Activity Classification using Smartphone Accelerometer Data

Anvisha Pai, Ofir Nachum, Max Kanter

December 5, 2013

1 Introduction

Mobile devices such as smartphones and smart-watches now come with in-built, highly accurate accelerometers. An accelerometer can measure absolute acceleration in the x, y, and z axes. They are commonly used for user-interface control, to make the phone responsive to actions like tilting or shaking.

Accelerometer data can also be analyzed to infer user activities on a larger scale. For example, being able to detect when a user is walking, jogging or sitting. With a system that is able to classify activities to a high accuracy, smartphones and smart-watches can provide personal analytics for health and rehabilitation purposes.

1.1 Related Work

There have been some research efforts into identifying activities from smartphone accelerometer data. Kwapisz et al. [1] mined data from smartphone sensors of different users doing different activities, and extracted statistics like the average, standard deviation, and time between peaks from portions of the data. With the features they used, they achieved 91.7% precision overall using Multilayer Perceptron. Ravi et al. [2] attempted to classify similar activities using the mean, standard deviation, energy of the Fourier Transform and correlation between signals on each axis. On a dataset similar to Kwapisz et al., they achieved 72% accuracy using Boosted SVM.

1.2 Problem Statement

Activity recognition is a interesting problem for machine learning because each user performs the same activity sightly differently. Therefore, it is important to find a method that can generalize the important features of an activity rather than the specifics of how a particular user performs the activity.

Our goal is to improve upon the results of the Kwapisz et al. data set [1] by investigating different methods for classification. Increasing classification accuracy will make this data more useful and suitable for use in medical applications. We decided to experiment with Support Vector Machines (SVMs) because they generally perform well in similar classification tasks, and have been used in other activity detection papers [2]. Moreover, our study of Hidden Markov Models (HMMs) in class motivated us to apply them to activity detection. Modeling activities with hidden states is a very intuitive approach because many activities inherently involve smaller-scale movements. For example, walking consists of moving the right foot up, then the right foot down, and subsequently the left foot up and down.

2 Data

We use data collected by Kwapisz, et. al. [1]. This data contains accelerometer data collected from 34 people performing six distinct daily activities (walking, jogging, sitting, standing, going upstairs, going downstairs). Each row of the data contains the accelerometer readings in the x, y, and z directions collected at 30 Hz.

2.1 Feature Extraction

In order to extract features from this data, we split it into *samples* of N consecutive readings, with $N/2$ overlap between consecutive samples. The choice of N depends on the method of classification and is discussed in the respective sections on SVMs and HMMs.

Once we have the data separated into samples of specified window size, we compute several statistical measures on the readings contained in the sample:

1. Average X
2. Average Y
3. Average Z
4. Standard deviation X
5. Standard deviation Y
6. Standard deviation Z
7. Covariance X, Y
8. Covariance Y, Z
9. Covariance Z, X
10. Correlation X, Y
11. Correlation Y, Z
12. Correlation Z, X

We chose these features by reviewing the existing literature (see Related Work). Additionally, we verified a feature's contribution by comparing test error (using SVM) with and without the feature. For example, although some of the existing papers on this topic use FFT-derived features, we did not see a non-negligible improvement when including these features, and thus decided not to include them.

2.2 Testing Strategy

2.2.1 Measuring Performance

The data we use does not represent the six activities equally. In fact, it is quite heavily skewed, with 38.6% of the data corresponding to walking, about 31.2% corresponding to jogging, and then 11.2%, 9.1%, 5.5%, and 4.4% corresponding to upstairs, downstairs, sitting, and standing, respectively. It is important to quantitatively score models in a way that encourages accuracy in all activities. For this reason, we do not use the standard 0-1 loss to score our models. Instead, we utilize the F1 score. The F1 score combines recall (accuracy relative to the true classification) and precision (accuracy relative to the model's classification). Thus, if our model is $\hat{h}$ and our testing set is given by $x^{(1)}, \ldots, x^{(n)}$ with labels $y^{(1)}, \ldots, y^{(n)} \in \{1, \ldots, K\}$ the F1 score for label $k \in \{1, \ldots, K\}$ is computed by

$$F1(\hat{h}, k) = 2\frac{precision(\hat{h}, k) recall(\hat{h}, k)}{precision(\hat{h}, k) + recall(\hat{h}, k)},$$

where

$$precision(\hat{h}, k) = \frac{\sum_1^n [[y^{(i)} = \hat{h}(x^{(i)}) = k]]}{\sum_1^n [[\hat{h}(x^{(i)}) = k]]}$$

$$recall(\hat{h}, k) = \frac{\sum_1^n [[y^{(i)} = \hat{h}(x^{(i)}) = k]]}{\sum_1^n [[y^{(i)} = k]]}.$$

To score the model considering all labels k, we take the weighted average:

$$Score(\hat{h}) = \sum_{k=1}^{K} F1(\hat{h}, k) \cdot \frac{\sum_1^n [[y^{(i)} = k]]}{n}.$$

We term this the *weighted F1 score* or the *F1 score* of the model $\hat{h}$ and we use this as the measure by which to compare the performance of different models.

