Human Activity Recognition Using Naive-Bayes Classifier and Neural Networks

Teal Hobson-Lowther
Colorado School of Mines
December 14, 2015

I. Introduction

Human Activity Recognition (HAR) is an exciting field of research that attempts to classify human movement from sensor readings. Sensor types vary widely, but tend to be split between two groups: analog sensors and images. This paper will focus on the use of analog sensors primarily. The goal of any useful HAR algorithm is to utilize a small sampling of human activity to generate an automatic classification structure that can determine the class of new, unlabeled data. HAR is useful to any field where a human's activities might need to be tracked or monitored: health care, video games, and sports to name a few.

In health care, applications range from monitoring of the elderly [1], to gathering statistics on the quality of a recovering patient's activities. As an example, consider a study that wishes to determine the effects of house-arrest on a group of subjects that are on parole. We want to know how often each subject is performing a certain household task. If we have several subjects, the hand-labeling of each activity type around the house would require the employment of humans to perform a highly monotonous task. HAR algorithms for activity classification take the need for repeated human input out of the experiment. A HAR algorithm is trained on a small sampling of labeled activities, and then allowed to classify the remaining samples automatically. An additional bonus of these algorithms is that they can be trained off-line, and then classify new samples on-line. This means that classification can be performed immediately, automatically, and without the need for human input after the training phase.

A common method of data acquisition for video games consists of placing several sensors on different joints of the human body. The movements of the human are translated into the computer's coordinate system and used as skeletons for the movement of 3D models. The implementation of HAR could be useful to the field in the following way: assume we want to collect several different samples of a human model sitting down and standing up. Data acquisition would consist of getting the model to sit down/stand up several times, and a human would label those movements. Then, a HAR classification algorithm could be developed to determine features that indicate sitting/standing, and the model could continue repeating the sitting/standing actions. These new actions are automatically classified, reducing the need for human input in the collection process.

HAR for sporting activities has been investigated by several researchers [7],[9],[10]. In general, the goal of these HAR techniques involves discerning statistics about the *quality* of repetitive actions. For example, a common sports application for HAR would involve getting a collection of metrics on decidedly "well-performed" bicep-curls. A training athlete could perform the action in their own way, and compare their metrics to those of the "well-performed", to determine ways in which they fall short. Analysis of these differences could result in more precise exercise goals for the athlete. Imagine, now, that the athlete has several different tasks which meed improvement. The metrics on all exercises could now be generated automatically by a HAR classification algorithm, while the athlete performs (online), rather than needing to be analyzed by a human user.

This paper aims to accomplish two goals. First, we hope to provide a thorough recap of previous research done in the area of HAR. A detailed description of a select sampling of previous works in the field will be found in Section II. Different collection methods and results are compared/contrasted there. Many researchers have found successful classification, but methodology tends to vary wildly from application to application. In particular, differing data collection and feature extraction techniques are employed throughout the literature. Our second goal is to properly implement classification algorithms on simple human activity (sitting, standing, walking, etc.), in the hopes that the methods derived herein may be applied to more complicated data (exercises, sporting activities, etc.). In particular, the data analyzed consists of in-home activity recognition, which might be used for the monitoring of the elderly or people on parole. There are five classes that are distinguished: sitting, sitting down, standing, standing up, and walking. The data is taken from 4 3-axis accelerometers placed on the right bicep, abdomen, left thigh, and right ankle. A more detailed description of the data, as well as collection methods, will be given in Section III. A description of the classification and pre-processing methods used in the paper are given in Section IV. This section includes the theory behind Naive-Bayes and Neural Network methods. Readers already familiar with these algorithms can skip these sections. Section IV also includes a detailed delineation of the pre-processing (feature extraction) methods employed in this paper, and is where the majority of the signal-processing can be found. Section V will give a description of the experiments performed, as well as a brief discussion of the results found. A discussion of the

paper's findings and trajectories of future work will be found in Section VI. The appendices contain the code used to generate these results.

II. Previous Work

Previous work in the field of HAR varies in many ways. In general, the scope and goals of the papers are vastly different across the board. Data acquisition methods, feature-extraction techniques, and classification algorithms used vary widely from paper to paper. Indeed, it seems that the lack of a somewhat ubiquitous benchmarking dataset is something that stands out when reviewing the literature. This lack of a benchmark dataset was noted by the authors of [1], who attempted to remedy this situation with a free, online, open-source HAR dataset. This is the data that I will be analyzing in this paper, in hopes to contribute to a more well-defined standard for algorithm comparison.

The feature-extraction methodology followed most closely comes from [1]. The paper focuses on activity recognition for the elderly, and contains a very well defined summary of human activity recognition papers. The table found in Figure 1 is a replication of a summary of previous work found in [1]. As we can see, the maximum classification accuracy obtained via a Naive-Bayes Classification method is 97.7%, and the maximum classification accuracy found using Neural Networks is 99.6%. Impressive performance was obtained using SVM and Decision Tree classification methods, as well. The data are described in detail in Section III. The Euler angles of the acceleration readings, the variance of each Euler angle, and a column that discretized the magnitude of acceleration vector were used as features. Again, data is publicly available, which is useful for comparison of classification effectiveness. An algorithm for determining influential feature-selection based on correlation was performed, and the link for that process presented. A C4.5 decision tree algorithm was used in connection with AdaBoost ensemble for classification.

