Human Activity Classification

Aristos Athens, Zachary Blum, Navjot Singh.

I. Introduction

Activity recognition is an important task in several healthcare applications. By continuously monitoring and analyzing user activity it is possible to provide automated recommendations to both patients and doctors [2, 6]. There are also applications to consumer products such as data logging for smart watches health apps [3]. Common consumer devices such as smart phones and smart watches generally ship with IMUs (Inertial Measurement Unit), which are packaged accelerometer and gyroscope sensors. Through the information provided by these IMUs, machine learning techniques can then be used to train activity classifiers, giving users, doctors, and app developers access to an individual's lifestyle and activity choices. In this paper we examine two questions in parallel: what is the "best" classification technique and how well can it perform with less features. To compare classification techniques we use a variety of metrics, including classification accuracy, required dataset size, and prediction speed. In particular, we examine the use of logistic regression, support vector machines (SVM), various decision tree techniques, and neural networks. We then examine if these techniques can perform just as well with reduced sensor channels, for example an IMU from a single body location vs. multiple IMU's placed across the body.

II. Related Work

Because of its many applications, supervised human activity classification using sensor data is a relatively popular research area. Through our research, we have found that related articles and their approaches can generally be divided into three categories: Naive Bayes Classification, SVM/Decision Trees, and Neural Networks. We consider the use of Naive Bayes as a classifier for human activity as clever and interesting, since it is usually used for text classification. One such article that uses the Naive Bayes Classifier is Long, Yin, and Aarts, 2009 [9]. This article was similar to our paper in that it included sensor data from multiple subjects for the purpose of multi-class activity classification, and evaluated their models using cross-validation (and compared their results to that of other methods, including decision trees). One way in which this paper differs from our work (and is a strength of the paper) is its use of Principal Component Analysis before conducting Naive Bayes to reduce the feature space and correlation between the features. This classification model, however, only had an accuracy of around 80 percent. One study that examined decision trees was [Parkka, 2006] [5]. Similar to our paper, the study used an ordinary decision tree grown via cross-validation and using the Gini Loss for each split. One strength of the article is that, in addition to this "automatically generated decision tree," the researchers also created a "custom decision tree" using their expert knowledge and analysis of the sensor output. However, the classes in the article are improperly balanced (with one class accounting for between 50 and 60 percent of all of the data), which could affect the true test performance (assuming that the test data also suffers from the same problem). The paper by Youngwook Kim and Hao Ling [7] discusses an interesting approach of human activity recognition through data obtained by a Doppler radar. Similar to our approach, this paper incorporate SVM models that are tuned through cross-validation over a range of hyperparameter values to pick an optimal one. However, the features and characteristics of the data are intrinsically different than the IMU and heart rate data that our paper deals with. For example, we do not have to take into consideration the angle of the subject with respect to the radar and apply a different model based on these edge cases. These variations in the data lead the authors to create a classification model consisting of both decision trees and SVM's. We believe our data is easier to classify and as such can achieve a higher classification rate than the 90% achieved in this paper. We found that the state-of-the-art approach to supervised human activity classification generally involved neural networks, particularly convolution neural networks (CNN) and recurrent neural networks (RNN). A number of studies [8] [12] use multiple datasets to classify similar activities. Both [Hamerla, 2016] and [Xue] incorporate CNN's and KNN's with moving windows in order to look at the activities of a user of a period of time and achieved accuracy results above 95%. Given the limited scope of the CS 229, we did not try to emulate these techniques but believe they can produce more robust models, especially for similar activities like playing soccer or running, where it may be important to look over a history rather than a particular time-stamp.