2.2.2 Training and Testing

For both SVMs and HMMs we split the data (samples for SVMs, sequences of samples for HMMs) randomly into 50% training and 50% testing. The F1 score of the resulting model for SVMs is largely unchanged from one random split of training and testing to another. On the other hand, the F1 score for HMMs varies in a non-negligible way. Therefore in order to generate a consistent F1 score for an HMM model, we perform the random split of training and testing three times and average the F1 scores of the three resultant models.

3 Support Vector Machines

3.1 Theory

Support vector machines (SVM) are one of the most fundamental methods of classification in machine learning. They are frequently used in supervised learning.

Finding a suitable SVM on a set of data reduces to a solving a quadratic program. We use the MATLAB implementation `svmtrain`, which uses the Sequential Minimal Optimization (SMO) algorithm, to solve this optimization problem.

3.1.1 Procedure

Since we have six different activities, we must train six times to obtain six SVM models, each one classifying one activity (labels for samples corresponding to other activities are zeroed out). When testing, we classify according to the SVM which returns a positive classification. If multiple SVMs or no SVMs return a positive classification, we return an arbitrary one.

3.1.2 Data Preperation and Window Size

We chose a window size $N = 256$, corresponding to 8.5 seconds. This window is then converted to a 12-element feature vector (means, standard deviations, etc.). This window is similar to windows of related works.

3.2 Parameter Setting

When using SVM, we have a choice of kernels as well as C (the penalty on violating constraints). In either case, the goal is to maximize weighted F1 score. To this end, analyzed the results of using linear, quadratic, and radial basis kernels for C in the range $[0.1, 1024]$. Figure 1 shows the results of using a linear kernel. For $C > 1$, the SVM algorithm did not converge in a reasonable amount of time.

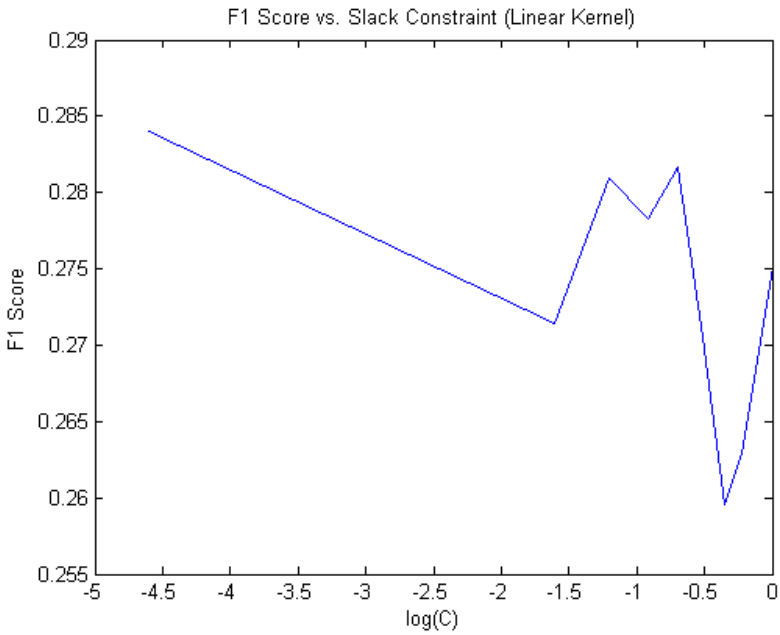

Figure 1: SVM with a linear kernel.

Figure 2 shows the results of using a quadratic kernel. Again, for $C > 1$, the SVM algorithm did not converge in a reasonable amount of time.

