User Activity Recognition basing on Mobile Devices

Definition

Project Overview

User Activity Recognition (UAR) is a research problem about identifying human movement state via carry-on sensor signals, which can assist take measures or supply services correspondingly. For example, we can achieve health monitoring via analyzing the amount of activities in daily life. Besides, in area of context awareness, UAR can be combined with other related information of the user's surroundings to infer the user's behavior motivation and provide the convenient service initiatively. All above have positive effects on improvement of people' s living standard. UAR is an interdisciplinary research relating to machine learning, data mining, signal processing and so on. The basic procedure includes collecting raw data from sensors, data cleaning and preprocessing, feature extraction, and finally training models for classification or prediction.

In this project, I would follow the basic procedure mentioned above and finally created an Android application capable of recognizing the activities of the user who is carrying the device in real-time sensing mode and recording user activities for a long time to generate the traces of different activity of that day in history trace mode. The application uses a classifier trained offline using the dataset collected by eight volunteers. Since the project is related to my job, the dataset will not be accessible by now, but I would introduce the method of data acquisition principles and the data structure.

Problem Statement

The problem we want to solve is collecting time series sensor data from accelerator, gyroscope, magnetometer and etc. built inside phones, extracting time-domain and frequency-domain features and finally achieving analyzing users' movement state, including sedentary, walking, running, biking, in car and in train.

In order to avoid the interference caused by the diversification of hands, we require smart phones to be placed in pants when collecting data, so that sensor signals can express speed change and various rotation angle of human bodies. The tasks involved are as follows:

1. Data acquisition and preprocessing
2. Feature extraction and model training
3. Evaluate the result and refinement
4. Make the classifier run on Android

The final application is expected to be useful for outputting real-time estimation of user activities, and generating the history statistic of the sensed activities of a day as well.

Metrics

The following concepts are given before defining the evaluation criteria:

1. TP_i(true positive): numbers of samples which is predicted to be C_i class, and actually belongs to C_i class;
2. FP_i(false positive): numbers of samples which is predicted to be C_i class, but actually not belongs to C_i class;
3. TN_i(true negative): numbers of samples which is predicted to be other classes except for C_i, and actually not belongs to C_i class;
4. FN_i(false negative): numbers of samples which is predicted to be other classes except for C_i, but actually belongs to C_i class;

Basing on the concepts above, the overall accuracy is used as the evaluation index for the performance of models, the definition is as follows:

$$\text{Overall accuracy} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} (TP_i + FN_i)}$$

Overall accuracy indicates the proportion of correctly predicted sample number to total number, which can reflect the overall classification ability of the model.

Analysis

Data Exploration

The input consists of time-series sensor data from accelerator, gyroscope and magnetometer built inside phones. Fig.1. shows examples of input data:

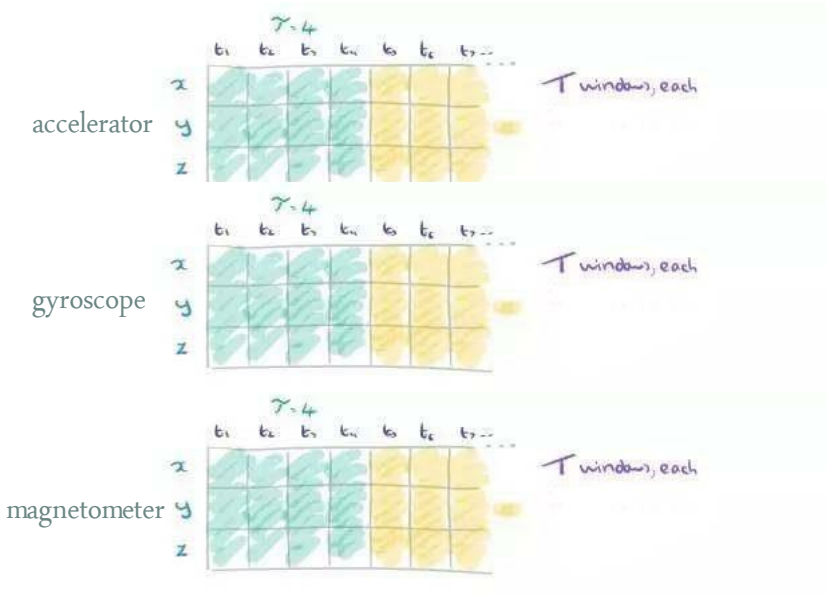

Fig.1.Time-series sensor signals

The sensors include three axes: x, y, z. We segment sensor signals via sliding window, which is later used for feature extraction. Finally follows classification model training.

