Human Activity Recognition Using Smartphone Internal Sensors

Yasaman Ghasem Pour and Chia-Yi Yeh

I. Abstract

Activity recognition has appealing use in healthcare applications such as monitoring the daily activity of elderly people or patients. Smartphones are inseparable from human life and thus the smartphone built-in sensors (e.g. gyroscope an accelerometer) can be reused for human activity recognition. In this project, we exploit features retrieved from the internal smartphone sensors and try several classifiers to identify human activities including walking and standing. We first perform preprocessing such as dimension reduction on our dataset, and then design parameters for several classifiers including neural network, K nearest neighbors (KNN), support vector machine (SVM), Dendogram-based SVM (DSVM), and random forest. The results of analysis show that random forest classifier and neural network perform the best. Also, we observe that static activities can be easily distinguished from dynamic activities. Furthermore, more features do not guarantee a better performance.

II. Introduction

Activity recognition (AR) aims to identify the actions carried out by a person. AR is of interest because it can be applied to many real-life problems such as eldercare and healthcare. The sensors which collect the data needs to be attached to the subject's body to allow continuous monitoring of numerous physiological signals. Nowadays we carry our smartphones almost everywhere and thus the smartphone built-in sensors such as gyroscope an accelerometer can provide the information of the user's movement. By analyzing the gyroscope and accelerometer signals, smartphones have the potential to help us recognize human activities.

In this project, we use a human activity recognition dataset in UCI repository [1]. The dataset is collected over 30 volunteers with Samsung Galaxy S II on their waist. The volunteers perform six activities: walking, walking upstairs, walking downstairs, sitting, standing, and laying. The 3-axial gyroscope and 3-axial accelerometer signals are captured at a sampling rate of 50 Hz. The experiment is videotaped and the data is labeled manually accordingly. The obtained dataset was randomly partitioned into two sets, where 70% of the volunteers were selected for generating the training data and the rest 30% for the test data. Instances for each class are approximately evenly distributed. Overall, there are 7352 instances in the training set, and 2947 instances in the test set.

Features are extracted from the raw 3-axial gyroscope and 3-axial accelerometer signals. The gyroscope and accelerometer signals are first passed through a low pass noise filter. Then the signals are sampled in fixed-width sliding windows of 2.56 seconds and 50% overlap. Since the sampling rate is 50 Hz, 2.56 seconds correspond to 128 readings in a window. The original signals were derived in time to obtain Jerk signals, and applied Fast Fourier Transform (FFT) to obtain frequency domain information. A set of variables such as mean, standard deviation, energy, and entropy are estimated from the time domain and the frequency domain signals. Finally, a vector of 561 features is obtained for each instance.

Our project goal is to classify the instances into the six activities. The classifiers used in this project include neural network, K nearest neighbors (KNN), support vector machine (SVM), Dendogram-based SVM (DSVM), and random forest. In the following sections, we first describe our dimensionality

reduction method. In section IV, we discuss how parameters of each classifier are chosen. Finally, we compare the results for all classifiers on test set.

III. Dimension Reduction

As described in the previous section, 561 attributes were obtained from the raw sensor signals for each instance. However, working with all 561 attributes is not easy since high dimensionality results in high computational cost in classification. Therefore, we first perform dimension reduction before we feed in the features to the classifiers. The method we use to reduce dimensionality is principal component analysis (PCA), which project the original features in the directions that have the highest variance.

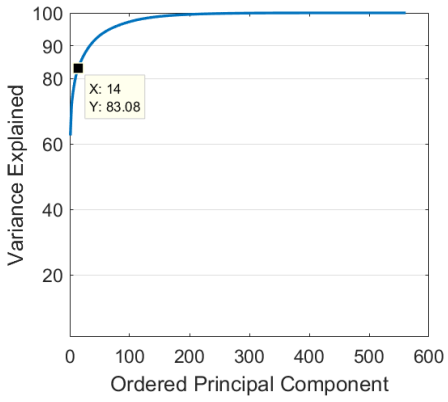


Fig. 1 Variance explained by the principle components

Fig. 1 shows the variance explained by the principle components. We see that the first few principle components capture most of the variance. The first component itself captures 63% of the variance. However, if we want to capture 99% of the variance, 155 components are needed, which still has high dimensionality.