III. Dataset and Features

We used the PAMAP2 Dataset from the UCI repository of machine learning datasets. [16, 15] This dataset includes raw 9-axis IMU data streams (from the hand, chest, and ankle) as well as heart rate data from nine different subjects performing various activities. Each IMU provides temperature, 3-axis acceleration, 3-axis angular velocity, and 3-axis magnetometer data at a rate of 100 Hz. In total there are 1.9 million data points, each containing 52 features. In the dataset, each time-step is labeled with an activity ID, one of 12 different activities that the subjects were engaged in. The 12 activities are the following: ironing, walking, lying, standing, sitting, Nordic walking, vacuum cleaning, cycling, ascending stairs, descending stairs, running, rope jumping. We had to clean the data for a few reasons. The frequency of the time-stamps matches the highest frequency sensors, the IMUs, which read every 0.01 seconds (100 Hz). However, the heart-rate monitor had a frequency of 9 Hz, meaning approximately 90% of the heart-rate column consisted of Nan values. We filled in missing heart-rate values by linearly interpolating between

nearest valid readings. For the logistic regression, SVM, and neural network models, an additional preprocessing step of removing the training mean and dividing by the training standard deviation was added. We combined each subjects data into a single matrix and divided that into a training and test set. We used 85% of our data for training and 15% for testing. To avoid over-fitting on only a certain subset of classes, we randomly split the data between training and testing. To train our models, we performed 5-fold cross-validation using our training set, to tune the hyperparameters for the given model. We then used this optimized model and analyzed its performance on the test set.

IV. Methods

A. Regression

As a baseline measure, we incorporated a standard logistic regression model with a loss function of the form shown in 1. To make the model more robust against overfitting, L2 regularization was incorporated with the C parameter inversely related to the strength of regularization. In order to pick the optimal value for C, the model was independently trained over a range of C values. The C value that produced the highest accuracy on the validation set was selected and tested on the test set.

$$J(\theta) = \sum_{i=0}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{1}{C}||\theta||_2^2 \tag{1}$$

For each training phase, 5-fold cross validation was incorporated in order to reduce the variance in a trained model. Stochastic Average Gradient (SAG) descent was selected as the solver to use for training as it generally provides fast convergence for large feature-sets such as ours [10]. The SAG solver is only guaranteed to converge quickly if all features are of about the same scale. Thus, the normalization preprocessing step described earlier is necessary.

B. Support Vector Machine

We wanted to also create a non-linear classifier in the hopes that it can outperform the linear logistic regression model, so a SVM was a natural choice, as it is fairly easy to implement with few parameters to tune. We created our SVM's by solving the following primal problem:

$$\begin{aligned} min_{y,w,b}\ & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\zeta_i \\ s.t.\ & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \zeta_i, \quad i = 1, ..., m \\ & \zeta_i \geq 0, \quad i = 1, ..., m \end{aligned} \tag{2}$$

and the decision function is defined as:

$$sgn(\sum_{i=1}^{n} y_i \alpha_i K(x_i, x) + \rho) \tag{3}$$

Through training, an SVM model can create multiple hyperplanes to split the training set into its labeled categories. This is done through the use of kernels that transform the input data into a higher dimension, so that the data can then be linearly separated. Part of the advantage for SVM's is that only a fraction (represented by n in eqn 3) of the original training set has to be retrained for creating the hyperplane during predictions. Another advantage is that various kernel functions can easily be tested and selected for the one that best fits a particular application. Thus, to select the optimal kernel function along with the C parameter specifying the soft-margin size, a grid search was performed over three standard kernel functions (polynomial, rbf, and linear) with a range of C values for each.

C. Deep Learning

Based on our literature review we believed that Deep Learning techniques should work well for this classification problem. We therefore implemented a MultiLayer Perceptron architecture for multi-class classification. A MultiLayer Perceptron architecture is a fully connected feedforward neural network with one input layer, one or more hidden layers and one output layer. Formally, the MLP can be considered a function f: $\mathbb{R}^n \longrightarrow \mathbb{R}^k$, where n is the number of input features and k is the number of classes. Each hidden layer can be formalized as f:$\mathbb{R}^a \longrightarrow \mathbb{R}^b$, where a is the input size and b is the output size. In matrix notation, it would be:

$$f(x) = A_{\mathrm{c}}(W_{\mathrm{c}} x + b_{\mathrm{c}}) \tag{4}$$

where x is the input vector, W_{c} is the weight matrix associated with layer c, and b_{c} is the bias vector associated with layer c, and A_{c} is the activation function associated with layer c.

