Data Analytics for the Smart Society

Lab Exercise 3: Human Activity Recognition

Emilie Krutnes Engen
100356077

Master in Big Data Analytics
Carlos III University of Madrid

Universidad
Carlos III de Madrid

1 Introduction

Human Activity Recognition (HAR) is a research field that attempts to classify different movements based on sensor data. The aim of this exercise is to apply different HAR algorithms to classify a set of unlabeled data. Each observation in the dataset have two accelerometer variables and three gyroscope variables used by the classifiers to distinguish between five simple activities. The HAR algorithms investigated are Naïve Bayes, Support Vector Machines and Neural Networks. We first train the models with the pre-processed training data and evaluate their performance in terms of classification accuracy. Then, the models are applied to the unlabeled test data.

2 The Data

The dataset used for this exercise includes five predictors: transformed accelerometer data on the XY-axis and the Z-axis and transformed gyroscope data on the X-, Y- and Z-axis. The observations are given with a sampling frequency of 16 Hz. The data is divided into a training set and a test set. The training set consist of eight sequences, corresponding to eight different persons performing a set of activities. The activities are labeled with a number from 1-5, representing the activity classes: Running, Walking, Standing, Sitting and Lying. The test data includes two sequences of unlabeled data. Thus, the HAR algorithms are applied to predict the activities for all observations in this sample.

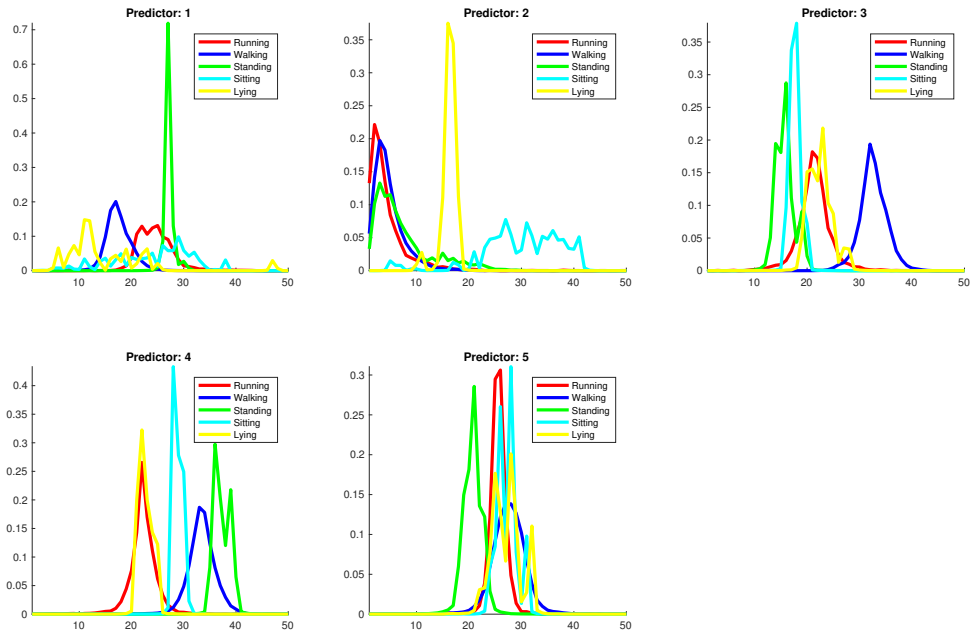

Figure 1: Distribution of the each predictor by activity class

Figure 1 show the distribution of the 5 predictors in the training data, where the colors represent the different activities. From the plots we observe that we by fitting a Gaussian to these distributions, should be able to classify the different activities with a decent performance level.

3 Activity Classification Models

Avci et al. (2010) propose several classification and recognition methods. Of these we have chosen to investigate two pattern recognition methods: Naïve Bayes (NB) and Support Vector Machines (SVM). In addition we also apply Neural Networks (NN) for the activity classification. According to the study performed by Avci et al. (2010), the research with Artificial Neural Networks have proven to give a high performance. The study compares the accuracy between the different classification methods. However the researches reviewed are conducted on different datasets. In this study we want to explore some of the methods applied to the same data in order to compare the performance of the classification procedures.

