

Human Activity Recognition via Cellphone Sensor Data

Wei Ji, Huguang Liu, Jonathan Fisher

Abstract—The purpose of this project is to identify human activities while using cell phones via mobile sensor data. We collect 2085 data samples, which includes 3-axis acceleration, angular velocity and orientation sensor data, from 4 volunteers using the MATLAB Mobile package. After cleaning, interpolating, and FFT, we get 135 raw features, and we further reduce the feature number to 21 via feature selection. After comparing the results of different models, e.g. Decision Tree, SVM, KNN, we successfully build an Ensembled Bagged Trees Model which gives 95.7% prediction accuracy over 626 test data on 9 human activities (walking, running, driving, typing, writing, talking, laying, sitting, standing).

I. INTRODUCTION

Mobile has become a necessity in people’s daily life in recent few years. In 2015, the mobile ecosystem had 4.7 billion unique subscriber and generated more than \$3.1 trillion revenue, which is 4.2% of the global GDP. So the ability to identify user activity and behavior in real time via cellphone sensor data is becoming extremely useful to a variety of applications: medical companies can use sensor data and user activities to keep track of bio-signals, tech companies can feed news, music or ads when users are performing different activities, e.g. sitting, running or driving, retail companies can use customer behaviors combined with GPS data to guess users propensity to consume. Using motion sensor data, like accelerator, gyros, orientation etc., to identify user behavior had been tried before by [1] [2] [3] [4] [5]. The experimental setup of [1] [2] [3] assume that the sensors are attached to the user waist or back pocket, which creates a somewhat artificial scenario. [4] and [5] has considered real-time scenario, which assumes cellphone are at hand or inside of pocket, but [4] only achieve 80% accuracy and [5] only achieved 91.15% accuracy due to using only acceleration data. In this paper we’ll remove the sensor attachment constraint. We assume a more real scenario where the user is holding the phone and also using all available cellphone sensor data to do prediction.

II. DATA SET

A. Data Collecting

MATLAB Mobile supports acquisition of data from built-in sensors on iPhone and Android Phones and also provides remote access to the script and data on the laptop with the same MATLAB session or MathWorks Cloud. MATLAB Support Package for Android/iOS sensors provides the displaying, logging, querying and sending sensor data.

Thus, we acquire approximately 15000 seconds of sensor data [Table 1] at 50Hz sample rate from 4 volunteers by taking the following steps:

TABLE I: Raw Sensor Data

Sensor	Features
Acceleration	3-axes acceleration data
Angular Velocity	3-axes gyroscopes for rotational motion
Magnetic Field	3-axes magnetic field
Orientation	azimuth, pitch, and roll
Position	latitude, longitude, altitude

- 1) Configure MATLAB Mobile on the mobile device. Connect mobile device to Cloud.
- 2) Load collecting script to MathWorks Cloud, so the MATLAB Mobile has access to the script.
- 3) Tag collecting script for the performing activity before recording.
- 4) Run the script which initialize a mobiledev object, enable sensors, set sample rate to 50Hz and start recording timer.
- 5) Save the logging results to Cloud after recording.

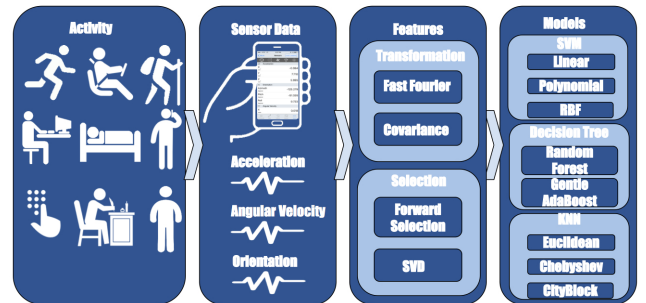


Fig. 1: Flow of work:

B. Data Preprocessing

We have several challenges when we process data. First challenge is when we identified during data collection is Sensors recording duplicate data samples. The Second challenge is that different sensors recording are not synchronized and the time range is not accurate either. Sample rate of sensor is not exactly 50Hz and not consistent. The time internal ranges between [0.018, 0.022].