We use softmax as the final activation, so each prediction is size $(kx1)$ where k is the number of classes. The classifier predicts a score for each class, instead of simply producing a single class label. This can give a sense of how close we are to the correct label. We converted the true labels to use one-hot encoding, that is we took an $(mx1)$ array and made it (mxk), where m is the number of datapoints. We initially found that making the model deeper produced worse results, but increasing the number of neurons per layer improved the accuracy. Our final architecture is Layer1 Relu Layer2 Relu Layer3 Softmax. The layers have weight sizes of Layer1$(n, 512)$, Layer2$(512, k)$. In order to generalize better we apply dropout for each hidden layer; this helps combat overfitting by suppressing each node with 50% probability. We tried different gradient descent rules, with the best result coming from the SGD optimizer. To evaluate loss we use categorical cross entropy, which is as follows:

$$CE(y') = -\sum_{j=i}^{k} y_{\mathrm{j}} log(y'_{\mathrm{j}}) \tag{5}$$

We use this in conjunction with softmax activation for the final layer, which gives us a probability, or confidence, for each prediction. Softmax output for the i^{th} element is as follows:

$$SM(y', i) = -\frac{e^{y'_i}}{\sum_{j=1}^{k} e^{y'_j}} \tag{6}$$

This is advantageous because we have multiple classes, instead of simple binary classification. We want the loss to give a sense

of the degree of error for each category, instead of a simple yes or no. For example, assume we have 3 classes, and the first class is the correct label for this time step. Consider two example outputs: (0.98, 0.01, 0.01) and (0.51, 0.48, 0.01). The first output is clearly superior to the second, because it has a higher confidence for the correct class, however both will "predict" the first class. If we use something like simple error rate for our loss, we are not able to capture the confidence of our predictions. Using softmax activation with cross entropy loss helps us capture this difference. Our final architecture is shown in figure 1. This neural network was implemented using TensorFlow's Keras [1].

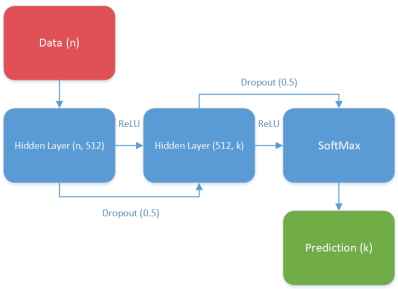

Fig. 1. We achieved our best results with this architecture. Our architecture had size Layer1$(n, 512)$, Layer2$(512, k)$, where n is the number of data features ranging from $12 - 52$ and $k = 18$ is the number of class labels. We use ReLU activation, softmax final activation, cross entropy loss, and applied dropout to avoid overfitting.

D. Trees

Decision trees are a useful method for multi-class classification for nonlinear feature sets. Decision trees perform greedy "splits" on the each feature of the data at a specific threshold. In order to choose a split, a decision tree seeks to maximize the difference between the loss of the parent node and the sum of the losses of the child nodes. The specific loss function we used was the Gini Loss shown below, where p_{mk} is the proportion of examples in class k present in region R_m, and q_m is the proportion of examples in R_m from tree T with $\mid T \mid$ different R_m regions [13]:

$$\sum_{m=1}^{|T|} q_m \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) \tag{7}$$

There are multiple methods of regularizing, or preventing overfitting, for decision trees including setting a minimum size of leaf (terminal) nodes, and setting a maximum tree depth, setting a maximum number of nodes [11]. For this paper, we chose to regularize using the maximum tree depth. All of these decision tree classifiers were implemented using the scikit-learn library [14]

1) Ordinary Decision Trees

The ordinary decision tree classifier uses the above algorithm to create one tree, and we test the resulting tree on a test set. While this generally perform well (depending on the dataset), there are a few reasons for the desire to "ensemble," or combine, multiple decision trees, including primarily the fact that individual decision trees have high variance which could lead to low prediction accuracy [11]

2) Boosted Decision Trees

One form of ensembling is using "boosting," which combines many "weak learners" (simple decision trees) in order to reduce bias in the model (at the expense of increasing variance). Boosting these weak decision trees is done for the purpose of ideally improving accuracy, though it may be prone to overfitting [11]. One specific algorithm for boosting, is AdaBoost (which was used in this paper), as described in [4].

