0.31      0.44       527                act 1       0.75      0.31      0.44       527
      act 2       0.79      0.82      0.80      3563                act 2       0.82      0.81      0.82      3563
      act 3       0.90      0.92      0.91      6475                act 3       0.89      0.94      0.92      5890
      act 4       0.98      0.99      0.99      5317                act 4       0.98      0.99      0.99      5317
      act 5       1.00      1.00      1.00      2529                act 5       1.00      1.00      1.00      2529

avg / total       0.91      0.91      0.91     18411          avg / total       0.92      0.92      0.92     17826
```

Figure 5: the scores for the other 4 stepst in the LOOcv can be found in the attached ipynb 'HAR sklearn'

[2]Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, "A Practical Guide to Support Vector Classification", National Taiwan University, Taipei 106, Taiwan, May 2016

3.2 Decision Tree and Random Forest Classifiers

Apache Spark is really quick in manipulating data and I just tried to see how the decision tree algorithm and the random forest ensemble perform with these data. I just plot the overhall accuracy obtained on the left-out person, to verify that it never outperforms Support Vector Machines.



```
Decision Tree Classifier
Accuracy on left-out person 1 = 0.664919
Accuracy on left-out person 2 = 0.952909
Accuracy on left-out person 3 = 0.905963
Accuracy on left-out person 4 = 0.773645
Accuracy on left-out person 5 = 0.902837
Accuracy on left-out person 6 = 0.790784
Accuracy on left-out person 7 = 0.954807
Accuracy on left-out person 8 = 0.828393
```

```
Random Forest Classifier
Accuracy on left-out person 1 = 0.96025
Accuracy on left-out person 2 = 0.964043
Accuracy on left-out person 3 = 0.920872
Accuracy on left-out person 4 = 0.838494
Accuracy on left-out person 5 = 0.913239
Accuracy on left-out person 6 = 0.777816
Accuracy on left-out person 7 = 0.957909
Accuracy on left-out person 8 = 0.905451
```

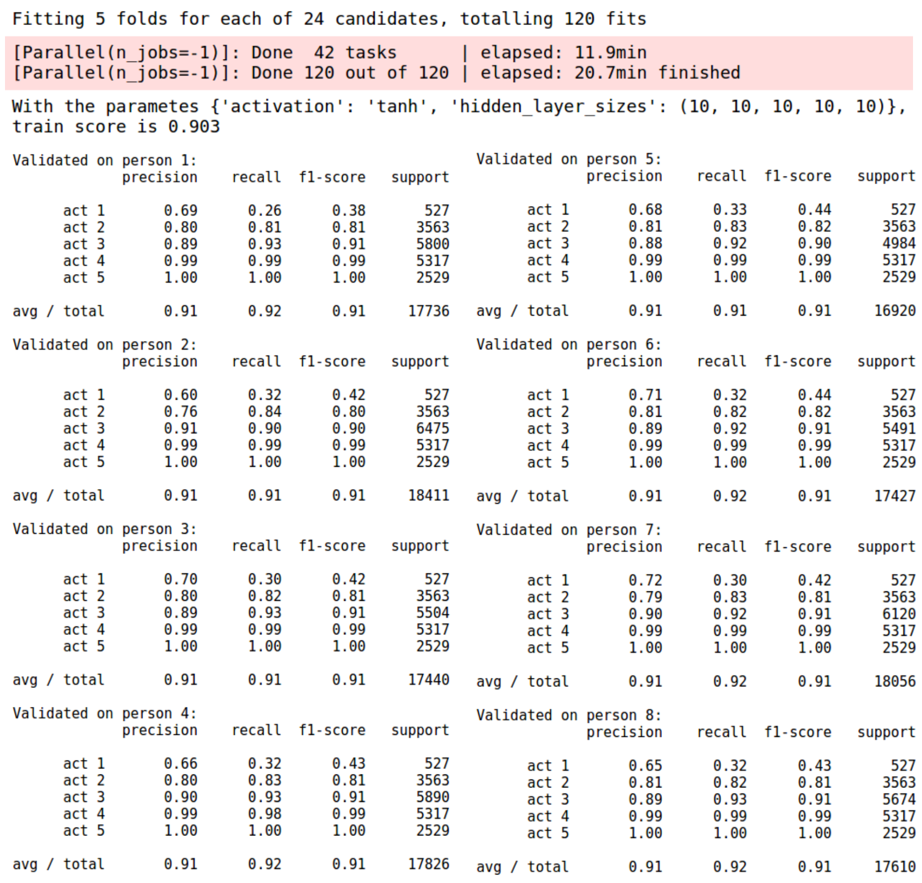
Figure 6: Find code in the attached ipynb 'HAR spark RForest'

I want to highlight that I have not had the possibility to previously tune hyper-parameters, neither for the decision tree, nor for the random forest. I could do it returning to python's scikit-learn and obtain enhanced results, but probably I would pay a toll in terms of over-fitting. I really like random forests but considering the previous observations over the high variety among people and problems with classifying odd human-beings, I had to admit that even the minimum risk of ovefitting has to be avoided and I abandoned decision trees.

3.3 Neural Networks: Multi-layer Percepton

Here we come to the most interesting model from my point of view. Until now I have proceeded on a pattern that avoided the feature extraction step suggested in the paper from Avci, Bosch, Marin-Perianu and Havinga. However, working with hidden-layers, could be thought as some kind of unsupervised alternative to this: part of the predictive capability of neural networks comes from the multi-layered structure of the networks that can learn to combine input features into higher-order ones.
The problem, as it was for SVM is the time necessary to train the neural-network, that determines great limitations also for hyper-parameter tuning. Naturally I have scaled (normalized) my data, like I did for the SVM, to converge before reaching the maximum number of iterations allowed. Nonetheless I have to select accurately the hyper-parameter I want to tune, otherwise I will end up running the grid-search for years; just think about the number of possible combinations for the hidden-layers. What I have finally chosen to try are: hidden-layers = [(5,),(50,50,50,50,50),(50,8),(50,25,5),(10,10,10,10,10),(100,100,100,100,100)]

activation function['logistic' = the logistic sigmoid function which returns f(x) = 1 / (1 + exp(-x)),
'tanh' = the hyperbolic tan function which returns f(x) = tanh(x),
'relu' = the rectified linear unit function which returns f(x) = max(0, x)]