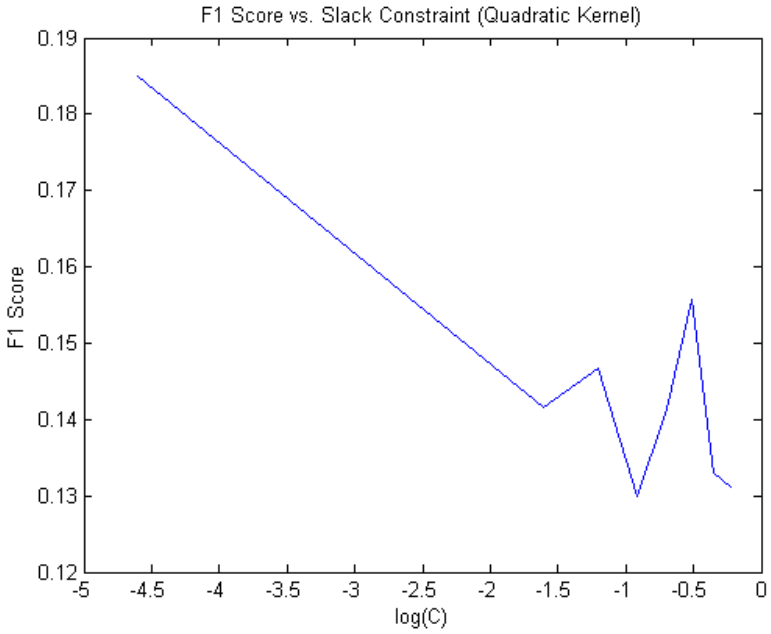

Figure 2: SVM with a quadratic kernel

Figure 3 shows the results of using a radial basis kernel.

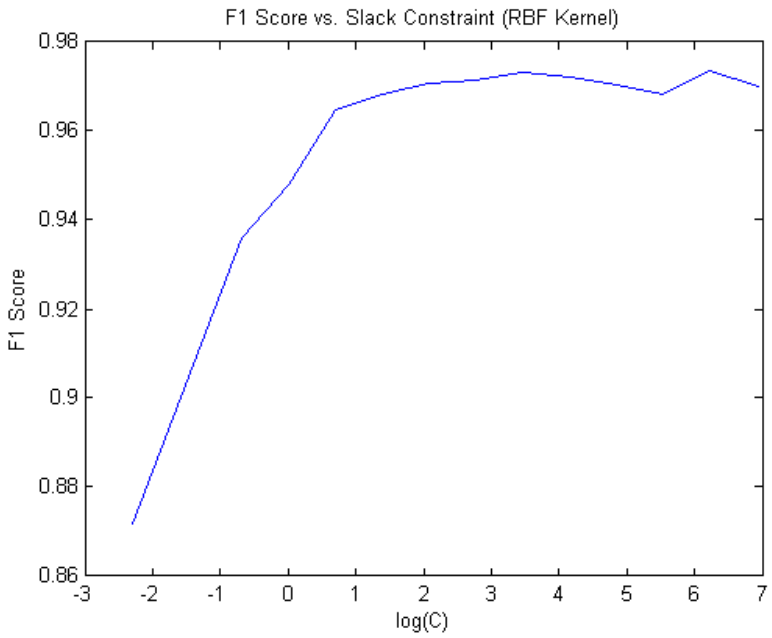

Figure 3: SVM with a radial basis kernel

These results show that a radial basis kernel with $C = 512$ is the best setting for training an SVM, yielding a weighted F1 score of 0.9735.

4 Hidden Markov Models

4.1 Theory

Hidden Markov Models (HMM) have a wide range of applications in building a statistical model to represent signals, like in speech recognition, natural language processing and computational biology. Since activities like walking and jogging are composed of a periodic repetition of certain actions, we believe that they can be modeled accurately using HMMs.

Hidden Markov models have some set of hidden states $s = 1, 2, ..., S$. Along with the transition matrix A, the emission matrix B and a vector of initial state probabilities, Π, we can fully specify a Markov model. In our case, we have a series of observations O. We used the Python library scikit-learn, and its training function, `hmm.fit`.

Scikit-learn's implementation of `hmm.fit` uses the iterative Expectation-Maximization algorithm, the Baum-Welch Algorithm. The algorithm trains until the increase in likelihood of the model is less than a specified convergence threshold.

4.2 Using HMMs for classification

4.2.1 Procedure

We trained an HMM for each activity using sets of 8.5-second sequences. This results in six distinct HMMs (one for each activity).

For testing, we computed the likelihood of the test sequence for each HMM and classified the sequence as the activity corresponding to the max-likelihood HMM, as shown in Figure 4 below.