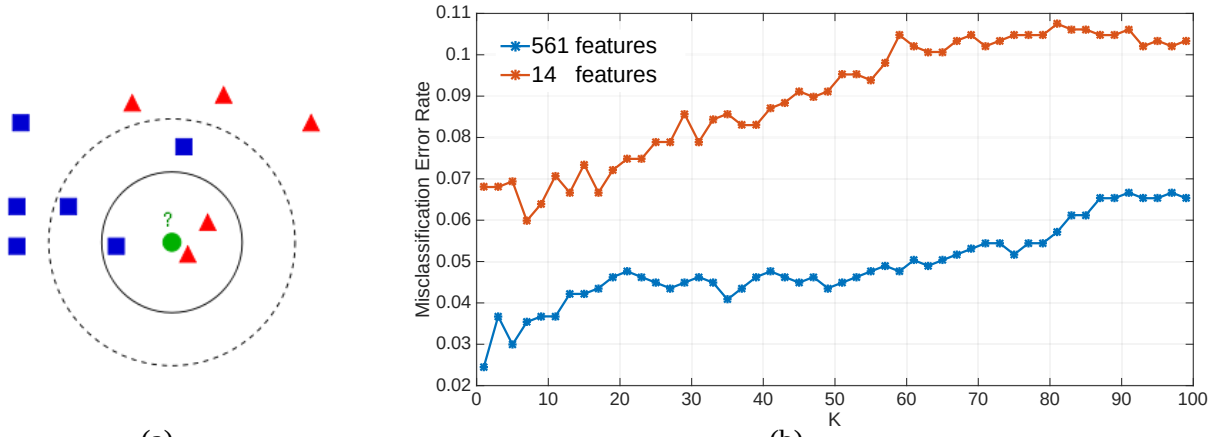
Since the variance explained by the ordered principle components decrease rapidly, we choose to use the first 14 components, where the 14th component explain only 1% of variance explained by the 1st component. That is, the 14th component has significantly less importance than the 1st component. The first 14 components together cover 83.08% of the variance. In the following section, we will feed in the 14 principle components to the classifiers. We will also discuss the impact of the reduced dimension in the results comparison section.

IV. Design Parameters for the Classifiers

Now that our data set has been preprocessed, we need to find key parameters of each intended classifier to ensure their optimal performance on our data set. To this end, we first divide our training set to two subsets using 10-fold cross-validation. The design parameters are chosen based on the performance of each classifier on the validation set. We use a common metric called “Misclassification Error Rate (MER)”, which is defined as the number of misclassified instances over the total number of instances. In other words, MER which is calculated for the validation set describes the performance of each model and helps us choose the optimal parameters.

1) K nearest neighbors (KNN)

KNN classifies a new instance into a certain class based on its K nearest neighbors, K is usually an odd number to avoid draws. The new instance is classified into class C if the majority of its neighbors are in class C. Fig. 2(a) is an illustration of KNN classifier, where the green circle is a new instance. For K=3, the new instance is classified into the red triangle, while for K=5, the majority class becomes the blue rectangle. We observe that the decision can be different when considering different numbers of neighbors, and therefore we need to find the best K for our dataset.