Research	# of sensors	Accelerometers' position	Solution	# of users	Learning mode	Test mode	Correct (%)
Liu et. al. (2012) [6]	1	hip, wrist (no info about orientation)	SVM	50	Super-vised	leave-one-out	88.1
Yuting et al. (2011) [7]	3	chest and both thighs (no info about orientation)	Threshold-based	10	--	--	98.6
Sazonov et al. (2011) [8]	1	foot	SVM	9	Super-vised	4-fold cross validation	98.1
Reiss & Stricker (2011) [9]	3	lower arm, chest and foot	Boosted Deci-sion Tree	8	Super-vised	8-fold cross validation	90.7
Min et al., (2011) [10]	9	torso, arms and legs	Threshold-based	3	--	Comparison with k-means	96.6
Maekawa & Watanabe (2011) [11]	4	wrists of both hands, waist, and right thigh	HMM	40	Unsu-pervised	leave-one-out	98.4
Martin et al. (2011) [12]	2	hip, foot and chest	Threshold-based	5	--	--	89.4
Lei et al. (2011) [3]	4	waist, chest, thigh, and side of the body	Naive Bayes	8	Super-vised	Several, w/ no cross validation	97.7
Alvarez et al. (2011) [13]	1	centered in the back of the person	Genetic fuzzy finite state machine	1	Super-vised	leave-one-out	98.9
Jun-ki & Sung-Bae (2011) [14]	5	forehead, both arms, and both wrists	Naive Bayes and SVM	3	Super-vised	leave-one-out	99.4
Ioana-Iuliana & Rodica-Elena (2011) [15]	2	right part of the hip, lower part of the right leg	Neural Networks	4	Super-vised	66% trainning vs 33% test	99.6
Gjoreski et al. (2011) [2]	4	chest, waist, ankle and thigh	Naive Bayes, SVM, C4.5, Random Forest	11	Super-vised	Leave-one-person-out	90.0
Feng, Meiling, and Nan (2011) [16]	1	Waist	Threshold-based	20	--	--	94.1
Czabke, Marsch, and Lueth (2011) [17]	1	Trousers' Pocket	Threshold-based	10	--	--	90.0
Chernbumroong, et al. (2011) [18]	1	Non-dominant wrist (watch)	C4.5 and Neural Networks	7	Super-vised	5-fold cross-vali-dation	94.1
Bayati & Chavarriaga (2011) [19]	--	Simulations instead of real accelerometers	Expectation Maximization	--	Unsuper-vised	Not mentioned	86.9
Atallah et al (2011) [20]	7	ear, chest, arm, wrist, waist, knee, and ankle	Feature Selection algorithms*	11	Super-vised	Not applied	--
Andreu et al. (2011) [21]	1	Not mentioned	fuzzy rule-based	--	Online learning	--	71.4

* This work is about sensor positioning and feature extraction

Fig. 1: Table containing previous work in HAR (up to 2011) with different classification methods and performance metrics, taken from [1]. Top performances by NB and NN methods are 97.7% and 99.6%, resp.

Several methods of exercise classification are applied to a dataset consisting of lying, cycling, climbing, walking, running, sitting, and standing in [8]. Naive-Bayes (NB), Gaussian Mixture Models (GMM), Logistic Regression, Parzen, Support-Vector Machine (SVM), K-Nearest Neighbors Clustering (KNN), Artificial Neural Network (ANN), and C4.5 Decision Trees were compared to novel methods presented here; cHMM with two distinct phases had 99.1% classification accuracy, consistently better than GMM. There was a small amount of data, which made some training techniques difficult. Investigation of "spurious data" filtering increased the accuracy.

Discrete Wavelet Transform (DWT), SVM, and mixture feature extraction techniques are discussed at length and applied to smartphone data in [7]. Labeled activities from soccer and hockey sessions for several athletes are used as training data for classification algorithms: SVM, Naive Bayes, ANN, and Decision Trees. Several variations of the DWT pre-processing parameters are implemented and discussed. The F-measure is used for determination of model accuracy.

Human activity is classified based on 3-axis accelerometer data using 43 features in [9]. Explanation of the feature extraction is discussed at length. The authors apply several learning techniques to classify human movement into walking, jogging, ascending stairs, descending stairs, sitting, and standing. Techniques used are: Decision Trees, Logistic Regression, and a Multilayer Perceptron (MLP). Classification accuracies of up to 91.7 percent are obtained (MLP). Discussion at the end points to several useful papers on the topic.

Three clustering algorithms (K-Means, GMM, and average-linkage hierarchical aggomerative clustering (HIER)) are used to classify human activity in [10]. The data are taken from smart phone three-axis accelerometer readings. The feature selection involved taking a sliding window of 64 readings of 25 normalized features: 12 from the accelerometer and gyrometer readings, and 12 from the FFT's of the sensors, and normalizing feature values to [0,1]. This technique of extracting Fourier coefficients via FFT is borrowed from this

Subject	Genre	Age	Height	Weight	Instances
A	Female	46 y.o.	1.62m	67kg	51,577
B	Female	28 y.o.	1.58m	53kg	49,797
C	Male	31 y.o.	1.71m	83kg	51,098
D	Male	75 y.o.	1.67m	67kg	13,161*

* A smaller number of observed instances because of the participant's age

Fig. 2: Table containing data distribution by subject, taken from [1].

paper and applied below.

III. Data

The dataset used throughout this paper is made publicly available through the Groupware research collective, and distributed through their website, http://groupware.les.inf.puc-rio.br/har. The researchers used Arduino LilyPad 3-axis accelerometers in four locations on the body (right bicep, abdomen, left thigh, and right ankle) to generate 165,632 data samples (12 features each) of five activity classes: sitting down, standing up, walking, standing, and sitting. The locations of the sensors, as well as the hardware set-up used to collect these data are shown in Figure 4. Scatter plots showing the three-axis readings for each sensor, by class, are shown in Figure 3. The data are collected from a group of four users. The distribution of the data by subject, taken from [1], is shown in Table 2. Because the sensors are independent, this dataset could be used for studies that utilize only a certain subset of the sensor readings (eg. just a bicep monitor). However, all four sensor locations were used for classification in this project. Extensive pre-processing was performed on the data before classification, and the methodology used will be discussed in Section IV-C.

IV. Methods

This section describes the theory and application of Naive-Bayes Classifier (Section IV-A) and Neural Networks (Section IV-B) for classification problems. The techniques used for feature extraction prior to classification are found in Section IV-C.

A. Naive-Bayes Classification

1) Theory: In general, Naive-Bayes Classification (NB) is an extension of the Maximum Likelihood Estimation (MLE) that allows for classification of unlabeled (testing) data based on past, labeled (training) data.

It is called "Naive" because it assumes that all elements of the input data are independent, and the "Bayes" term is included because the NB classifier exploits Bayes' Rule to develop a probability measure based on an input and past data.