In this project, we developed an application installed on each phone to collect labeled sensor data at some rate (e.g.20Hz), and then send data array to remote storage on distributed clusters. The data structure is in JSON format:


```
[
    {
        "IMEI": "860988033040002",
        "activity": "SEDENTARY",
        "time": 1492315086578,
        "accelerometer": {
            "x_Axis": 1.468811,
            "y_Axis": -9.157242,
            "z_Axis": 2.3529816,
        },
        "gyroscope": {
            "x_Axis": -0.0028839111,
            "y_Axis": 0.020584106,
            "z_Axis": -0.009613037,
        },
```

```
        "magnetometer": {
            "x_Axis": 33.2016,
            "y_Axis": 31.707764,
            "z_Axis": 8.891296,
        }
    },...
]
```


There are about 90000 samples for each class, in other words, we have 540000 samples in total.

Exploratory Visualization

Firstly, we plot the time-series sensor data to guarantee the six activities can be classified intuitively. Fig.2. shows one segment of each class. The x-axis represents the sequence number of data; the y-axis is the amplitude of $\sqrt{x\_axis^2 + y\_axis^2 + z\_axis^2}$ for accelerator.

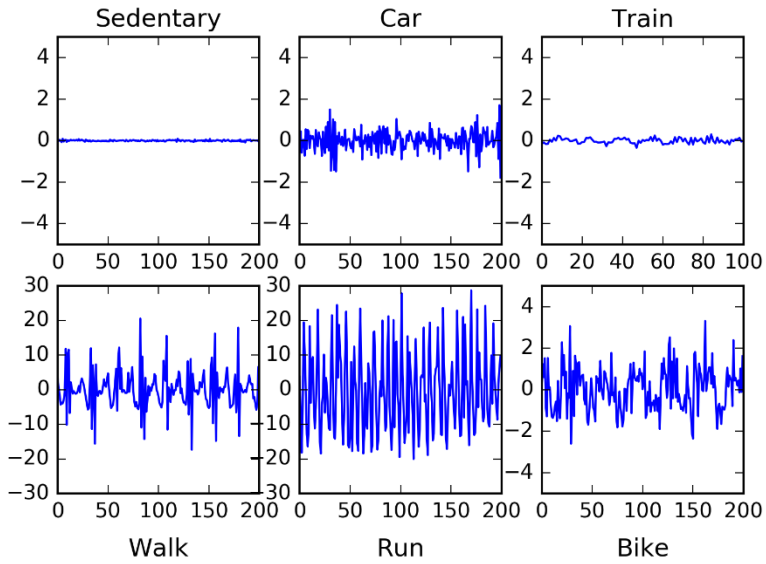

Fig.2. One segment of time-series sensor signals for six activities

As Fig.2 shows, there are notable differences among each class. Besides, in the procedure of data checking, we found that it is impossible to avoid the inaccuracy of the sensor data due to human manipulation. For example, Fig.3 shows the sensor data, which are supposed to belong to walking class, was wrongfully marked with sedentary label. The definition of each axis is the same with Fig.2. In this case, we solved the problem by filtering out the data, which are notably different from others in visual form.