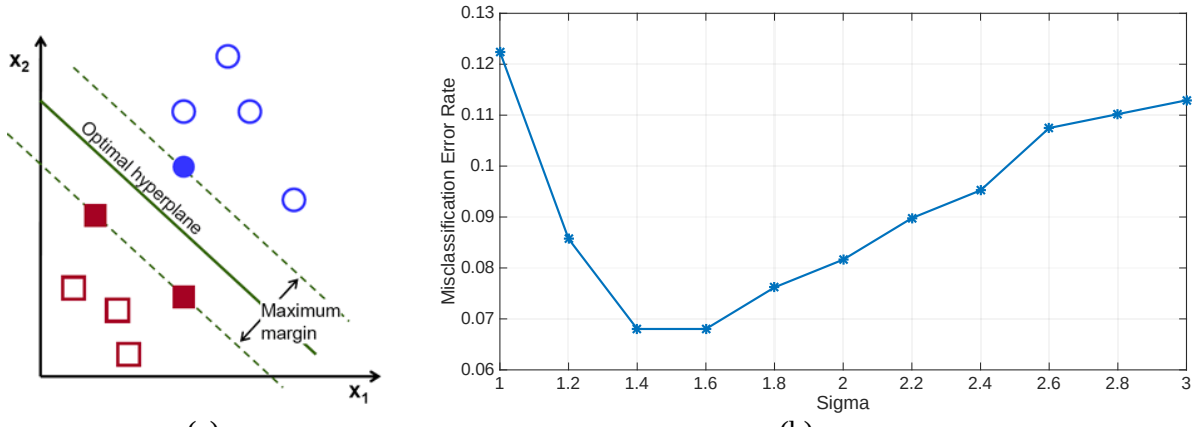
(a) (b)

Fig. 2 (a) An illustration of KNN classifier. (b) Validation set MER of KNN classifier with the number of neighbors (K) varies from 1 to 99.

Fig. 2(b) shows the validation set MER with various numbers of neighbors K. We observe that the MER is smaller with higher dimensional features, at the cost of higher computational complexity. The MER is minimized when considering 7 nearest neighbors in the case of 14 features, while in the case of 561 features, considering only the nearest neighbor is the optimal strategy. This implies that when we decrease the dimensionality, some information is lost and more neighbors need to be considered to have a better classification. We will choose K = 7 to apply to the test set in the next section.

2) SVM

SVM is originally a binary classifier that finds a hyperplane maximizing the margin between the two classes, as shown in Fig. 3(a). The extension of SVM to multiple classes is an on-going research. Generally speaking, there are two strategies: one-against-all and one-against-one. In the one-against-all strategy, we partition these K classes into two-class problems: one class contains instances of an intended



(a) (b)

Fig. 3 (a) An illustration of SVM classifier. (b) Validation set MER of SVM classifier using Gaussian kernel with different values of σ.

Kernel Function	MER on Validation Set
Linear	0.1973
Quadratic	0.1020
Polynomial	0.1088
Gaussian (σ = 1.4)	0.0680

Table. 1 MER on validation set with different kernels

class and the other contains the instances of all other classes. Then, we apply binary SVM for this new two-class problem. Therefore, for a new test instance, we need to perform K binary SVM and ideally, only one of the K classifiers will show a positive result and all other classifiers will show negative results, assigning the new instance to a unique class.

In the one-against-one method, we apply binary SVM for all K(K-1)/2 possible pairs of classes. A new test instance is then tested against these K(K-1)/2 classifiers and K(K-1)/2 scores (votes) are obtained. In a perfect case, the correct class will achieve the maximum possible vote which is K-1. Although this approach potentially performs better, it requires prohibitively expensive computation cost. Therefore, we choose the first strategy in this paper.

We apply different kernel functions including linear, quadratic, polynomial, and Gaussian. Table. 1 shows the validation set MER when we apply those kernel functions in SVM. Since Gaussian kernel has a parameter σ needed to be determined, we first search over a range of σ to find the σ that minimizes MER. Fig. 3(b) shows that when σ = 1.4 or σ = 1.6, the MER is minimized. Therefore, we choose σ = 1.4 for Gaussian kernel. Since Gaussian kernel with σ = 1.4 has the smallest validation set MER in Table. 1, Gaussian kernel with σ = 1.4 is chosen for the final evaluation on test set.