Bayes' Rule is the most basic equation of conditional probability. It allows the user to extract the probability of an event A occurring given an event B also occurred ($p(A|B)$), as long as the user knows the probability of the event B occurring given A has occurred ($p(B|A)$), the probability of A occurring

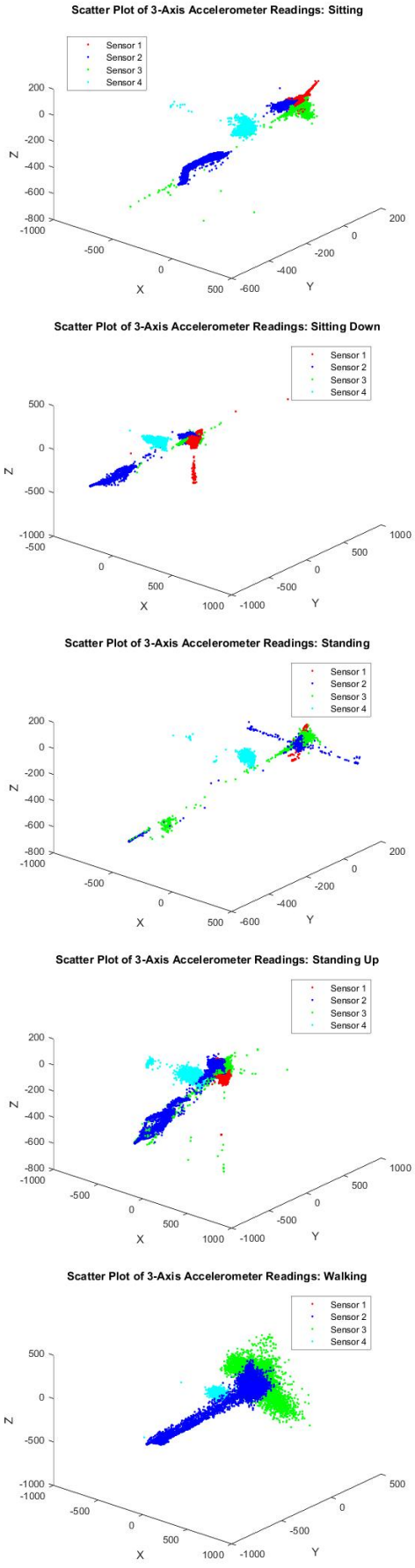

Fig. 3: 3D scatters of the three-axis accelerometer readings. Top to bottom: Sitting, Sitting Down, Standing, Standing Up, Walking. Red: Sensor 1, Blue: Sensor 2, Green: Sensor 3, Cyan: Sensor 4. Further description in Section III.

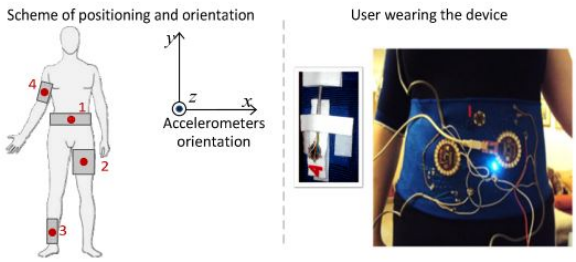

Fig. 4: Placement (left) and hardware implementation (right) used to generate the data set analyzed. There are four three-axis (x,y,z) accelerometers placed on the right bicep, abdomen, left thigh, and right ankle. An Arduino LilyPad setup was used to collect readings. [1]

($p(A)$), and the probability of event B occurring ($p(B)$). In equation form, it reads:

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

In general, we don't focus on the denominator and just make a simpler statement, that $p(A|B) \propto p(B|A)p(A)$

However, Equation 1 can be altered for n independent events B_i in the following way:

$$p(A|B_1, B_2, ..., B_n) \propto p(B_1, B_2, ..., B_n|A)p(A) \tag{2}$$

The Naive assumption that all events A, B_i are independent makes the math much simpler. In particular, independence of events B_i implies that $p(B_1, B_2, ..., B_n|A) = \prod_{i=1}^{n} p(B_i|A)$, and Equation 2 becomes:

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

Using this information for classification is not a complicated matter. Let C_j denote the jth class, and the elements of each input $\mathbf{x}$ be x_i. Then the classification is as follows:

$$\begin{aligned} y(\mathbf{x}) &= \underset{y}{\operatorname{argmax}}\, p(y = C_j)p(\mathbf{x}|y = C_j) \\ &= \underset{y}{\operatorname{argmax}}\, p(y = C_j) \prod_{i=1}^{n} p(x_i|y = C_j) \end{aligned} \tag{4}$$

2) Application: To perform Naive-Bayes Classification, we split the data into two groups: 80% of the data is used for training the classifier. The remaining 20% is reserved as testing data, to determine the quality of our classifier.

Say we have P sensor readings in a single class (with J classes), and let $\mathbf{x}_p$ denote the pth sensor reading from the training set. The elements of $\mathbf{x}_p$ are 12 accelerometer readings (3-axis for 4 accelerometers), denoted x_{pi}. We first collect all of the training points of a single class and put them into a "bag of words", so that we have a vector containing all readings of the training set for that class. Then, we count the frequency of each value and use it to construct the probability of each value in the class. This gives us $p(x_i|y = C_j)$ for each value x_i and each class C_j.

Now, we can determine whether a sensor reading belongs to class a or b by taking all the elements of the new reading, $xtest_i$, and calculating $\prod_{i=1}^{n} p(xtest_i|y = C_a)$ and $\prod_{i=1}^{n} p(xtest_i|y = C_b)$. If $\prod_{i=1}^{n} p(xtest_i|y = C_a) > \prod_{i=1}^{n} p(xtest_i|y = C_b)$, we say the new reading belongs to class a. If the converse is true, we say the new reading belongs to class b. The classification is readily extended to multiple class comparison, by simply taking the maximum $\operatorname{argmax}_j \prod_{i=1}^{n} p(xtest_i|y = C_j)$ over all classes being compared C_j.

B. Neural Network Classification

1) Theory: The theory of a basic feed-forward multi-layer perceptron, trained using back propagation, will now be discussed. The most simple neural processing unit is called a neuron. The neuron's name derives from its ability to mimic the activity of the most simple processing unit in the brain. A figure, showing the structure of a neuron, is shown in Figure 5.