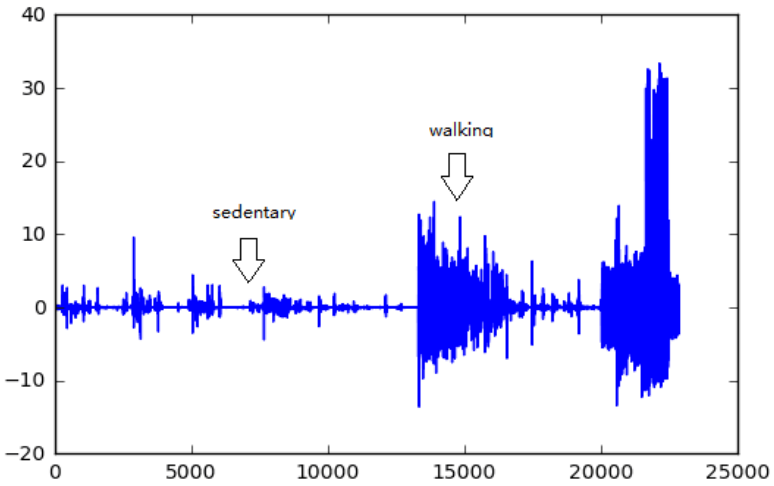

Fig.3. Marking sensor data with wrong labels

Algorithms and Techniques

The classifier we choose is Random Forest, which has advantageous properties such as high generalization performance, reduction in overfitting and insensitive to outliers. Since we would like our model to be feasible for any user on condition that the training dataset only come from 8 volunteers. On the other side, there can always be some inevitable outliers. Random Forest can outperform other algorithms such as SVM, GBDT in the case mentioned above.

The following parameters can be tuned to optimize the classifier, which refer to APIs of scikit-learn:

- Training parameters
  - n_estimators (the number of trees in the forest)
  - criterion (the function to measure the quality of a split, such as “gini”, “entropy”)
  - max_depth (the maximum depth of the tree)
  - min_samples_split (the minimum number of samples required to split an internal node)
  - min_samples_leaf (the minimum number of samples required to be at a leaf node)
  - oob_score (whether to use out-of-bag samples to estimate the generalization accuracy)
- Preprocessing parameters (see the Data Preprocessing section)

Benchmark

We took Activity Recognition of Intel Context Sensing SDK[1] as benchmark. Because they just describe usage scenarios and instructions for use, the dataset and relative techniques are not provided, we can only compare with them via validation accuracy and application effect of the model.

Methodology

Data Preprocessing

In this section, we segmented time series data into discrete sliding window, as Fig.4 shows. For example, sorting sensor data of movement $state_j$ for $user_i$ by time; Segmenting time series sensor signals via a sliding window of size w (e.g.200), and sliding step size is s (e.g.64). The window data is used for feature extraction for the next module. All above are the same for other users and activities.

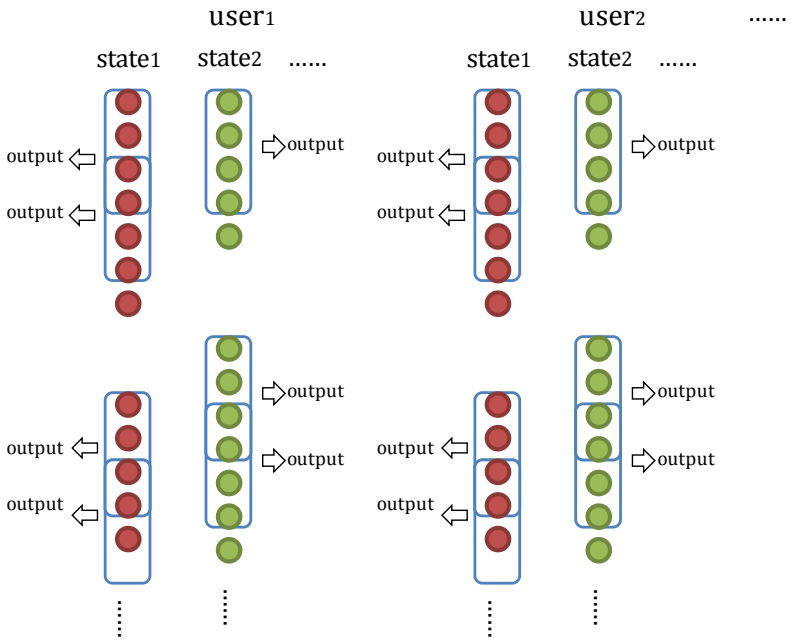

Fig.4.Segmenting time-series sensor data via sliding window

The data format is as follows:

user_id<tab>activity_type<tab>timestamp<tab>accelerator_x<tab>accelerator_y<tab>accelerator_z<tab>gyroscope_x<tab>gyroscope_y<tab>gyroscope_z<tab>magnetometer_x<tab>magnetometer_y<tab> magnetometer<tab>window_index