To solve these problems, we

- 1) Cleaned the sensor data by de-duplicate rows and sort by timestamp.
- 2) We divide each recording sample into $0.02 \times 128 = 2.56$ second segments. And each segment is used as a data point.

- 3) Only choose the data sample with overlapping time span cross all sensors.
- 4) To make all the sensor data synchronized at the exact same sample rate, we interpolated and re sampled the data. We tried several interpolation methods like: Nearest Neighbor, Linear, Cubic, and Spline. Spline gives the best result, because it doesn't over smooth the sample data like cubic interpolation or sharpen the sample data like linear.

III. FEATURES

A. Sensors selection

After we examining the collected data we realized that the position data and magnetic field data is independent of user's behavior. So we have 9*128 raw features on time domain.

B. Feature Transformation

We expect some of the collected data to be periodic (with unknown initial phase), like the data collected while walking, going upstairs and going downstairs, and some to be random, like driving. This is why looking at the spectral components of the sampled data makes much more sense than looking at raw data. After converting each sensor 128 time samples using FFT we examined the top 10 spectrum give the best performance (frequencies in the range 0 - BW) on several ML algorithm as presented in the following subsection. Fig. 2 verifies our assumption - it can be seen that the blue, yellow and purple lines have pics at spectral components at frequencies greater than zero while the red and green lines don't. Thus we have 9*10 features. We also evaluate the covariance of every 2 sensors, which gives 45 features.

C. Feature Selection

135 features is still too many to make algorithms run efficiently. Large feature number can potentially lead to a very complex model, which tends to over-fit training data. We adopt SVD and forward feature selection to reduce feature dimensions.

1) *SVD reduce feature dimension*: Running a series of SVD on a bagged tree model shows that Top 7 features can preserve 95% of total energy, and give a 9.3% test error. Top 22 features can preserve 99% of total energy, which gives 6.6% test error. Top 33 features can preserve 99.94% of total energy, and give a 6.4% test error. Fig 3.

2) *Forward selection reduce feature dimension*: Running a series of forward feature selections on the bagged tree model shows that, the 18, 21, 27-feature model overplay 135 feature model. The 21 feature model achieves lowest test error 4.3% at number of trees larger than 300. This is better than the performance of 33 features set using SVD feature selection. Work shown in Fig. 4

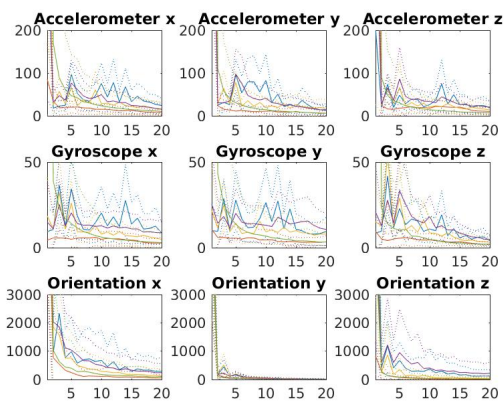


Fig. 2: The average FFT Spectrum of different sensors blue: walking, red: driving, yellow: walking upstairs, purple: walking downstairs, green: sitting. Solid curve: average of all sample data dashed curve: 1σ clearance band.