The input to a neural network is a vector $\mathbf{x}$ of N input elements $x_i, i = 1, ..., N$. A weighted linear combination of these inputs is presented to each neuron, such that this combination, presented to the mth neuron (v_m), is given by:

$$v_m = \sum_{i=1}^{N} w_{im} x_i - w_{0m}$$

where w_{im} denotes the weight from input i to neuron m, and w_{0m} is a bias term associated with a neuron, and acts to center the neuron's response about a particular critical value.

The activity (output) of each neuron is the result of running the v_m through a non-linear *activation function* $\phi()$, such that the response of the mth neuron, y_m, is given by:

$$y_m = \phi(v_m).$$

The most common activation function by far is the logistic sigmoid, $\sigma()$, where

$$\sigma(\triangle) = \frac{1}{1 + \exp^{-\triangle}}.$$

This paper will be using the logistic sigmoid as the activation function for all neurons in the network.

The understanding of a single neuron's activity is the basis for the understanding of all of the network. Indeed, the network simply consists of different architectures linking the inputs/outputs of these neurons in interesting ways. The most common linkage method is that of the Multi-Layer Perceptron (MLP), shown in Figure 6. Several neurons are linked to the inputs in parallel, and form the *hidden layer* of the network. Then, the outputs of each neuron in the hidden layer are used as inputs to another set of parallel neurons in the *output layer*. The number of neurons in the output layer is equal to the size of the desired network output.

In the case of class prediction, the hidden layer can be any size, while the output layer is generally size Z, corresponding to the total number of classes. The inputs are accelerometer readings from the time series. For example,

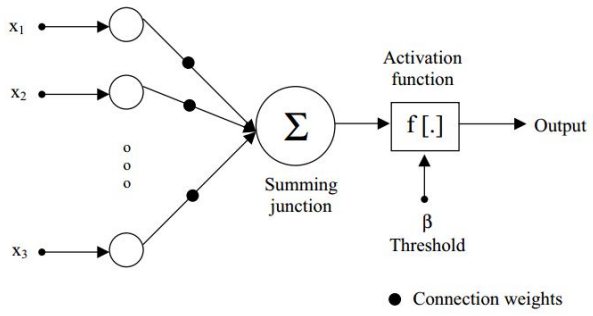

Fig. 5: Working of a single neuron [4]

a network with K accelerometer readings would consist of inputs $x_{1t}, x_{2t}, ..., x_{Kt}$, and the output of the network would be a Z dimensional binary vector with all zeros, but a "1" in the element that corresponds to the proper class (e.g. a five-class 0-4 problem with correct class 3 would be denoted $y_t = [00010]$).

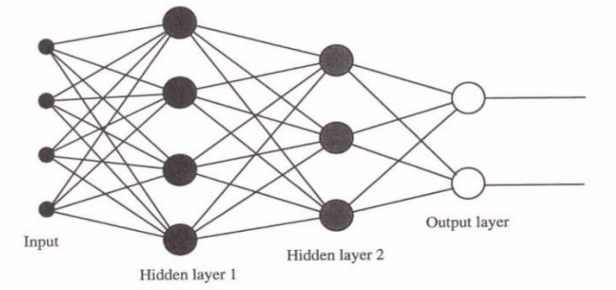

Fig. 6: Generic MLP architecture for binary classification [3]

The flexibility of the network comes from the way the weights are determined. First the weights connecting each element of the network are randomly initialized. Then, a series of input and output pairs are presented to the network. When a training sample pair is presented, the network attempts to minimize the error between its predicted output based solely on the input vector, and the actual known value of the output. The weights of the network are updated automatically based on the minimization of this error, and a new sample is presented. Training occurs until a specified lower bound for error is reached by the network. The method through which the weights update is known as back-propagation, discussed below. The true beauty of the network is that, through automatic processes, it determines structure *on its own*, and therefore no underlying knowledge of the system is required to get accurate predictions with a net. For this reason, many people regard NNs as a "black box" method that yields little to no insight into the actual problem. It gets results but the interpretability of the results is a bit vague.

Back-propagation is a way of transmitting the error of a network from the output back through the previous layers, and updating the weights such that an error is reduced. The performance of each trained neural network will be calculated via the cross-entropy (log-loss) metric, where we assume that $p \in [y, 1-y], q \in [\hat{y}, 1-\hat{y}]$:

$$H(p,q) = -\sum_i p_i log(q_i) = -ylog(\hat{y}) - (1-y)log(1-\hat{y})$$

where the total cross-entropy error is:

$$\begin{aligned} L(\mathbf{w}) &= \frac{1}{N}\sum_{n=1}^{N} H(p_n, q_n) \\ &= \frac{1}{N}\sum_{n=1}^{N} \left[y_n log\hat{y}_n + (1-y_n)log(1-\hat{y}_n)\right] \end{aligned}$$

for each of the N classes. Stochastic Gradient Descent is the most basic algorithm for performing back-propagation. Concise description of the idea can be found in [4] and [3].

2) Application: In this paper, the hyper-parameters of the networks used, (number of neurons in hidden layer, number of hidden layers) are found experimentally, by changing until a minimum cross-entropy error on the testing data set was obtained. The optimal configuration found through these experiments is shown in Figure 7. The programs were implemented using Matlab's Neural Network Toolkit, with custom coding to break away from the limitations of the Toolkit GUI. The GUI tends to push the user on a one-way track for different types of prediction/classification problems, and does not allow for much flexibility in the model. For example, the desire to have the network train via a different back-propagation algorithm is infeasible using the GUI.

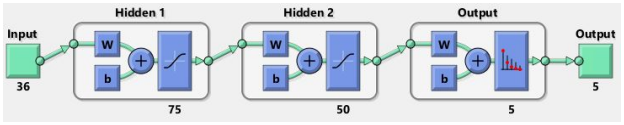

Fig. 7: Optimal NN architecture used throughout the paper, experimentally derived. The 36 inputs correspond to each of the features extracted from raw sensor data in Section IV-C.