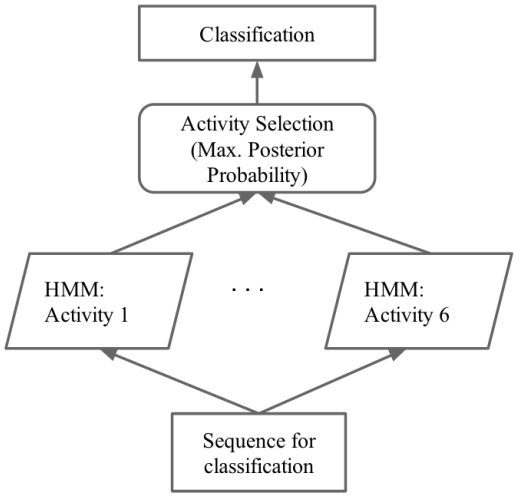

Figure 4: HMM Classification Procedure

4.2.2 Data Preparation and Window Size

We chose a small window size of $N = 16$ (0.5 seconds) for HMMs. The choice of $N = 16$ was made after preliminary analysis of the accuracy of an HMM model on different window sizes. Figure 5 shows how a smaller window performs better. The choice of $N = 16$ is small enough to allow for longer sequences while large enough so that the features (average, standard deviation, correlation, etc.) are still meaningful.

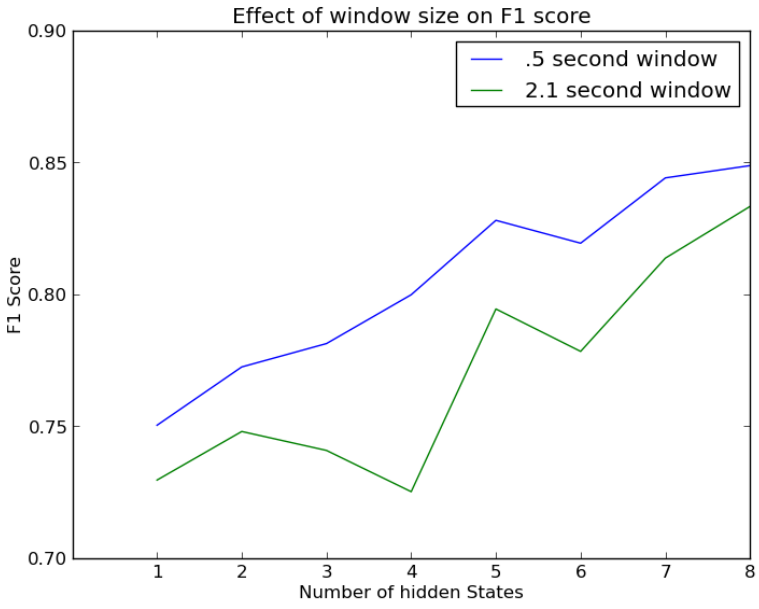

Figure 5: Determining window size

Training an HMM requires a sequence of these samples. In order to make the HMM and SVM classifications comparable, we use a sequence of 32 samples, which corresponds to 8.5 seconds (recall that consecutive samples overlap by $N/2$). Figure 6 illustrates this procedure.

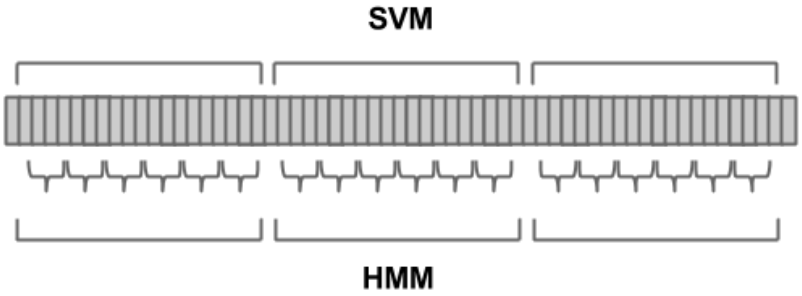

Figure 6: SVMs uses a sample window size of $N = 256$. Although HMMs use a smaller sample window size of $N = 16$, consecutive samples are combined into a sequence, making the classification problem equivalent to that of the SVMs.

4.3 Parameters Used

We focused on optimizing two parameters of our HMM classifier: the number of hidden states S and the convergence threshold to stop training the classifier.

Rather than try an exhaustive search of the combinations of the two parameters, we took a two step approach.