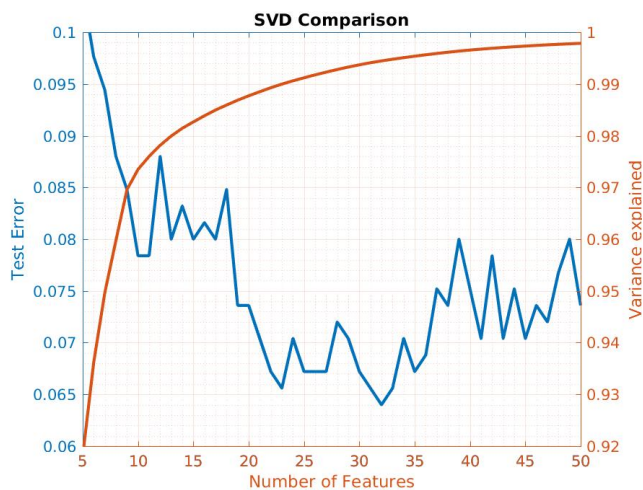


Fig. 3: Feature Selection using SVD method : blue curve shows test error using principle components with Ensemble Bagged tree classification. Red curve shows how much variance is explained by these principle components.

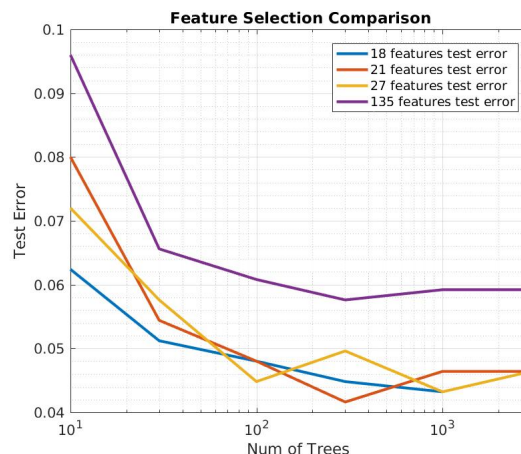


Fig. 4: Feature Selection using Forward Selection method: blue, red, yellow, purple curve shows test error using 18,21,27,135 features with Ensemble Bagged tree classification.

IV. METHODOLOGY

A. Model Comparison

We collect, process, and select 21 features on four users performing nine different activities and try to find the best classification model. We compare K-Nearest Neighbors(KNN), Support vector machine(SVM), and Decision Tree method using holdout validation.

1) *K-Nearest Neighbors*: The kNN algorithm predicts according to the k "nearest" neighbors in feature space. We compare kNN models with different distance function and k number in Fig. 5. However, because of the large variance between different sensors, we do not get a good result from this method. Results show that performance of the Euclidean and Cityblock distance functions are better than performance of Chebychev distance. The best result we get for the kNN method is 7%.

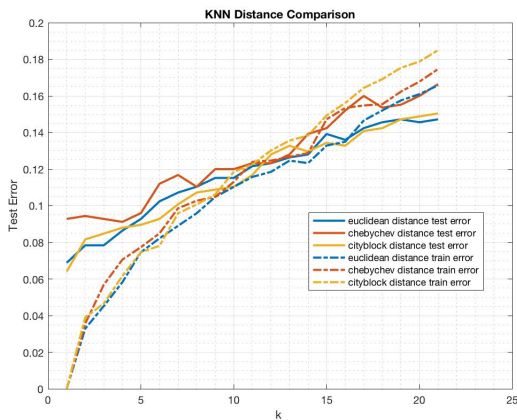


Fig. 5: KNN Model Comparison: training(solid line) and test(dashed line) error of distance computed by Euclidean(blue), Chebychev(red), Cityblock(yellow) method.

2) *Support Vector Machine*: The SVM algorithm predicts by computing the hyperplane with largest margin as the classification separation plane. The SVM algorithm only works with binary classifiers. To allow multi-class classification, we use one-vs-one method to create k(k-1) binary classifier with SVM, where k is the number of labels. Also to allow mis-classifications, we add a penalty factor C to the cost function. The cost function to minimize for SVM is formula 1

$$\min_{w,b,s} \left(\frac{1}{2} \langle w, w \rangle + C \sum_i s_i \right) \quad (1)$$

$$s.t. \quad y_i(\langle w, x_i \rangle + b) \geq 1 - s_i \quad \text{and} \quad s_i \geq 0$$

. A larger C value increases the weight of mis-classifications, which leads to a stricter separation. This factor C is also called box constraint. We compare SVM models with different kernel and different C value in Fig. 6. Linear kernel outperforms polynomial and rbf kernel. The best result we get for the SVM method is 7%.