C. Data Pre-Processing

The raw data contains 12 features: 3 (x,y,z) accelerometer readings from four different sensors, described in detail in Section III. The results of pre-processing the data is a vector of 36 features (9 for each sensor). The first three features are the pitch, roll, and magnitude of acceleration for each sensor, and are the only instantaneous features. The raw features are thrown out after this step, and the pitch, roll, and magnitude are normalized using the mat2gray() command in Matlab. The pitch (θ), roll (ϕ), and acceleration magnitude (α) are calculated as:

$$\theta_i = tan^{-1}\left(\frac{ay_i}{\sqrt{ax_i^2 + az_i^2}}\right) \quad (5)$$

$$\phi_i = tan^{-1}\left(-\frac{ax_i}{az_i}\right) \quad (6)$$

$$\alpha_i = \sqrt{ax_i^2 + ay_i^2 + az_i^2} \quad (7)$$

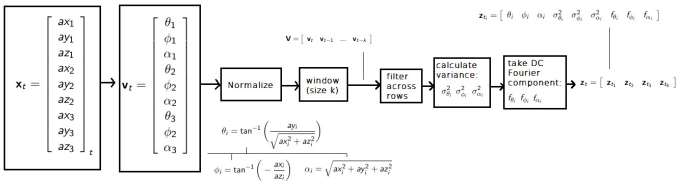

Fig. 8: Flow-chart for pre-processing algorithm, desribed in detail in Section IV-C.

where ax_i, ay_i, az_i are the x, y, z accelerometer readings for sensor $i = 1, 2, 3, 4$.

The next 6 features are extracted via a windowing method. For a given input data $\mathbf{x}_t$, a window of size k is calculated as $\mathbf{X}_t = \begin{bmatrix} \mathbf{x}_t, & \mathbf{x}_{t-1}, & \dots & , \mathbf{x}_{t-k} \end{bmatrix}$. This window is filtered using an averaging filter of size q, where

$$q = floor(k^{\frac{2}{3}}),$$

using the fspecial() and imfilter() functions in Matlab. Once the averaged window for a particular value of t has been obtained, the next 6 features for each sensor may be computed. Features 4-6 are the variances of the pitch, roll, and acceleration magnitude over the window, given by the usual unbiased variance calculation:

$$\sigma^2_{\beta i} = \frac{1}{n-1} \sum_{n=1}^{K} (x_{\beta ni} - \bar{x}_{\beta i})^2 \tag{8}$$

where β is either θ, ϕ, or α, $\bar{x}$ is the mean of the input feature over the window, and $i = 1, 2, 3, 4$ representing the four different sensors, and n is a counter that runs across all elements contained within the window. Features 7-9 are the fundamental (DC) component of the Fast Fourier Transform (FFT) over the window. A description of this algorithm, as well as how it is implemented using the fft() command in Matlab, can be found by typing "help fft" in Matlab's command line. The preceding steps give us all 9 features for each accelerometer, again resulting in a set of 36 features for each time sample. Pseudo-code for the pre-processing algorithm can be found in Algorithm 1, a flow-chart representing the process is found in Figure 8, and the full Matlab code is included in Appendix A. Figure 9 shows normalized histograms for each of the 9 features for Sensor 4. Each color corresponds to a different class. We can clearly see that simply fitting a Gaussian to these distributions would, in most cases, give us the ability to distinguish between classes (based on mean and std. deviation). That being said, some sensor readings allow for distinguishable classes for a given feature, while other sensors provide no insight to the naked eye.

V. Experiments/Results

Application of Naive Bayes and Neural Networks classifiers to the pre-processed data (described in Section IV-C) is delineated below (Sections V-A and V-B, resp.).

Algorithm 1 Pre-processing pseudo-code.

I) Raw data consists of four three-axis accelerometer readings (12 raw features)
II) Calculate the pitch roll, and L_2 norm of the accelerometer data for each sensor (12 features)
III) Throw out the raw data
IV) Normalize result of II
V) Window the data (take previous time samples of length window_size)
VI) Filter the windowed data with a filter_size length averaging filter
VII) Calculate the variance of window's normalized pitch, roll, and acceleration module data. (12 features)
VIII) Calculate the DC Fourier component of window's normalized accelerometer readings, pitch, and roll (12 features)

A. Naive-Bayes

To perform Naive Bayes classification on the extracted feature set, run the code in Appendix B. First, 80% of the shuffled data is used for training a NB classifier, NBModel, using fitcnb(). Then, the remaining 20% of data are labeled using predict() with our NBModel. A confusion matrix is generated using confusionmat(), and the total errors for a particular class is calculated as the ratio of the diagonal elements of the confusion matrix divided by the sum of the corresponding row. A sample confusion matrix for one of the 10 trials is shown in Table I. The average classification accuracies (over all trials) for each class are shown in Table II.

Applying the Naive-Bayes Classifier yielded mixed results. The classification accuracy varied widely over the different classes. "Sitting" yielded the highest accuracy, with a value relatively competitive with best results in the literature: 98.46%. "Standing Up" was the most misclassified activity, and compared dismally with the best results in literature. There were several instances where inputs that belonged to this class were determined by the network to belong to class "Standing." Also notable in terms of inaccuracy was "Sitting Down", with an overall accuracy of 67.7%. Most misclassification resulted in assigning the class "Standing" to those of "Sitting Down". The feature extraction methods presented in the paper may have led to a higher misclassification rate for the following reason: in the Naive-Bayes classifier, the separate inputs are considered to be independent and identically distributed (iid). The method of windowing the data based on previous time samples and finding statistics on these windows in effect *correlates* the data, thereby breaking independence and ensuring the conditions required for the Naive-Bayes classifier are not met. For example, taking the variance of a window obviously generates a correlation between a feature and its past values (variance doesn't exist for a single realization of a random variable). Results could be improved by applying a feature-extraction method that does not correlate data in time, and is more targeted at preserving the independence of data. That being said, the aims of the project were to find a classifier with optimal classification accuracy. This model certainly falls short