3) Random Forest

Another ensemble method for decision trees, for the purpose of improving prediction accuracy, is random forest. Random Forest is a form of bagging (bootstrap aggregation), which involves sampling with replacement from the original population for the purpose of reducing variance (at the expense of an increase in bias, increased computational cost, and decreased interpretability of the trees). For a random forest, a large number of decision trees are generated, and the bias is further reduced (by decorrelating the trees) by only considering a subset of the total number of features at each split in the decision tree [11].

V. Results and Discussion

As discussed previously, we were interested in classification performance on both the full set of features and performance when using a reduced set of features. We tried several feature combinations and discovered that we could get decent performance when using just the hand IMU plus heart rate sensor. The hand IMU performed better than any individual IMU. This is a positive result, as we are particularly interested in applications where a user is holding a phone or wearing a smart watch. We will refer to the hand IMU plus heart rate data as the "reduced" or "limited" feature-set. Our primary evaluation metric for all models was classification accuracy. This is simply the count of correctly classified data points divided by the count of classifications attempted. It is as follows:

$$f(y') = \frac{1}{m} \sum_{i=1}^{m} 1\{y'_{\mathrm{i}} == y_{\mathrm{i}}\} \tag{8}$$

where y' is our set of predicted labels, and y is the set of true labels. We used various loss functions, as described in the subsection for each technique. The full results are shown below in Figure 3, and explained in detail in the remainder of this section.

	Train Accuracy (Limited)	Test Accuracy (Limited)	Train Accuracy (Full)	Test Accuracy (Full)
Regression	64.2	63.9	81.7	81.5
SVM	98.7	95.0	99.9	98.9
Ordinary Decision Trees	97.8	87.3	98.7	92.7
Boosted Decision Trees	99.9	94.0	99.97	98.5
Random Forest	99.9	93.7	99.97	98.0
Neural Net	86.1	84.0	99.8	98.1

Fig. 2. Training and Test Results for all Methods, for both Full and Limited Feature-sets

A. Logistic Regression

Through 5-fold cross-validation, in fig. 3 we see that the optimal C value for both feature-sets occurs around 0.01.The SAG algorithm automatically calculates the learning rate value, so one was not needed to be selected prior [10]. With a chosen C value of 0.01, the limited feature-set model performed with an 63.9% accuracy on the test set and the full feature-set model performed with a 81.5% accuracy, both of which are very similar to the validation sets. From the confusion matrix (which is not posted due to space constraints), the main sources of error for the both feature-set models are activities like vacuum cleaning and ironing, Nordic walking and walking, or lying and sitting. As expected, these activities have similar body movements, so from the point of view of a sensor reading, these may be hard to distinguish for a linear classifier.

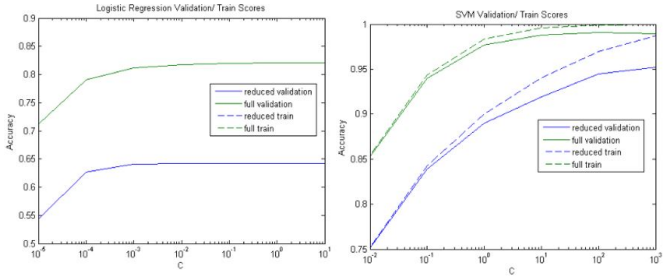

Fig. 3. Validation curves for a given range of regularization constants C for the SVM and logistic regression models. Training curves for logistic regression cannot be seen easily as they lie just above the validation curves

B. Support Vector Machine

Using 5-fold cross-validation for each training model, fig. 3 shows that a good C value for the limited feature-set model is 1000 and 100 for the full. Using these C values, on the test set the limited feature-set and full feature-set models performed with accuracy values of 95.0% and 98.9%, respectively. Due to space constraints, only the graphs for the rbf kernel are shown because it outperformed both the linear and polynomial kernel. Comparing the highest validation scores, the rbf kernel produced scores that were higher than the linear and polynomial kernels by on average about 15% and 10%, respectively for both feature-sets combined. For larger values of C (greater than about 10) the model becomes considerably more expensive to train and predict since the amount of support vectors needed to store becomes larger. For a C value of 100 on the reduced feature-set, the model consisted of about 10,000 training samples, about 8% of the training data. From the confusion matrix (not posted due to space constraints), the main sources of error for the SVM model in general were distinguishing between vacuum cleaning and ironing, ascending stairs and walking, and standing and sitting. These points of confusion are similar to those of logistic regression, but SVM seems to do a much better job in being able to distinguish between active and inactive tasks.