3.1 Naïve Bayes Classification

Naïve Bayes (NB) is a simple probabilistic classifier that allows for classification of unlabeled data based on past, labeled data. The supervised learning algorithm uses Bayes' theorem to calculate a probability measure based on previous data. Bayes theorem calculates the probability of the occurrence of an event A given the occurrence of an event B, $p(A|B)$, based on the knowledge of the probability of B given A, $p(B|A)$, and the probability of A, $p(A)$, and B, $p(B)$.

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \tag{1}$$

For this application we normally disregard the denominator, which gives us the following statement

$$p(A|B) \propto p(B|A)p(A) \tag{2}$$

Given n independent events B_i we get

$$p(A|B_1, B_2, \ldots, B_n) \propto p(B_1, B_2, \ldots, B_n)|A)p(A) = \prod_{i=1}^{n} p(B_i|A)p(A) \tag{3}$$

From this the classification is given by the following equation

$$y(\mathbf{x}) = \max_y p(y = C_j)p(\mathbf{x}|y = C_j) = \max_y p(y = C_j) \prod_{i_1}^{n} p(x_i|y = C_j) \tag{4}$$

where y denotes the predicted label and C_j denotes class j. This method is called "Naive" due to the assumption of independence between every pair of features. This assumption is in most cases not true. Despite this, Naïve Bayes have proven to perform well in many real-world situations. It require only a

small amount of training data to estimate the necessary parameters and have the advantage of a low computation time compared to more sophisticated methods.

3.2 Support Vector Machine

The Support Vector Machine (SVM) is originally a two-class classifier. To understand the procedure we consider the problem of classifying a set of training data into two classes with feature vector x_i and class label y_i. Assuming that the two classes can be seperated by a hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$ and that we have no prior knowledge of the data distribution, then the optimale hyperplane is the one that maximizes the margin $\mathbf{w}$. The optimal values for $\mathbf{w}$ and b is found by solving the constrained minimization problem, using Lagrange multipliers α_i (Schuldt et al., 2004).

$$f(\mathbf{x}) = \text{sign}\Big(\sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\Big) \tag{5}$$

The x_i values with nonzero α_i are the support vectors. α_i and b derived by applying a Support Vector Clustering (SVC) algorithm (Schuldt et al., 2004).

In order to build a multiclass classifier, we have to combine multiple two-class SVMs. For doing this, we use an approach called *one-versus-one*. This approach train $K(K-1)/2$ different 2-class SVMs on all possible pairs of classes, where K is the number of classes. The test set is then classified according to the class with the highest number of "votes" (Bishop, 2007).

3.3 Neural Network Classification

In a Neural Network (NN) we have several neurons, which are simple neural processing units. Given the input vector $\mathbf{x}$ a weighted linear combination of the input vector is presented to each neuron (Bishop, 2007).

$$v_j = \sum_{i=1}^{N} w_{ij} x_i - w_{0j} \tag{6}$$

where v_j denotes the j^{th} neuron, w_{ij} the weight from input i to neuron j and w_{0j} the bias term associated with the given neuron. The output class of each neuron is given by a non-linear activation function $\Phi()$ (Bishop, 2007).

$$y_j = \Phi(v_j) \tag{7}$$

The activation function used in this study is the logistic sigmoid $\sigma()$, given by

$$\sigma(\Delta) = \frac{1}{1 + e^{-\Delta}} \tag{8}$$

The Neural Network is constructed by linking the inputs and outputs of these neurons using the linkage method Multi-Layer Perceptron (MLP) (Bishop, 2007). Here several neurons are linked to the inputs in parallel, creating the hidden layer of the network. We have constructed two hidden layers, where the outputs of the neurons in the first layer works as inputs for the neurons in the second hidden layer. The outputs of the neurons in this layer are then given as inputs to a set of neurons in the output layer. The number of neurons in this layer correspond to the number of classes given by the classification problem.

4 Implementation

The classification methods applied in this study is implemented in Matlab. We first load the pre-processed dataset and retrieve the data and labels from the training set and the data from the test set. Then the cell arrays are converted to matrices. The Matlab code for these steps are given below.

```
% ------------------------------------------------------------------------
% Get training and test sets
% ------------------------------------------------------------------------

% Load preprocessed data
filename = 'HAR_database.mat';
load (filename);

% Data and labels
data = database_training(:,1);
labels = database_training(:,2);

% Convert cell arrays to matrices
data_matrix=[];
labels_matrix=[];
data_test=[];

for i=1:length(labels)
    data_T = data{i,1}.';
    data_matrix = [data_matrix;data_T];
    labels_T = labels{i,1}.';
    labels_matrix = [labels_matrix;labels_T];
end

for i=1:length(database_test)
    data_test_T = database_test{i,1}.';
    data_test = [data_test;data_test_T];
end

% ------------------------------------------------------------------------
```

We proceed by training the different classification models and evaluate their performance by computing the confusion matrices.