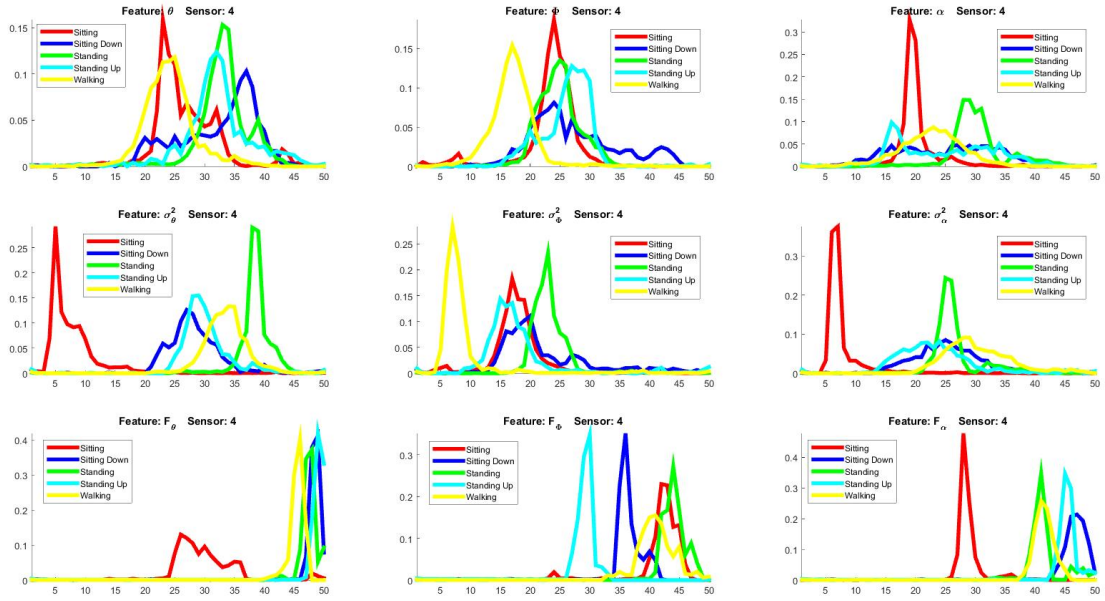

Fig. 9: Histograms of feautures, by class, for Sensor 4. First column, top to bottom: pitch (θ), variance of θ (σ^2_θ), and DC component of θ (f_θ). Second column, top to bottom: roll (ϕ), variance of ϕ (σ^2_ϕ), and DC component of ϕ (f_ϕ). Third column, top to bottom: magnitude of acceleration (α), variance of α (σ^2_α), and DC component of α (f_α). Description of pre-processing algorithm given in Section IV-C.

of the accuracy of the NNs (described below), but does present a couple of advantages. First, the model is very simple, and for this reason requires significantly less processing power and RAM than the more complex Neural Networks. Second, the training time for the NB classifier is *significantly* less than that of the NN, training in about 1.5 minutes, compared to the NNs 6.5 minutes.

True Class	**Predicted Class**				
	Sit	S. Down	Stand.	St. Up	Walk.
Sitting	9907	65	0	90	0
Sitting Down	29	1589	452	39	238
Standing	57	88	8678	211	598
Standing Up	67	286	835	1200	77
Walking	25	33	1367	84	7113

TABLE I: Sample confusion matrix, result of running NB classifier on pre-processed GroupwareHAR dataset. Diagonal elements are successful classifications, off-diagonal elements are misclassifications.

Class	**Accuracy**
Sitting	98.46%
Sitting Down	67.7%
Standing	90.1%
Standing Up	48.68%
Walking	82.5%

TABLE II: Overall NB classification accuracy for each activity.

B. Neural Network

The algorithm for generating a trained Neural Network for classification of the feature set found in Section IV-C is delineated in Appendix C. First, we run PreProcessing.m to generate the features. Then, the patternnet() command is used to create a Neural Network, "net", that has 75 neurons in the first hidden layer and 50 neurons in the second hidden layer. The train() function sets the input layer to the size of one time-sample of the input data (36 features), and the output layer to the size of one sample of the binary class data (5 output classes). The overall classification performance, network training state, and confusion matrix generated as a result of training the network are shown in Figure 10. The performance plot shows the amount of cross-entropy error for the training (blue), validation (green), and testing (red) sets. The training state is shown below the performance plot, and contains two plots. The first is the gradient of the backpropagation algorithm that updates network weights. As the gradient gets smaller, the weights get updated by less and less. The validation checks plot is right below the gradient plot, and shows the number of successful validation checks *in a row*. A successful validation check occurs when the cross-entropy error on the validation set does not change significantly. When the number of successive validation checks reaches 6, the network stops training. Moving right to left, top to bottom in the lower portion of Figure 10, we see the confusion matrices for the training data, validation data, testing data, and all

data, resp. Confusion matrices are read as follows: the correct class of an input vector is row i. The network outputs when presented with a input belonging to class i are the columns j. The elements [i,j=i] of the confusion matrix reflect correct outputs (j) to inputs known to be of class (i). The elements [i,j$\neq$ i] are incorrect network response (j) to an input known to be of class (i). The last element of each row [i] represents the total network response accuracy for class [i].

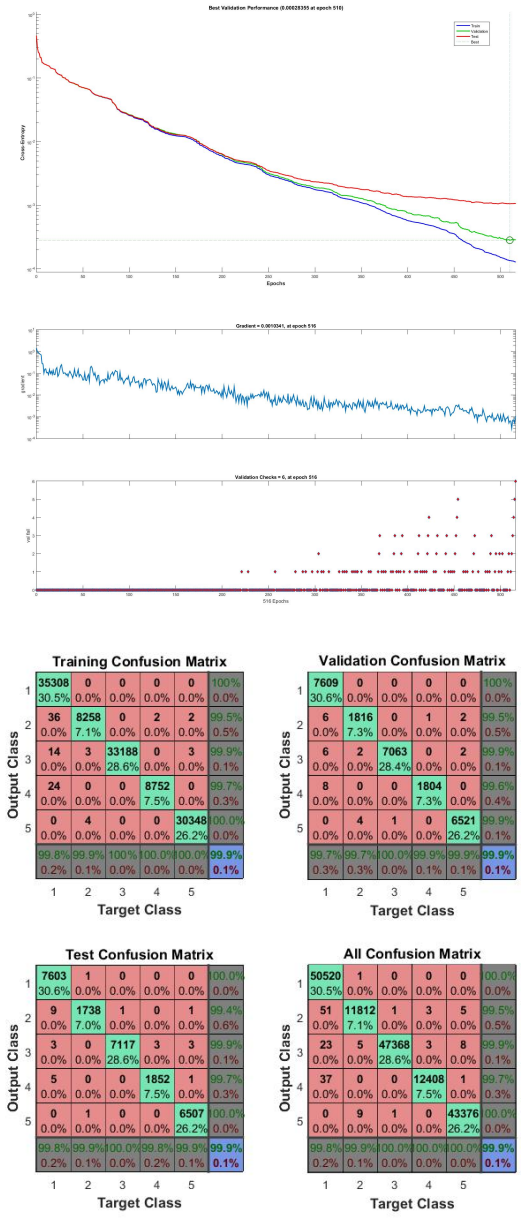

Fig. 10: **Top:** Overall network performance in terms of class cross-entropy over each training epoch. **Middle:** Training state showing the gradient and validation checks as the network progresses. **Bottom:** Confusion Matrix for the 75-50 NN classifier. Total classification accuracy is 99.9%.