C. Decision Trees

1) Ordinary Decision Trees

In order to tune the maximum depth hyperparameter of the decision tree, we used scikit-learn's validation curve function to perform 5-fold cross-validation. The training and cross-validation curves for the limited feature-set is shown in the left side of 4 below as a function of maximum tree depth, and similar curves were produced for the full feature-set It is

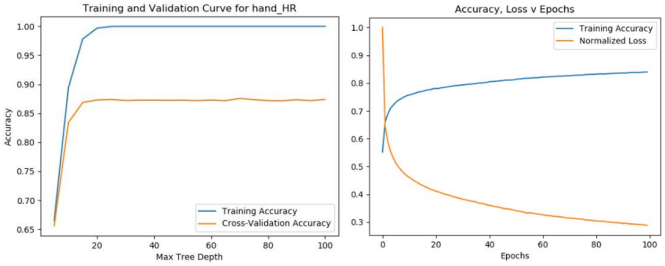

Fig. 4. Left: Training and Validation Curves for Limited Feature-set using Ordinary Decision Trees. Right: Training loss and accuracy for the Multilayer Perceptron neural net when trained on the reduced feature-set.

typical in practice to use the "one-standard error rule" when using cross-validation to choose the simplest model whose accuracy or error is within one standard deviation of the best performing model [4]. This serves to further reduce the chance of overfitting. Using the one-standard error rule for the validation accuracy, the results suggest that for both the limited and full feature-sets, a maximum depth of 15 should be used. Using a maximum depth of 15, the limited feature-set achieved a test accuracy of 0.873 and the full feature-set achieved a test accuracy of 0.927. Because of the size of the tree, it was in-feasible to include the image of the tree in this report. From the confusion matrix for this model, the most frequent misclassifications were between vacuum cleaning and ironing, ascending and descending stairs, ironing and standing, and vacuum cleaning and standing. While the first two common misclassifications are understandable based on the similarity in hand movements and heart rate data, the latter two misclassifications are surprising and highlight areas where the decision tree performed poorly.

2) Boosted Decision Trees

For boosted decision trees, we used the default learning rate, and manually tuned the maximum depth of the base decision tree estimators, and the number of trees. Since the base estimator of boosting should be a "weak learner," and since the "strong learner" from the results from Ordinary Decision Trees had a maximum depth of 15, we decided to limit our weak learners to having a maximum depth of less than or equal to 10. For both the limited and full feature-sets, we created tables that presented the training and validation accuracy values for different combinations of maximum depth of the weak learners (from 1-10) and number of trees (50, 100, 250, or 500). Using the one-standard error rule from the optimal validation performance, we chose 500 trees of max-depth=10 for the limited feature set and 250 trees of max-

depth=9. Using these models, the limited feature-set scored an accuracy of 0.940 on the test set, and the full feature-set scored an accuracy of 0.985 on the test set. A confusion matrix for the limited feature-set is shown below in 5

	Lying	Sitting	Standing	Walking	Running	Cycling	Nordic Walking	Ascending Stairs	Descending Stairs	Vacuum Cleaning	Ironing	Rope Jumping
Lying	28054	154	19	63	0	5	65	10	41	153	195	0
Sitting	17	26652	477	75	0	0	0	12	7	338	141	0
Standing	0	74	26921	36	0	0	9	11	3	795	525	0
Walking	2	4	61	33010	0	89	399	161	61	397	20	0
Running	0	7	49	211	13122	43	212	258	188	83	21	262
Cycling	0	0	0	182	12	23769	232	178	97	31	0	8
Nordic Walking	0	0	68	287	3	110	26562	307	151	37	113	0
Ascending Stairs	5	0	27	511	5	30	194	14487	1125	1369	21	0
Descending Stairs	0	7	35	251	0	74	94	1836	12264	1017	184	0
Vacuum Cleaning	0	7	93	44	0	9	1	387	164	24976	630	0
Ironing	2	3	182	13	0	0	0	27	98	621	34755	0
Rope Jumping	0	0	0	6	419	104	156	57	33	0	0	6226

Fig. 5. Confidence Matrix for Boosted Decision Trees on Limited Feature-set. The four most frequent misclassifications are highlighted in yellow. These represented misclassifications between ascending and descending stairs, and between vacuum cleaning and ascending/descending stairs. These misclassifications are expected because of the relative similarity in hand motions and heart rate between these activities.