3) Dendogram-based SVM (DSVM)

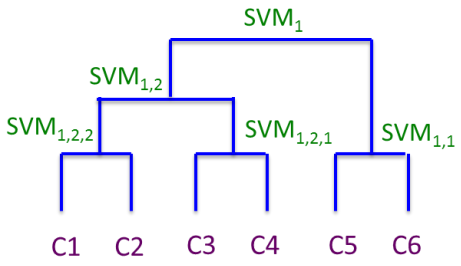
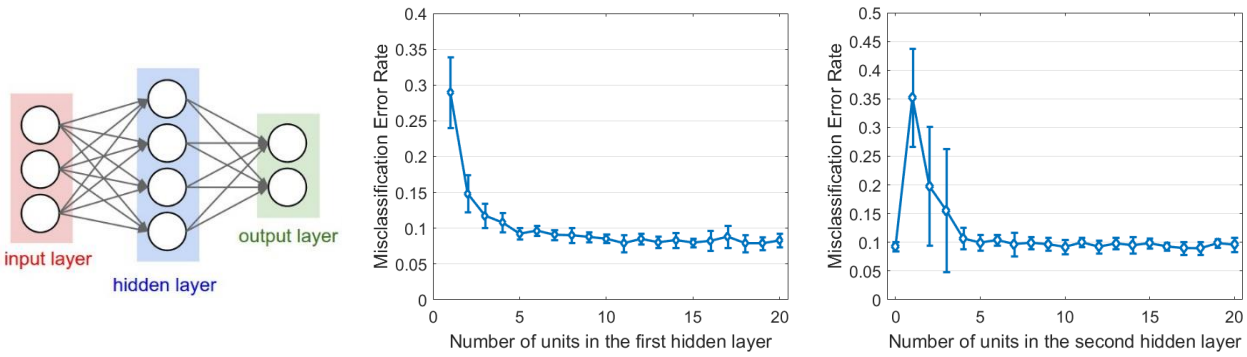


Fig. 4 An illustration of DSVM classifier

Dendogram-based SVM **(**DSVM) is an alternative method for multi-class SVM [2]. DSVM requires (K-1) SVMs for a K-class problem. The main idea of DSVM is that solving many small problems in a hierarchical way is preferable to solving a complex great problem. As illustrated in Fig. 4, there are two major steps in DSVM: (1) Calculating K gravity centers and perform hierarchical clustering, (2) Perform binary SVM in each node. Since DSVM is SVM with different grouping, the parameter for DSVM is the same as the parameter for SVM, which is Gaussian kernel with σ = 1.4.

4) Neural Network

Neural network is inspired by the neurons and synapses in human brains. Each neuron performs a simple task, yet when neurons are connected by synapses, they can accomplish complex tasks. Neural network consists of perceptrons and weights connecting the perceptrons, as illustrated in Fig. 5(a). Each perceptron, just like a neuron, performs a simple function. By assigning different weights in the network, the neural network can implement different functions. When there are more units in the hidden layers, or there are more hidden layers in the network, a more complex the classifier can be implemented. We need the classifier to have enough hidden units so that it is complex enough to successfully classify the instances. However, too many hidden layers or hidden units increase the computational complexity and



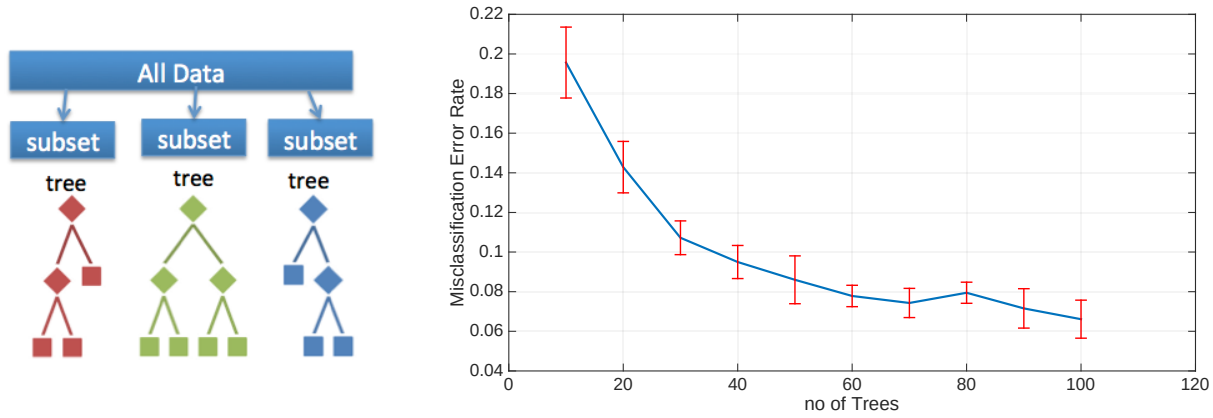
(a) (b) (c)