First, we fixed the convergence threshold at 10 and found the the value of S that maximized the F1 score. We first considered values of S from a minimum of 2 states up to 32 states, which is the maximum number of samples in a sequence. However, in practice we found calculating $S > 12$ to take too long.

From Figure 7 we can see that the best number of states to use is 7. After 7 hidden states the increase in model complexity doesn't lead to significant improvement in the F1 score.

Second, we fixed S and iterated over possible values for the convergence threshold. We decided to test 8 different orders of magnitude. Figure 8 shows that thresholds between .01 and 100 perform similar with respect to F1 score. We selected 100 as our convergence threshold because it talks significantly fewer iterations of Baum-Welch to train.

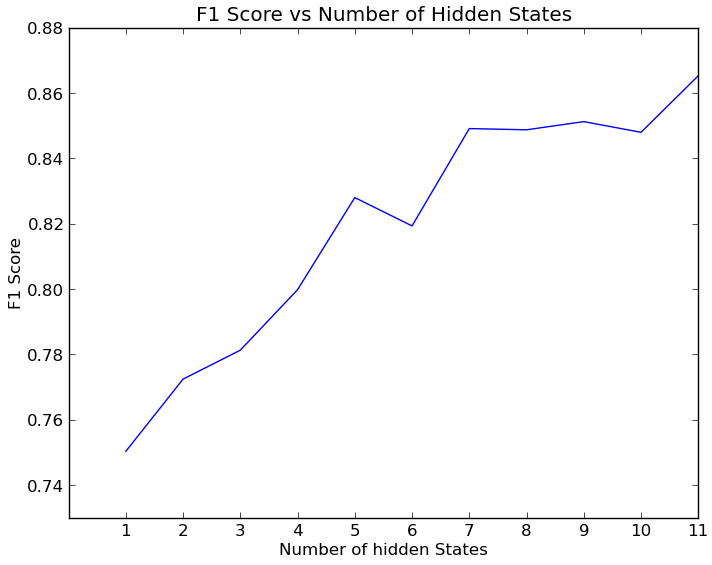

Figure 7: Searching for optimal number of states

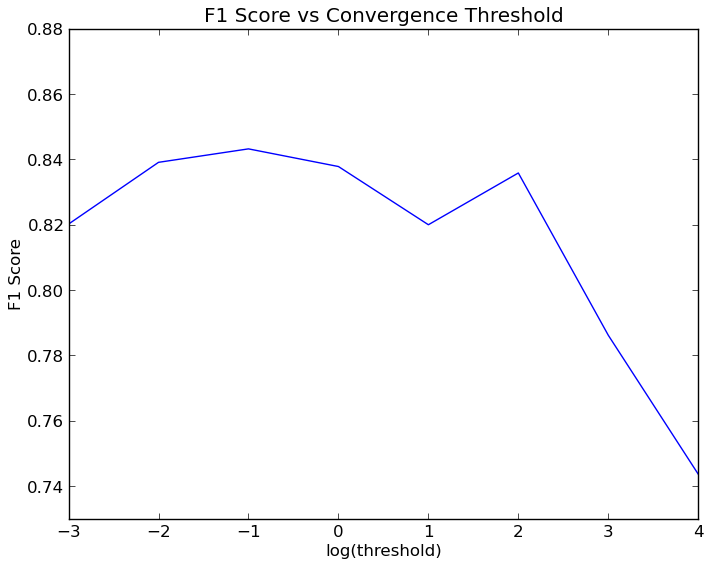

Figure 8: Searching for optimal convergence threshold

5 Results

We used our implementations of HMMs and SVMs to build up a confusion matrix across the different activities. A confusion matrix displays the number of samples for each known label and predicted label combination. From the confusion matrix, we calculated the precision and recall for each activity and the resulting F1 score.

5.1 SVM

An SVM trained using a radial basis kernel and $C = 512$ on 50% of the samples yields 0.9735 F1 Score. The classifications are represented in the following Table 1.

	Walking	Jogging	Sitting	Standing	Upstairs	Downstairs	Precision
Walking	1643	0	0	0	10	6	99.0%
Jogging	0	1319	0	0	3	0	99.8%
Sitting	0	0	185	0	0	0	100%
Standing	0	0	2	158	0	0	98.8%
Upstairs	5	2	0	0	397	6	96.8%
Downstairs	13	4	8	7	25	364	86.5%
Recall	98.9%	99.6%	94.9%	95.8%	91.3%	96.8%	

Table 1: Confusion matrix for SVM.