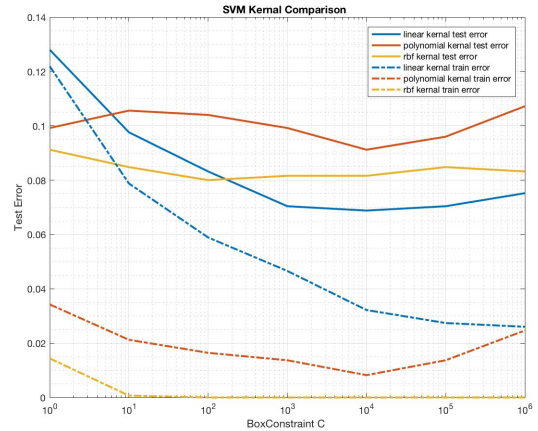


Fig. 6: SVM Model Comparison: training(solid line) and test(dashed line) error using linear kernel(blue), polynomial kernel(red), Gaussian radial basis function(rbf) kernel(yellow).

3) *Decision Tree*: Decision Tree predicts using a tree-like graph of decisions. However, according to the complexity of the data features, a single Decision Tree performs really badly. We ensemble decision tree models with Bagging and boosted method. We try to optimize its behavior with controlling the depth and width of the tree, number of split and number of trees. Results are shown in Fig 7. Both methods are able to give a better results than the kNN and SVM methods. We find the optimized algorithm is Bagged Trees with Number of trees bigger than 300 assembled using formula 2

$$\hat{f}^{bag}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{(b)}(x) \quad (2)$$

. To reduce the variance of bagged tree, we use random forest algorithm and consider 13 features out of 25 features on each step of splitting the tree. The best result so far at test error equals 5%.



Fig. 7: Tree Model Comparison: training(solid line) and test(dashed line) error using Ensemble Bagged Tree(blue), Ensemble Boosted Tree(red).

V. EVALUATION AND RESULT

The optimal model - ensemble Decision Tree with bagging method 21 features, achieves approximately 5 percent classification error for the four users. The confusion matrix and ROC curve of the optimal model is shown in Fig. 8, and Fig. 9

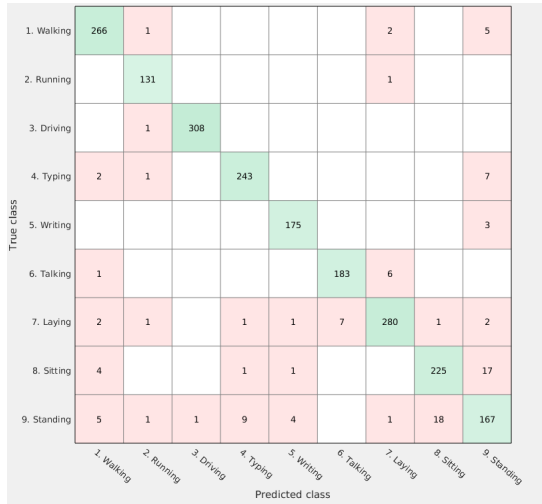


Fig. 8: Confusion Matrix of Ensemble Bagged Tree model, number of trees = 300, with cross validation.