4.1 Naïve Bayes

We train a multiclass Naïve Bayes (NB) classification model by applying the Matlab function *fitcnb* that returns a multiclass NB model trained by the predictors and class labels in the training set. For evaluating the NB classifier we apply a 10-fold cross-validation. This separates the training data into 10 random partitions. For each iteration 9 partitions are used for training the model and the last partition is used for evaluating the performance. This process is repeat 10 times. The confusion matrix is constructed based on the predictions in all partitions. The Matlab code for implementing the NB classifier is presented below.

```
% ------------------------------------------------------------------------
% 1. Naive Bayes
% ------------------------------------------------------------------------

% Build Naive Bayes classifier
NB_model = fitcnb(data_matrix,labels_matrix);

% 10-fold CV
NB_model_CV = crossval(NB_model);

% Predict labels
NB_labels = kfoldPredict(NB_model_CV);

% Construct confusion matrix
NB_cm = confusionmat(labels_matrix,NB_labels);

% Inspect dataset
[n,p] = size(data_matrix);
is_labels = unique(labels_matrix);
n_labels = numel(is_labels);
tabulate(categorical(labels_matrix))

% Convert the integer label vector to a class-identifier matrix
[~,group_labels_NB] = ismember(NB_labels,is_labels);
CI_matrix_NB = zeros(n_labels,n);
idx_linear_NB = sub2ind([n_labels n],group_labels_NB,(1:n)');

% Flags the row corresponding to the class
CI_matrix_NB(idx_linear_NB) = 1;
[~,groups_NB] = ismember(labels_matrix,is_labels);
l_matrix_NB = zeros(n_labels,n);
idx_linear_label_NB = sub2ind([n_labels n],groups_NB,(1:n)');
l_matrix_NB(idx_linear_label_NB) = 1;

% Plot confusion matrix
figure;
plotconfusion(l_matrix_NB,CI_matrix_NB);

% ------------------------------------------------------------------------
```

4.2 Support Vector Machine

Similar to the Naïve Bayes classifier, the Support Vector Machine (SVM) classifier is also evaluated using a 10-fold cross-validation. We first construct a SVM template to standardize the predictors. The SVM classifier is then trained by applying the Matlab function *fitcecoc*, which returns a trained, multiclass error correcting output codes (ECOC) model using SVM binary learners. The function applies $K(K–1)/2$ binary SVM models using the one-versus-one coding design, where K is the number of unique class labels. We predict the activity classes using *kfoldPredict*. Based on these predictions the confusion matrix is calculated, including the out-of-sample classification error.

```
% ------------------------------------------------------------------------
% 2. SVM
% ------------------------------------------------------------------------

% Build SVM model
t = templateSVM('Standardize',true);
SVM_model = fitcecoc(data_matrix,labels_matrix,'Learners',t);

% 10-fold CV
SVM_model_CV = crossval(SVM_model);

% Predict labels
SVM_labels = kfoldPredict(SVM_model_CV);

% Confusion matrix
SVM_cm = confusionmat(labels_matrix,SVM_labels);

% Convert the integer label vector to a class-identifier matrix.
[~,group_labels_SVM] = ismember(SVM_labels,is_labels);
CI_matrix_SVM = zeros(n_labels,n);
idx_linear_SVM = sub2ind([n_labels n],group_labels_SVM,(1:n)');

% Flags the row corresponding to the class
CI_matrix_SVM(idx_linear_SVM) = 1;
[~,groups_SVM] = ismember(labels_matrix,is_labels);
l_matrix_SVM = zeros(n_labels,n);
idx_linear_label_SVM = sub2ind([n_labels n],groups_SVM,(1:n)');
l_matrix_SVM(idx_linear_label_SVM) = 1;

% Plot confusion matrix
figure;
plotconfusion(l_matrix_SVM,CI_matrix_SVM);

% ------------------------------------------------------------------------
```

4.3 Neural Networks

In order to train a Neural Network (NN) we first have to transform the labels to binary vectors. The vector size corresponds to the number of activity classes, where all elements are zero, except for the element of the given class which is one. The network is built using Matlab's Neural Network Toolkit. Different network architectures were tested experimentally. The chosen network is presented in Figure 2.