3) Random Forest

We decided to include 100 trees in our random forest model, because 100 was a well-performing trade-off of time and accuracy (for random forests, increasing the number of trees will only serve to decrease the variance, and will not increase the likelihood of overfitting). We used the default option of only considering a random subset of the square-root of the total number of features for each split. In order to tune the maximum depth for each decision tree in the random forest, we again used scikit-learn's validation curve function to perform 5-fold cross-validation. The training and cross-validation curves for the limited feature-set and the full feature-set, as a function of maximum tree depth in the random forest, were created using a similar approach to the cross-validation for ordinary decision trees. Using the one-standard error rule for the validation accuracy, the results suggest that for both the limited and full feature-sets, a maximum depth of 20 should be used. Using a maximum depth of 20, the limited feature-set achieved a test accuracy of 0.937 and the full feature-set achieved a test accuracy of 0.980. From the confusion matrix for this model, the most frequent misclassifications were between vacuum cleaning and ironing, ascending and descending stairs, vacuum cleaning and ascending/descending stairs. These are exactly the common misclassifications found in boosting, and are expected because of the relative similarity in hand motions and heart rate between these activities.

D. Deep Learning

The MultiLayer Perceptron neural network was able to achieve high classification accuracy. When trained on the entire dataset, it consistently achieved training accuracies greater than 98% and test accuracies greater than 95%. The best model we trained produced a test accuracy of 98.1%. When trained on the reduced feature-set consisting of only the hand IMU and the heart rate sensor, it achieved 81.4 % test accuracy. We noted a trend in accuracy vs hidden layer size. Increasing the size of each layer (number of neurons) improved performance, while increasing the depth (number of hidden layers) degraded performance. We did not notice a significant difference in performance when using ReLU vs other activation functions. However, we did find that our model converged faster when using softmax for the final activation function in conjunction with categorical cross entropy loss. As expected, reducing the dropout rate tended to improve training accuracy, but reducing it too much caused a degradation in test accuracy. In this project we were limited in both time and compute, and we believe we can improve accuracy given more of both. We can improve performance by training for more epochs. Loss continued to decrease at the end of our training, indicating performance was still improving when training finished. We could also further increase the number of neurons per hidden layer, at the cost of a larger model with slower training time. Figure 4 shows the training accuracy and loss when this model was trained on the limited feature-set for 100 epochs.

VI. Conclusion and Future Work

Unsurprisingly, logistic regression performed the worst. Having the advantage of extremely low memory usage and speed for predictions, it can still be a viable method in low-compute devices like microcontrollers. SVM's, on the other hand, provide great results in accuracy but may not be viable solutions for microcontrollers because of the large number of support vectors required to store and do operations on for predicting. For decision tree methods, as expected, ensembling methods improved test performance over ordinary decision trees for both the full and limited feature-sets. Boosted decision trees performed slightly better than random forest on both feature sets, which is promising because it is generally less computationally intensive, and thus is a good candidate for a model to actually deploy in a smart device. We would also be interested in exploring different types of Deep Learning architectures. We considered using RNN's (Recurrent Neural Networks) but our feature-set had a relatively large number of features per time step, and the activities did not involve more than a few actions, so it was not necessary to take history into account when classifying a single time-step. As such, simple feed-forward neural nets were sufficient for this problem. However, we would like to explore CNN's (Convolutional Neural Networks) which could potentially give similar or improved performance while using substantially less memory. In general, the limited feature-set performed only slightly worse than the full feature-set on all of the methods, which is a promising result for actual deployment in smart devices. In the future, we would like to test these models using real IMU's. In particular, we would want to see if a low-compute embedded device could perform classifications with neural nets or SVM's in real-time, in addition to computationally cheaper methods such as decision trees.