The performance of the neural network varies from trial to trial because of the random initialization of weights, as well as the possible existence of local minima in the error function. Because of this, a single training of a neural network may not be used as a valid metric. Alternatively, we can train the neural network several times and look at the cross-entropy performance across multiple trials, as well as the mean and standard deviation of the performance. For this reason, 15 trials were run, and the mean classification accuracy was 99.91% over the entire dataset. The worst class prediction performance by the neural network is certainly that of class 2, or "Sitting Down." The total classification accuracy for the training/testing/validation data varied between 99.4-99.7%. The best performance of our classifier comes from the first class, "Sitting," with a classification accuracy of 100% across all trials. In fact, there were an average of 1 instance of "sitting" being misclassified over all 50,520 sitting samples. The total classification accuracy of the network is not only commercially viable on a sample-by-sample basis (99.9% classification accuracy is said to be required of commercially viable algorithms), but is also significantly higher than the accuracy achieved in [1] using decision trees.

A further discussion of the implications of a sample-by-sample classification is required here. The general metric for measuring classification accuracy of a machine learning algorithm is to look at how many testing samples were labeled properly. This is a great metric for situations such as image classification, where there is no time-series component to the data. Essentially, the network has one chance to successfully label the image. If it fails, there are no second chances. This is not so much the case when the data come from a time-series. The network that has been generated through the methods described in this paper has a *sample-by-sample* accuracy of 99.91%. This means that for 1000 input samples, there will be only 1 misclassified. Now imagine the application of this network to a series of sensor readings. Say the sensor collects data at a sample rate of $\beta \frac{samples}{s}$, and further assume that a particular action lasts T seconds. The number of samples collected for this action is $T\beta$. Each of these samples is classified with an accuracy of 99.91%, and therefore the majority of samples collected will be properly predicted. If we take the class output with the maximum class frequency for a particular sample period, the classifier will actually be more than 99.91% accurate over all actions performed. From this perspective, the classifier can become more powerful than the confusion matrix in Figure 10 tends to indicate. Regardless, the performance is adequate for most applications.

VI. Discussion

In this paper, methods for feature-extraction and automated classification of human activity data have been presented. Mixed results were obtained using the Naive-Bayes classifier, mostly because of the highly correlated features that were extracted from the data and used as input to the model. A neural network structure was proposed that achieved significantly higher classification accuracies, reaching a commercially viable level of 99.91%. This level of accuracy is on par

with the highest found in the current literature. To be fair, not every paper in the literature uses the same data. In fact, the methods for obtaining data are almost as disparate as the authors of the papers themselves. That being said, this paper significantly raised the bar on the current best performance standards on this benchmark dataset. Future work generated by this paper is essentially to apply the methods presented to more complicated (interesting) data. There are many situations in which the methods could be applied. For more complex data, however, more elaborate feature extraction techniques may need to be explored in order to obtain the levels of accuracy achieved here. It seems that the correct classification of a dataset is dependent on the application of preprocessing techniques that tend to be highly specified on a case-by-case basis. Finding the correct features to feed the classifier is, to some extent, an art. Hopefully some of the techniques for HAR presented in this paper can be generalized and applied to a wider variety of classification problems in the future.

REFERENCES

[1] Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements. Proceedings of 21st Brazilian Symposium on Artificial Intelligence. Advances in Artificial Intelligence - SBIA 2012. In: Lecture Notes in Computer Science. , pp. 52-61. Curitiba, PR: Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-34458-9. DOI: 10.1007/978-3-642-34459-6_6.

[2] H. Wang, 'Lecture 10: Review of Linear Algebra and Optimization', Colorado School of Mines, 2015.

[3] M. Mohandes, S. Rehman and T. Halawani, 'A neural networks approach for wind speed prediction', Renewable Energy, vol. 13, no. 3, pp. 345-354, 1998.

[4] A. More and M. Deo, 'Forecasting wind with neural networks', Indian Institute of Technology, Mumbai, India, 2015.

[5] MATLAB and Statistics Toolbox Release 2014a, The MathWorks, Inc., Natick, Massachusetts, United States.

[6] MATLAB and Neural Network Toolbox Release 2014a, The MathWorks, Inc., Natick, Massachusetts, United States.

[7] E. Mitchell, D. Monaghan and N. O'Connor, 'Classification of Sporting Activities Using Smartphone Accelerometers', Sensors, vol. 13, no. 4, pp. 5317-5337, 2013.

[8] A. Mannini and A. Sabatini, 'Machine Learning Methods for Classifying Human Physical Activity from On-Body Accelerometers', Sensors, vol. 10, no. 2, pp. 1154-1175, 2010.

[9] J. Kwapisz, G. Weiss and S. Moore, 'Activity recognition using cell phone accelerometers', SIGKDD Explor. Newsl., vol. 12, no. 2, p. 74, 2011.

[10] Y. Kwon, K. Kang and C. Bae, 'Unsupervised learning for human activity recognition using smartphone sensors', Expert Systems with Applications, vol. 41, no. 14, pp. 6067-6074, 2014.