Fig. 5 (a) An illustration of neural network classifier. (b) Validation set MER of neural network with single hidden layer. (c) Validation set MER of neural network with two hidden layers, the first hidden layer is fixed to 5 units.

thus are undesirable. That is, we need to determine the number of hidden layers, and the number of units in each layer for our dataset.

To find a good network structure for our data set, we first try a single hidden layer neural network and varies the number of perceptrons in the hidden layer. As shown in Fig. 5(b), the validation set MER first decreases when there are more hidden units. However, the marginal benefit of adding an extra hidden unit diminishes after 5 units. We then explore the possibility of adding a second hidden layer given the first hidden layer has 5 units. The validation set MER of two hidden layers with 5 units in first layer is shown in Fig. 5(c). The first data point in Fig. 5(c) that has 0 unit in the second hidden layer corresponds to the case where there is only a single hidden layer with 5 perceptrons. We observe that using two hidden layers cannot achieve a lower MER than that with a single hidden layer, meaning that there is no benefit to add a second layer. Therefore, we will apply single hidden layer with 5 units on the test set.

5) Random Forest



(a) (b)

Fig. 6 (a) An illustration of random forest classifier. (b) Validation set MER of random forest with the number of trees in the forest varies from 10 to 100.

Random forest is a tree-based classfier which operates by constructing multiple decision trees, as illustrated in Fig. 6(a). Each decision tree is constructed based on a subset of data set chosen randomly. The number of instances in each subset is chosen to be the square root of total number of training instances, which is a common practice. For a new test instance, we first find the suggested class (or label) by each decision tree in our "forest". The final classification decision is made via voting.

A key design parameter of this classifier is the number of trees to be considered in the forest. We change the number of trees and observe the MER on the validation set (shown in Fig. 6(b)). We observe that increasing the number of trees first reduces MER significantly; however, after 60 trees, MER reductions is slow with increasing number of trees. We choose 100 trees for further evaluation on test set.

V. Results and Discussions

Once the design parameters for each classifier are determined, we analyze their performance on the test data. To this end, we first compare MER of different described classifiers and then dig into confusion matrix to obtain further information on classifiers' performance.

1) Comparing Different Classifiers

We apply the parameters we obtain for each classifier in the previous section to the test set. That is, 7 neighbors for KNN, Gaussian kernel with $\sigma = 1.4$ for SVM and DSVM, single hidden layer with 5 units for neural network, and 100 trees for random forest. Table. 2 shows the test set MER for different classifiers. In addition to the case of 14 features, we also consider the case of 155 features, which capture 99% of the variance, and the case of all 561 features.