5.2 HMM

An HMM with 7 hidden states and a convergence threshold of 100 achieves a F1 Score of 0.8556. The confusion matrix is shown in Table 2.

	Walking	Jogging	Sitting	Standing	Upstairs	Downstairs	Precision
Walking	1365	18	3	0	0	108	91.4%
Jogging	32	1142	46	3	12	25	90.6%
Sitting	0	0	216	9	0	0	96%
Standing	1	17	2	160	0	1	88.4%
Upstairs	47	17	1	0	273	47	71.0%
Downstairs	27	6	0	0	57	224	71.3%
Recall	94.4%	95.2%	80.6%	93.0%	80.0%	55.3%	

Table 2: Confusion matrix for HMM.

5.3 Discussion

On the whole, the SVM implementation performed very well with an F1 score of 0.9735.

Table 3 compares the precision on activity detection from our SVM and HMM implementation to Kw [1] on the same dataset. Compared to their best-performing algorithm, the Multilayer Perceptron, our SVM implementation had a higher accuracy and precision across every activity.

	Our Implementation SVM	Our Implementation HMM	Kwapisz et al. Multilayer Perceptron	Kwapisz et al. Best Result
Walking	99%	91.4%	91.7%	93.6%
Jogging	99.8%	90.6%	98.3%	98.3%
Sitting	100%	96.0%	95.0%	95.7%
Standing	98.8%	88.4%	91.9%	93.3%
Upstairs	96.8%	71.0%	61.5%	61.5%
Downstairs	86.5%	71.3%	44.3%	55.5%

Table 3: Comparing our results to Kwapisz et al.

Our SVM implementation performs significantly better than Kwapisz et al. for every activity. SVMs also perform better than HMMs for precision and recall, with an F1 score of 0.9735 compared to the HMM score of 0.8556.

In particular, SVMs do a better job discerning between similar activities such as walking, upstairs, and downstairs. These activities involve very similar motions, and distinguishing between them is difficult because each user performs them slightly differently. We believe this difference is explained because SVMs use support vectors, and are fundamentally focused on training based off the training points that are most similar between two activities. The focus on drawing the border between two similar, but different training points during training translates to better performance classifying similar activities during testing.

Our HMM implementation also performs better than Kwapisz et al. for certain activities, namely upstairs and downstairs. We also perform comparably on walking, but slightly worse on jogging, sitting and standing. This might be because we used the same number of states for each HMM ($S = 7$), which could incorrectly represent the real hidden states. It is possible that $S = 7$ was optimal for walking, upstairs and downstairs - but not for standing or sitting which could be seen as just 1 or 2 hidden states.

6 Conclusion

Activity detection based on accelerometer data is a difficult machine learning problem. We tackled this problem in a dataset with six activities and multiple subjects by segmenting the data into distinct windows and extracting statistics like the average, standard deviation and correlation between axes on each window. We then used SVMs and HMMs to classify the data.

We were able to improve upon previous work that used the same dataset. SVMs performed extremely well for all activities with an F1 score of 0.9735, and significantly outperformed Kwapisz et al. HMMs did not perform as well, with an F1 score of 0.8556. However, they outperformed Kwapisz et al. for certain activities.

We came away with important insights on why HMMs might not work as well as SVMs on this dataset, namely that each activity might have a different number of hidden states. Moreover, SVMs work better because they focus on uncovering the support vectors, i.e. activities that are very similar; thus they perform better on tricky activities like walking, upstairs and downstairs that have similar features.

7 Future Work

There is still room for improving our models in future work. When we trained our HMMs we assumed a fixed window size and number of hidden states across each activity model. One possibility to improve performance of HMMs is to unfix these values and set them independently for each activity.

In addition, there is the possibility that our feature vectors can be improved. There are possibly many more ways to combine the data within a single window to create a valuable feature.

It may also be interesting to take on the challenge of classifying more diverse activities (for example, playing soccer, sleeping). Or we may take a new direction and attempt to classify the user rather than the activity.

Finally, we noticed throughout the study that different techniques have different strengths when classifying activities. To take advantage of this, it would be interesting to explore the effectiveness of meta methods, such as boosting, in a future study.

References

[1] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. Activity recognition using cell phone accelerometers. *SIGKDD Explor. Newsl.*, 12(2):74–82, March 2011.

[2] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L. Littman. Activity recognition from accelerometer data. In *Proceedings of the 17th Conference on Innovative Applications of Artificial Intelligence - Volume 3*, IAAI'05, pages 1541–1546. AAAI Press, 2005.