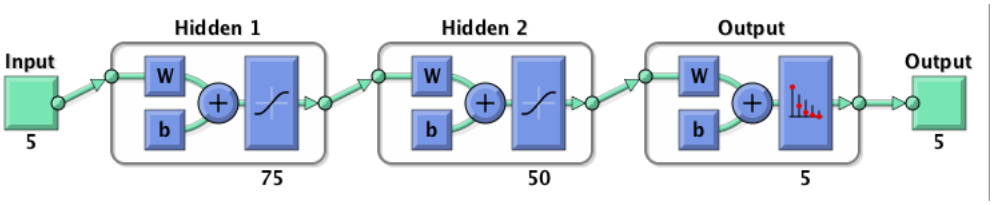

Figure 2: The NN architecture used with two hidden layers of 75 and 50 neurons, respectively

The number of inputs corresponds to the number of predictors in the given dataset. We create a pattern recognition network using the Matlab function *patternnet*. The network have two hidden layers, with 75 and 50 neurons in the first and second layer, respectively. Then the network is trained using the predictors and binary label vectors from the training set. The output is a five-dimensional binary vector with 1 for the element corresponding to the predicted activity class. The output vectors are then converted to class integers and the final confusion matrix is constructed. The Matlab code for the NN implementation is presented below.

```
% ------------------------------------------------------------------------
% 3. Neural Networks
% ------------------------------------------------------------------------
% Transform labels to binary vectors
labels_bin = zeros(size(labels_matrix,1),5);
for i = 1:size(data_matrix,1)
    if (labels_matrix(i) == 1)
        labels_bin(i,:) = [1 0 0 0 0];
    end
    if (labels_matrix(i) == 2)
        labels_bin(i,:) = [0 1 0 0 0];
    end
    if (labels_matrix(i) == 3)
        labels_bin(i,:) = [0 0 1 0 0];
    end
    if (labels_matrix(i) == 4)
        labels_bin(i,:) = [0 0 0 1 0];
    end
    if (labels_matrix(i) == 5)
        labels_bin(i,:) = [0 0 0 0 1];
    end
end

% Transpose data and labels
data_matrix_T = data_matrix.';
labels_bin_T = labels_bin.';

% Build Neural Network
NN_model = patternnet([75,50]);
```

```
view(NN_model)

% Train Neural Network model
[NN_model,tr] = train(NN_model,data_matrix_T,labels_bin_T);
view(NN_model)

% Predict labels
NN_labels = NN_model(data_matrix_T);

% Get classes
NN_classes = vec2ind(NN_labels);

% Confusion matrix
NN_cm = confusionmat(labels_matrix,NN_classes);

% Convert the integer label vector to a class-identifier matrix.
[~,group_labels_NN] = ismember(NN_classes.',is_labels);
CI_matrix_NN = zeros(n_labels,n);
idx_linear_NN = sub2ind([n_labels n],group_labels_NN,(1:n).');

% Flags the row corresponding to the class
CI_matrix_NN(idx_linear_NN) = 1;
[~,groups_NN] = ismember(labels_matrix,is_labels);
l_matrix_NN = zeros(n_labels,n);
idx_linear_label_NN = sub2ind([n_labels n],groups_NN,(1:n).');
l_matrix_NN(idx_linear_label_NN) = 1;

% Plot confusion matrix
figure;
plotconfusion(l_matrix_NN,CI_matrix_NN);

% ------------------------------------------------------------------------
```