```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

```
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed: 11.9min
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed: 20.7min finished
```

```
With the parametes {'activation': 'tanh', 'hidden_layer_sizes': (10, 10, 10, 10, 10)},
train score is 0.903
```

```
Validated on person 1:
             precision    recall  f1-score   support

      act 1       0.69      0.26      0.38       527
      act 2       0.80      0.81      0.81      3563
      act 3       0.89      0.93      0.91      5800
      act 4       0.99      0.99      0.99      5317
      act 5       1.00      1.00      1.00      2529

avg / total       0.91      0.92      0.91     17736

Validated on person 2:
             precision    recall  f1-score   support

      act 1       0.60      0.32      0.42       527
      act 2       0.76      0.84      0.80      3563
      act 3       0.91      0.90      0.90      6475
      act 4       0.99      0.99      0.99      5317
      act 5       1.00      1.00      1.00      2529

avg / total       0.91      0.91      0.91     18411

Validated on person 3:
             precision    recall  f1-score   support

      act 1       0.70      0.30      0.42       527
      act 2       0.80      0.82      0.81      3563
      act 3       0.89      0.93      0.91      5504
      act 4       0.99      0.99      0.99      5317
      act 5       1.00      1.00      1.00      2529

avg / total       0.91      0.91      0.91     17440

Validated on person 4:
             precision    recall  f1-score   support

      act 1       0.66      0.32      0.43       527
      act 2       0.80      0.83      0.81      3563
      act 3       0.90      0.93      0.91      5890
      act 4       0.99      0.98      0.99      5317
      act 5       1.00      1.00      1.00      2529

avg / total       0.91      0.92      0.91     17826
```

```
Validated on person 5:
             precision    recall  f1-score   support

      act 1       0.68      0.33      0.44       527
      act 2       0.81      0.83      0.82      3563
      act 3       0.88      0.92      0.90      4984
      act 4       0.99      0.99      0.99      5317
      act 5       1.00      1.00      1.00      2529

avg / total       0.91      0.91      0.91     16920

Validated on person 6:
             precision    recall  f1-score   support

      act 1       0.71      0.32      0.44       527
      act 2       0.81      0.82      0.82      3563
      act 3       0.89      0.92      0.91      5491
      act 4       0.99      0.99      0.99      5317
      act 5       1.00      1.00      1.00      2529

avg / total       0.91      0.92      0.91     17427

Validated on person 7:
             precision    recall  f1-score   support

      act 1       0.72      0.30      0.42       527
      act 2       0.79      0.83      0.81      3563
      act 3       0.90      0.92      0.91      6120
      act 4       0.99      0.99      0.99      5317
      act 5       1.00      1.00      1.00      2529

avg / total       0.91      0.92      0.91     18056

Validated on person 8:
             precision    recall  f1-score   support

      act 1       0.65      0.32      0.43       527
      act 2       0.81      0.82      0.81      3563
      act 3       0.89      0.93      0.91      5674
      act 4       0.99      0.99      0.99      5317
      act 5       1.00      1.00      1.00      2529

avg / total       0.91      0.92      0.91     17610
```

Figure 7: Find code in the attached ipynb 'HAR sklearn'

4 Conclusion

In conclusion Support Vector Machines are the model that yields the higher accuracy and does not fail for any activity; of course it has a lower precision for activity 1, but the success percentage is strong enough to rely on it.

```
test = mat['database_test']
test = map(lambda x : np.transpose(x[0]), test)
xTest1 = pd.DataFrame(test[0])
xTest2 = pd.DataFrame(test[1])

xTest1.columns = [1,2,3,4,5]
xTest2.columns = [1,2,3,4,5]

xTest1_scaled = preprocessing.scale(xTest1, axis=1)
xTest2_scaled = preprocessing.scale(xTest2, axis=1)
xTrain_scaled = preprocessing.scale(data[[1,2,3,4,5]], axis=1)


clf_rbf = svm.SVC(kernel='rbf',C=10, gamma=1)
model = clf_rbf.fit(xTrain_scaled, data['label'])

predictions1 = model.predict(xTest1_scaled)
predictions2 = model.predict(xTest2_scaled)

predictions = np.array([predictions1,predictions2])
scipy.io.savemat('SVM_testPredictions.mat',{'predictions':predictions})
```