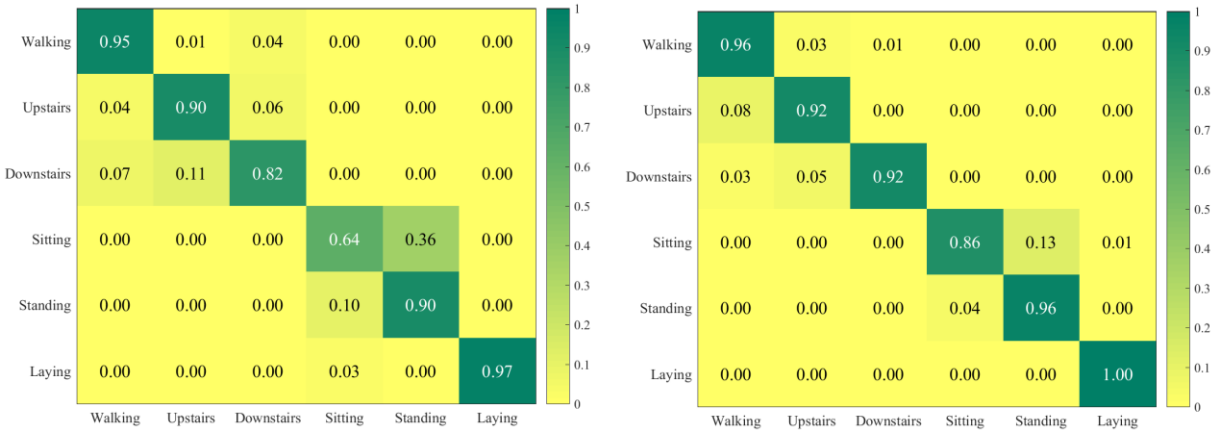
We see that random forest has the least MER when we use only the first 14 principle components. While in the case of 155 features and 561 features, neural network outperforms the other classifiers. We also observe that the MER does not necessarily decrease as the number of dimension increases. For example, MERs with 155 features are generally smaller than the MERs with all 561 features. One possible explanation is that the extra features may be junk features that cannot be generalized outside training set, and therefore hurt the MER.

Classifier	MER (14 Features)	MER (155 Features)	MER (561 Features)
KNN	0.1449	0.0903	0.0906
SVM	0.1564 (sigma = 1.4)	0.0628 (sigma = 15)	0.1286 (sigma = 50)
DSVM	0.1724	0.1522	0.1567
Neural Network	0.1429	0.0472	0.0621
Random Forest	0.1208	0.0869	0.1215

Table. 2 MER on test set for different classifiers with different numbers of features

2) Dig into Confusion Matrix

MER give us a general idea of how well a classifier performs, however, we cannot know the class-wise performance by looking at MER. In comparison, confusion matrix gives us the classification accuracy of each class, true positive and false positive values. While here we use neural network as an example, the findings can be generalized to other classifiers.



(a) (b)

Fig. 7 Confusion matrix for single layer neural network with 5 units. (a)14 features. (b) 561 features.

In the confusion matrix shown in Fig. 7, the x axis is the predicted activity, and the y axis is the labeled activity. For example, in the first row of Fig. 7(a), 95% of the walking instances are correctly classified, while 1% and 4% of the walking instances are classified as walking upstairs and walking downstairs, respectively. The diagonal of the confusion matrix represents the classification accuracy for each class.

The first observation is that the static activities are easily separable from the dynamic activities. As we can see in Fig.7, no static instance is misclassified as a dynamic instance, and vice versa, for both 14 features and 561 features.

The second observation is that laying usually has the highest classification accuracy. This can be because the user orientation of laying is very different from the other activities, and therefore the cluster of laying instances is far from the rest of the instances.

Finally, we observe that sitting and standing are not easily differentiable. Even we have all the 561 features, 13% of the sitting instances are misclassified as standing. When we have only 14 features, the two classes are even more non-separable. This is because sitting and standing have similar features, the cluster of sitting is closed to the cluster of standing. When dimension is reduced, the details that separate the two clusters are left out and cause misclassifications.

VI. Conclusion

In this project, we apply several classifiers on a smartphone human activity recognition dataset to identify human activities including walking, walking upstairs, walking downstairs, standing, sitting, and laying. The classifiers used in this project include KNN, SVM, Dendogram-based SVM, neural network, and random forest. We first preprocess the features to reduce its dimensionality. We then design the parameters for each classifier using cross-validation. Finally, we apply the classifiers on test set and find that: (i) random forest classifier and neural network have the best performance, (ii) static activities can be easily distinguished from dynamic activities, and (iii) more features do not guarantee a better performance.

References

[1] Anguita, Davide, et al. "A Public Domain Dataset for Human Activity Recognition using Smartphones." ESANN. 2013.

[2] Bennani, Younes, and Khalid Benabdeslem. "Dendogram-based SVM for multi-class classification." CIT. Journal of computing and information technology 14.4 (2006): 283-289.