Finally we predict the activity classes of the observations in the test set, using the trained models. The outputs labels for each method is converted to cell arrays and stored in a .mat file.

```
% --------------------------------------------------------------------
% 1. Naive Bayes
% --------------------------------------------------------------------
% Predict label for test set
NB_labels_test = predict(NB_model,data_test);

% Insert labels in cell array
NB_labels_test = NB_labels_test.';
len_test = length(database_test{1,1});
db_NB_labels_test = cell(2,[]);
db_NB_labels_test{1,1} = NB_labels_test(1:len_test);
db_NB_labels_test{2,1} = NB_labels_test(len_test+1:length(NB_labels_test));

% Save results
save('NB_results.mat','db_NB_labels_test')

% --------------------------------------------------------------------
% 2. SVM
% --------------------------------------------------------------------
% Predict label for test set
SVM_labels_test = predict(SVM_model,data_test);

% Insert labels in cell array
SVM_labels_test = SVM_labels_test.';
db_SVM_labels_test = cell(2,[]);
db_SVM_labels_test{1,1} = SVM_labels_test(1:len_test);
db_SVM_labels_test{2,1} = SVM_labels_test(len_test+1:length(SVM_labels_test));

% Save results
save('SVM_results.mat','db_SVM_labels_test')

% --------------------------------------------------------------------
% 3. Neural network
% --------------------------------------------------------------------
% Transpose test data
data_test_T = data_test.';

% Predict labels
NN_labels_test = NN_model(data_test_T);

% Get classes
NN_classes_test = vec2ind(NN_labels_test);

% Insert labels in cell array
db_NN_labels_test = cell(2,[]);
db_NN_labels_test{1,1} = NN_classes_test(1:len_test);
db_NN_labels_test{2,1} = NN_classes_test(len_test+1:length(NN_classes_test));

% Save results
save('NN_results.mat','db_NN_labels_test')

% --------------------------------------------------------------------
```

5 Results

The experimental results from the application of the HAR methods on the pre-processed data are presented below.

5.1 Naïve Bayes

The application of the Naïve Bayes classifier yield relatively good results, especially considering the simplicity of this procedure. From the confusion matrix in Figure 3 the diagonal elements show the number of cases that were correctly classified, and the off-diagonal elements show the misclassified cases. The blue cell in the bottom right corner shows the total percent of correctly classified cases and the total percent of misclassified cases, in green and red respectively. The bottom row show the percentage of correctly classified and misclassified observations for each target class. We observe that the classification accuracy vary over the different classes. Class 3 (Standing) yield the highes accuracy, with 98.3 % correctly classified observations. Class 1 (Running) is the most misclassified activity, with only 66.6 % correctly classified observations. Most misclassification for this class was due to incorrectly assigning observations of class "Running" to the class "Walking". We also observe that the size of Class 1 (Running) is relatively small, in terms of number of observations, compared to the other classes. When the number of observations present in training the model is low, correct predictions are difficult to obtain. However, the overall out-of-sample classification accuracy is 92.3 %, which indicates that the NB classifier generalizes fairly well. Because the accelerometer and gyroscope data was applied without conducting segmentation and feature extraction, the correlation between features in the data is limited. Thus, the simple NB classifier is able to obtain good results.

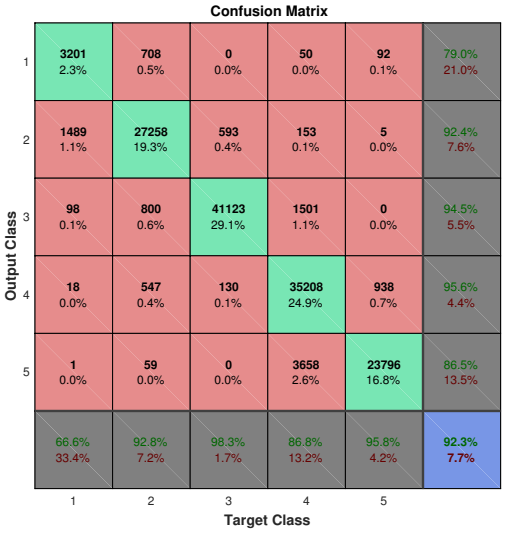

Figure 3: Sample confusion matrix from the NB classifier with the number of correctly classified observations on the diagonal and the number of misclassified observations off the diagonal, and the associated relative frequencies

5.2 Support Vector Machines

The results from application of the SVM classifier show poor performance, compared to the NB classifier. From the confusion matrix in Figure 4 we observe that the SVM classifier only yield an out-of-sample accuracy of 83.4 %. The low accuracy primarily comes from the lacking ability to correctly classify Class 1 (Running) and Class 2 (Walking). Most of the members of "Running" were incorrectly classified as "Walking" and a large part was also incorrectly classified as "Sitting" (Class 4). This result in an overall accuracy of only 4.9 % for the "Running" class. The misclassifications in the "Walking" class were classified as "Standing", leading to an out-of-sample accuracy of 68.9 % in this class.