APPENDIX A
PRE-PROCESSING CODE

```
%%%%%%%%%%%%%%%%%%%%%%%%
%Accelerometer Data Preperation for Classification:
%Lives in a file called "PreProcessing.m", called by both Naive-Bayes
%and Neural Network Classifiers.
%%%%%%%%%%%%%%%%%%%%%%%%

clear all; close all;
init = load('PUC_withUser');
data = init.data;
target = init.target;
%rng('default');
%%now 'data' contains all of the accelerometer data from the HAR data set:
% by column: 1) User ID 2)X1 3)Y1 4) Z1) 5)X2 6)Y2 7)Z2 8)X3 9)Y3 10)Z3
% 11)X4 12)Y4 13)Z4
%and 'target ' containts all of the class data (0-4).
% Class: 0) Sitting 1) Sitting Down 2) Standing 3) Standing Up 4) Walking
%%
%From each exercise of each person, calculate the variance of pitch and
% roll
idx = {};
wholedata = [data, target];
i = 1;
for k=0:4
  for user = 0:3
    [data_idx,I] = find(wholedata(:,1) == user & wholedata(:,14) == k);
    idx{k+1,user+1} = data_idx;
    i = i + 1;
  end
end
%Now idx contains all of the indexes of the users and exercise labels.
%columns are for each user, rows are for each exercise.

window_size = 50;
data = data(:,2:13);
data_preproc = zeros(size(data));
accel=[];
%% Calculate pitch, roll for each filtered accelerometer reading:
for i = 1:size(data,1)
  for j = 1:4
    accel = data(i,((3*j-2):3*j));
    pitch = atan(accel(2)/sqrt((accel(1)^2)+(accel(3)^2)));
    roll = atan(-accel(1)/accel(3));
    accel_module = norm(accel);
    data_preproc(i,(j*3-2):(j*3)) = [pitch,roll,accel_module];
  end
end

%% Normalize the columns of the data:
for i = 1:size(data,2)
  data_preproc(:,i) = mat2gray(data_preproc(:,i));
end
% now data preproc contains the normalized pitch, roll, and magnitude of each
% accelerometer, by column: 1)pitch1 2) rolll 3)magnitude1, and so on for
% all four sensors.
%%
%Now split the data up by which exercise and user it was:
data_windowed_nm = zeros(size(data_preproc,1),(size(data_preproc,2) + 24));
data_windowed_nm(:,1:9) = [data_preproc(:,1:3),zeros(size(data_preproc,1),6)];
data_windowed_nm(:,10:18) = [data_preproc(:,4:6),zeros(size(data_preproc,1),6)];
data_windowed_nm(:,19:27) = [data_preproc(:,7:9),zeros(size(data_preproc,1),6)];
data_windowed_nm(:,28:36) = [data_preproc(:,10:12),zeros(size(data_preproc,1),6)];

%prepare an averaging filter of size filter_size:
filter_size = floor(window_size/((window_size)^(1/3)));
f = fspecial('average',[1 filter_size])'; %create an averaging filter
for m = 1:5
  for n = 1:4
    for i = min(idx{m,n}):max(idx{m,n})
      k = max(i - window_size,min(idx{m,n}));
      temp = data_preproc(k:i,:);
      smoothed = imfilter(data_preproc(k:i,:),f);

      %Calculate the fourier transforms of the window, take the
      %fundamental:
      f_fundamental = fft(smoothed,[],1);
      f_fundamental = f_fundamental(1,:);
      window_var = var(smoothed,1);
      if numel(window_var) > 1
        for j = 1:4
          data_windowed_nm(i,(9*j-5):9*j) = [window_var(3*j-2), ...
                                             window_var(3*j-1), ...
                                             window_var(3*j), ...
                                             f_fundamental((j*3-2):j*3)];
        end
      else
        if (i-1) > 0
          for j = 1:4
            data_windowed_nm(i,(9*j-5):9*j) = data_windowed_nm(i-1,(9*j-5):9*j);
          end
        else
          for j = 1:4
            data_windowed_nm(i,(9*j-5):9*j) = zeros(1,6);
          end
        end
      end
    end
  end
end
%% Shuffle the data:
idx = randperm(size(data_windowed_nm,1));
data_windowed_nm = data_windowed_nm(idx,:);
target = target(idx,:);
target_bin = zeros(size(target,1),5);
for i = 1:size(data,1)
  if (data(i) == 0)
    target_bin(i,:) = [1 0 0 0 0];
  end
```

```
  if (target(i) == 1)
    target_bin(i,:) = [0 1 0 0 0];
  end
  if (target(i) == 2)
    target_bin(i,:) = [0 0 1 0 0];
  end
  if (target(i) == 3)
    target_bin(i,:) = [0 0 0 1 0];
  end
  if (target(i) == 4)
    target_bin(i,:) = [0 0 0 0 1];
  end
end
```

APPENDIX B
NAIVE-BAYES CLASSIFICATION CODE

```
%%%%%%%%%%%%%%%%%%%%%%%%
%% Naive Bayes Classification:
%Takes as input the results of PreProcessing and performs
%Naive Bayes Classification... returns a class error array,
%such that e = [err(class0) err(class1) err(class2) err(class3) err(class4)]
%%%%%%%%%%%%%%%%%%%%%%%%

%Run preprocessing algorithm
PreProcessing

%Split the data into testing and training:
p = .8;
train_length = floor(p*length(target));
data_train = data_windowed_nm(1:train_length,:);
data_test = data_windowed_nm((train_length):length(target),:);
target_train = target(1:train_length);
target_test = target((train_length):length(target));

%% Implement Naive Bayes Classifier:
NBModel = fitcnb(data_train,target_train);
[label] = predict(NBModel,data_test);
C = confusionmat(target_test,label);

% Calculate the misclassification percentage for activities 1-5:
e = [C(1,1)/sum(C(1,:)), ...
     C(2,2)/sum(C(2,:)), ...
     C(3,3)/sum(C(3,:)), ...
     C(4,4)/sum(C(4,:)), ...
     C(5,5)/sum(C(5,:))];
```

APPENDIX C
NEURAL NETWORK CLASSIFICATION CODE

```
%%%%%%%%%%%%%%%%%%%%%%%%
%Neural Network Classification:
%Takes as input the results of PreProcessing and trains an ANN to classify
%the data.
%%%%%%%%%%%%%%%%%%%%%%%%

PreProcessing
x = data_windowed_nm';
t = target_bin';
net = patternnet([75,50]);
view(net)
%%
net = train(net,x,t);
view(net)
y = net(x);
perf = perform(net,t,y);
classes = vec2ind(y);
perf
```