Appendices

All code used in this project can be found at: https://github.com/aristosathens/Human$_{A}ctivity_{C}lassifier$

Contributions

Aristos, Zach, and Navjot all contributed equally to this project. Aristos focused on Deep Learning, Navjot focused

on Logistic Regression and SVM, and Zach focused on Trees (ordinary decision trees, boosting, and random forests). All three members worked on data preprocessing, analysis, and writing this report.

REFERENCES

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] C. E. Matthews et al. "Amount of time spent in sedentary behaviors and cause-specific mortality in US adults". In: *American Journal of Clinical Nutrition* (2012). URL: https://www.ncbi.nlm.nih.gov/pubmed/22218159.

[3] Xiaoyong Chai and Qiang Yang. "Multiple-Goal Recognition from Low-Level Signals". In: *AAAI* (2005). URL: http://www.aaai.org/Papers/AAAI/2005/AAAI05-001.pdf.

[4] Trevor Hasting. *The Elements of Statistical Learning*. URL: https://web.stanford.edu/~hastie/Papers/ESLII.pdf. pg. 339.

[5] Panu Korpipa Juha Parkka Miikka Ermes. "Activity Classification Using Realistic Data From Wearable Sensors". In: *IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE*. IEEE, 2006. URL: https://s3.amazonaws.com/academia.edu.documents/12195796/parkka06.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1544770601&Signature=QBZmGI0nx4\%2FLEqsaBbGl6Sm1AXQ\%3D&response-content-disposition=inline\%3B\%20filename\%3DActivity_Classification_Using_Realistic.pdf.

[6] P. T. Katzmarzyk and I-M. Lee. "Sedentary behaviour and life expectancy in the USA: A cause-deleted life table analysis". In: *BMJ Open* (2012). URL: https://bmjopen.bmj.com/content/2/4/e000828.

[7] Youngwook Kim and Hao Ling. "Human Activity Classification Based on Micro-Doppler Signatures Using a Support Vector Machine". In: *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING*. IEEE, 2009. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4801689&tag=1.

[8] Nie Lanshun Li Jiazhen Ding Renjie Zhan Dechen Chu Dianhui Li Xue Si Xiandong. "Understanding and Improving Deep Neural Network for Activity Recognition". In: School of Computer Science and Technology, Harbin Institute of Technology. URL: https://arxiv.org/ftp/arxiv/papers/1805/1805.07020.pdf.

[9] B; Aarts R.M Long X.; Yin. "Single-accelerometer-based daily physical activity classification". In: *Proceedings of the EMBC*. IEEE, 2009. URL: https://pure.tue.nl/ws/files/2828549/Metis234421.pdf.

[10] Francis Bach Mark Schmidt Nicolas Le Roux. *Minimizing Finite Sums with the Stochastic Average Gradient*. 2nd ed. Springer.

[11] Andrew Ng. *CS229 Course Notes Decision Trees*. URL: http://cs229.stanford.edu/notes/rf-notes.pdf. CS229 online course notes.

[12] Thomas Plotz Nils Y. Hammerla1 Shane Halloran2. "Deep, Convolutional, and Recurrent Models for Human Activity Recognition using Wearables". In: Open Lab, School of Computing Science, Newcastle University, UK. URL: https://arxiv.org/pdf/1604.08880.pdf.

[13] Rajan Patel. *STATS 202 Lecture 19*. URL: https://web.stanford.edu/class/stats202/content/lec19-cond.pdf. STATS 202 online course notes.

[14] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[15] Attila Reiss. *PAMAP2_Dataset: Physical Activity Monitoring*. URL: https://archive.ics.uci.edu/ml/machine-learning-databases/00231/readme.pdf. UCI Machine Learning dataset.

[16] Attila Reiss and Dider Stricker. "Introducing a New Benchmarked Dataset for Activity Monitoring". In: *IEEE Computer Society Washington* (2012). URL: https://dl.acm.org/citation.cfm?id=2358027.