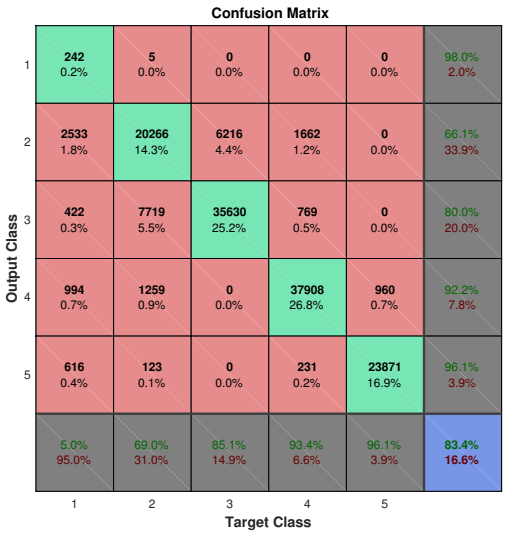

Figure 4: Sample confusion matrix from the SVM classifier with the number of correctly classified observations on the diagonal and the number of misclassified observations off the diagonal, and the associated relative frequencies

5.3 Neural Networks

The results from the application of the Neural Network classifier show a significantly higher classification accuracies than the pattern recognition techniques presented above. In Figure 5 the training, validation, test and overall confusion matrices are presented. We observe that the Neural Network obtain an overall classification accuracy of 95.8 %. The training, validation and test partitions all obtain approximately the same accuracy.

The best performance comes from Class 5 (Lying), obtaining an accuracy of 95.5 %, 95.6 % and 95.6 % in the training, validation and test set, respectively. Similar to the NB and SVM classifiers, Class 1 (Running) show the most misclassifications. The overall performance of this class is 70.3 %, which is an improvement compared to the other classification algorithms.

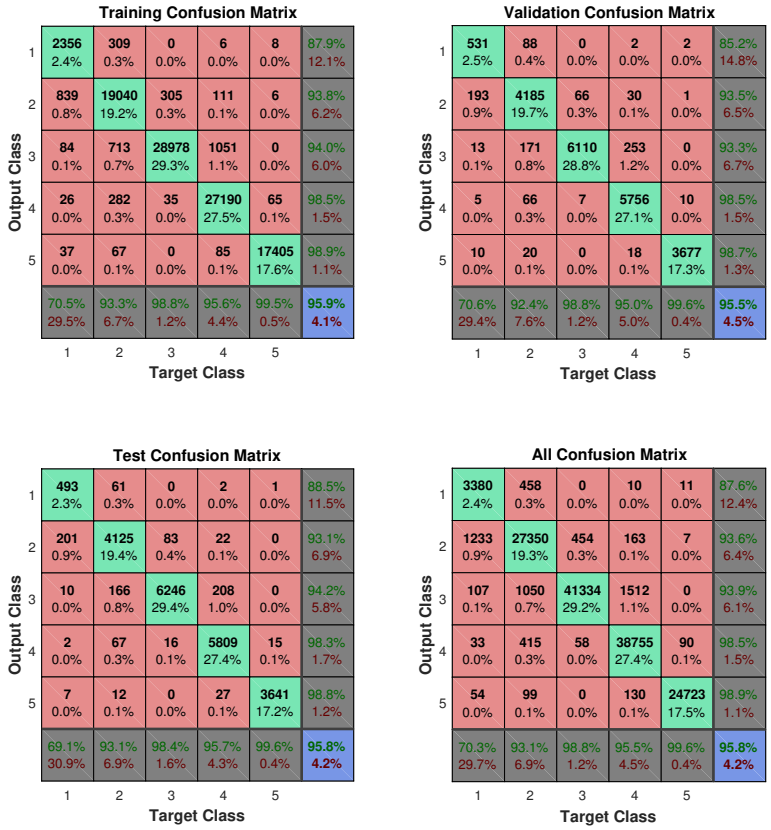

Figure 5: Sample confusion matrix from the NN classifier with the number of correctly classified observations on the diagonal and the number of misclassified observations off the diagonal, and the associated relative frequencies

The classification performance for the three states is presented in Figure 6. The plot show the cross-entropy error for the training (blue), validation (green) and test (red) sets over each epoch.