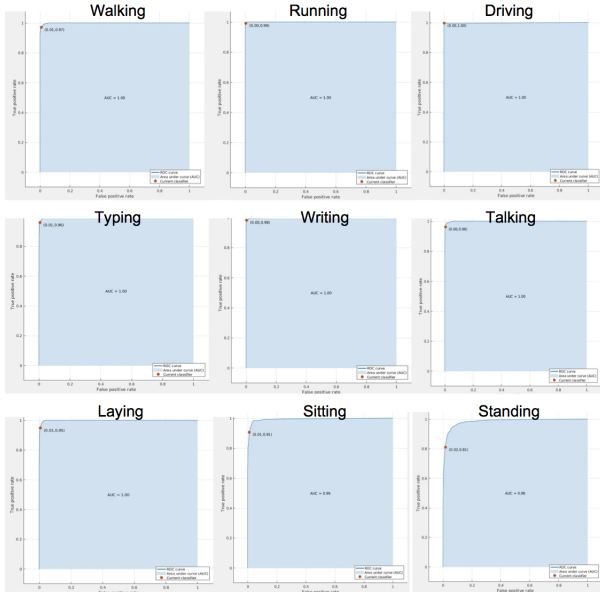


Fig. 9: ROC Curve of 9 activities.

From the Confusion Matrix and ROC, we can conclude that our learning algorithm separates most labeled activities well. The worst two performance classes, standing and sitting, have a true positive rate of 81% and 91%. Other classes all have true positive rates larger than 95%.

However, we also discover the following challenges:

1) Less intensive activities, sitting and standing are more

TABLE II: Model Comparison

Model	Test Error
Decision Tree, Bagged Random Forest	4.3%
Decision Tree, Gentle adaBoosted	6.6%
SVM, linear kernel	6.9%
SVM, polynomial kernel	9.1%
SVM, rbf kernel	8.0%
kNN, Euclidean distance	7.0%
kNN, Chebychev distance	9.3%
kNN, CityBlock distance	6.5%

difficult to differentiate. We think it is because these activities are similar in the view of sensors.

2) Standing has a high false negative, this might be due to lack of training data samples.

3) Typing and standing, walking and standing, and typing and laying are harder to classify from each other. This might be because of multiple activities are happening at the same time, but we only supports tagging one activity per data sample.

Here is a summary of all results that we use:

VI. CONCLUSION AND FUTURE WORK

The results presented above show that it is possible to identify user behavior while the users are holding phones in their hands. However, from this project, we noticed the following aspects require improvements:

1) Adding more data is still improving our test error on the margin. We would like to try to add more data to see how much this will improve the model.

2) Having a better definition of activities would improve this model. We would like to allow the possibility of simultaneous multiple activities tagging, since some of the activities can be done simultaneously.

3) Sensor data collection from different types of phones can be significantly different. The reason for this could be that the sensor calibration and precision are different for each phone. We end up using the data from the same phone.

4) There exists an activity pattern for each user. Using one user's training data to predict another user's behavior is performance worse than predict this user's behavior. Personalized model for each user might improve the prediction accuracy. We would like to solve these problem in the future.

VII. ACKNOWLEDGEMENTS

We thank Professor Andrew Ng, Professor John Duchi, and Teaching Assistant Rishabh Bhagava for their instructions.

REFERENCES

- [1] Shu Chen and Yan Huang, "Recognizing human activities from multi-modal sensors," Intelligence and Security Informatics, 2009. ISI '09. IEEE International Conference on, Dallas, TX, 2009, pp. 220-222.
- [2] Human Activity Recognition on Smartphones using a Multiclass Hardware-Friendly Support Vector Machine, Davide Anguita, et. al.
- [3] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L. Reyes-Ortiz. A Public Domain Dataset for Human Activity Recognition Using Smartphones. 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2013. Bruges, Belgium 24-26 April 2013.gnal Detection and Estimation. New York: Springer-Verlag, 1985, ch. 4.

- [4] A Study on Human Activity Recognition Using Accelerometer Data from Smartphones, Akram Bayat, Marc Pomplun, Duc A. Tran, MobiSPC, 2014.
- [5] Activity recognition from user-annotated acceleration data, Bao, Ling and Intille, Stephen S, Pervasive computing, 2004, Springer
- [6] Key Features of MATLAB Mobile, The MathWorks, Inc. <https://www.mathworks.com/products/matlab-mobile/features.html>