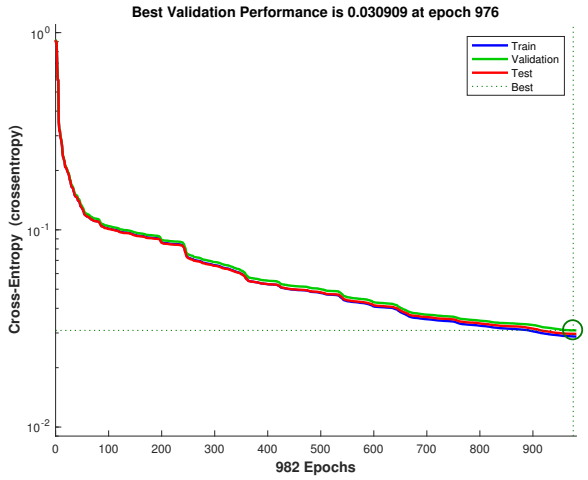

Figure 6: The network performance for the training, validation and test set in terms of the cross-entropy error over each epoch

The plot in Figure 7 show the training state, including the gradient plot and the validation check plot. The first show the gradient of the back-propagation algorithm updating the network weights. When the gradient gets smaller, the size of the updates of weights become less and less. The validation plot shows the number of successful validation checks in a row. A successful validation is given when the cross-entropy error in the validation set does not change significantly. When the system obtain 6 successful validation checks in a row, the network training stops.

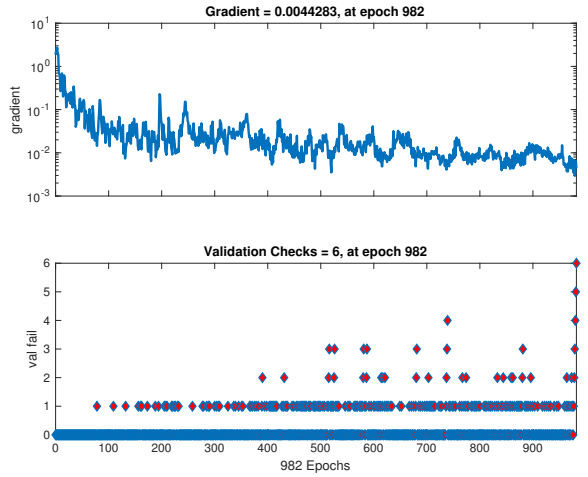

Figure 7: The training state showing the gradient and validation check plots for each epoch

Each time a neural network is trained a different solution is obtained, due to different initial weight and bias values and different divisions of data into training, validation, and test sets. As a result, different

neural networks trained on the same problem can give different outputs for the same input. Because of this, a single training of a Neural Network may not be a valid metric. In this study the Neural Network is only trained once. For further studies of classification using Neural Networks the training phase should be repeated several times. Calculating the cross-entropy performance, the mean and standard deviation across the trials will increase the validity of the performance metric obtained.

6 Discussion and conclusion

In this exercise methods for human activity classification have been presented. Mixed results were obtained for the SVM classifier, which also suffered from slow computation. Naïve Bayes showed relatively good performance, due to the lack of correlation in the features. The Neural Network methodology achieved significantly higher accuracies, with an overall accuracy of 95.6 %. This is consistent with the findings from previous research presented in Avci et al. (2010). Although these studies are not performed on the same datasets and apply different feature extraction techniques. All methods showed a high classification error for Class 1 (Running). Including more observations for this class, would most likely increase the overall accuracy of all classification methods.

For further research, a more sophisticated evaluation scheme is recommended. In this study 10-fold cross-validation was conducted. Alternatively, different leave-one-out (LOO) evaluation method can be applied. For instance leave-one-actor-out, leave-one-activity-out or leave-one-observation-out. In the Neural Network classification we experimented with different number of hidden states. Investigating several other combinations of hidden states for the specific classification problem is advised.

For application on more complex data, different techniques for feature extraction should be elaborated. Including more sensor data and developing a more sophisticated pre-processing phase, may be necessary to obtain high classification accuracy in these cases. It is reasonable to believe that pre-processing and feature extraction techniques will vary from case to case. However, the classification methods presented in this study are fairly general and can therefore easily be applied to a wide range of classification problems.

References

Avci, A., Bosch, S., Marin-Perianu, M., Marin-Perianu, R. and Havinga, P. (2010). Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey, *Architecture of computing systems (ARCS), 2010 23rd international conference on*, VDE, pp. 1–10.

Bishop, C. (2007). Pattern recognition and machine learning (information science and statistics), 1st edn. 2006. corr. 2nd printing edn, *Springer, New York* .

Schuldt, C., Laptev, I. and Caputo, B. (2004). Recognizing human actions: A local svm approach, *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, Vol. 3, IEEE, pp. 32–36.