Implementation

The implementation process can be split into three main stages:

1. Feature extraction stage
2. The classifier training stage
3. The application development stage

Table1 offers lists of measures for computing feature vectors of each column of each window. We also applied smoothing filter beforehand, so that the "characteristic" acceleration of phones in three dimensions can be as close to human movement as possible.

Table1: List of measures for computing feature vectors [2]

Function	Description	Formulation
Mean(**w**)	Arithmetic mean	$\overline{w}=\frac{1}{N}\sum_{i=1}^{N} w_i$
Std (**w**)	Standard deviation	$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(w_i-\overline{w})^2}$
MathQ1(**w**)	The first quartile	Q1(**w**)
MathQ3(**w**)	The third quartile	Q3(**w**)
Diff	Interquartile range	Q3−Q1
Energy(**w**)	Average sum of the squares	$\frac{1}{N}\sum_{i=1}^{N} w_i^2$
Max(**w**)	Largest value in array	$\max_i(w_i)$
Min(**w**)	Smallest value in array	$\min_i(w_i)$
Range(**w**)	Range between maximum and minimum	$\max_i(w_i)-\min_i(w_i)$
MAE(**w**)	Mean of absolute deviation	$\frac{1}{N}\sum_{i=1}^{N}\lvert w_i-\overline{w}\rvert$
Correlation(w_1, w_2)	Correlation coefficient between two signals	$C_{1,2}/\sqrt{C_{1,1}C_{2,2}}$, C=cov($w_1, w_2$)
JumpFall(**w**)	Largest continuous rise height and descent slope	$\max(\sum w_{i+1}-w_i)$, where $w_{i+1} > w_i$ $\max(\sum w_i - w_{i+1})$, where $w_i > w_{i+1}$
FFT(**w**)	Flourier transformation	$F(w)=\mathcal{F}[f(t)]=\int_{-\infty}^{\infty} f(t)e^{-iwt}dt$ Take top3 dominant frequencies and corresponding amplitudes.

After feature extraction, we got 127 features for each sample window, and then we divided the original collection of feature vectors into training set and testing set, and trained the classifier on the former. Since we wanted our model file to be available on Android system, so we mainly developed the application using Java, and only made some experiments on Python.

Basing on some APIs from Weka [3][4], we built the classifier, and serialized it into a file so that it can be reloaded on mobile terminals, below are the key implement functions, for more details, see WekaClassifier.java:

a. loadTrainSet(…): Loads training dataset from arff file

b. buildClassifierModel(…): Trains the model, it supports define model structure with the optional parameter *option*

c. saveModelFile(…): Serializes the model into a file

d. loadModelFile(…): Deserializes the model into an object

e. evaluateFromFile(…): Tests the model with labeled dataset in arff file and outputs accuracy

f. classifyInstance(…): Predicts the result of one sample feature

During the application development stage, we started some threads to collect sensor data all the time. Once we got a sample window, we extracted features and then predicted the label with the model trained offline.

Refinement

Considering both computational efficiency and power consumption, we mainly refined the algorithm from two sides:

- ✧ Streamline the code structure to reduce redundant loops, especially in the feature extraction module
- ✧ Select feature subset to decrease the complexity of the model.

The latter was done in a Jupyter Notebook, for more details, see file feature_selection_for_UAR.ipynb, the result was shown in Fig.5, the x-axis and the y-axis respectively represent the index and the degree of importance for each feature.

We can see that there is a “Long Tail” in Fig.5. Balancing efficiency and accuracy, we reserved the features whose importance score are above 0.001. Therefore, we finally get 78 features.

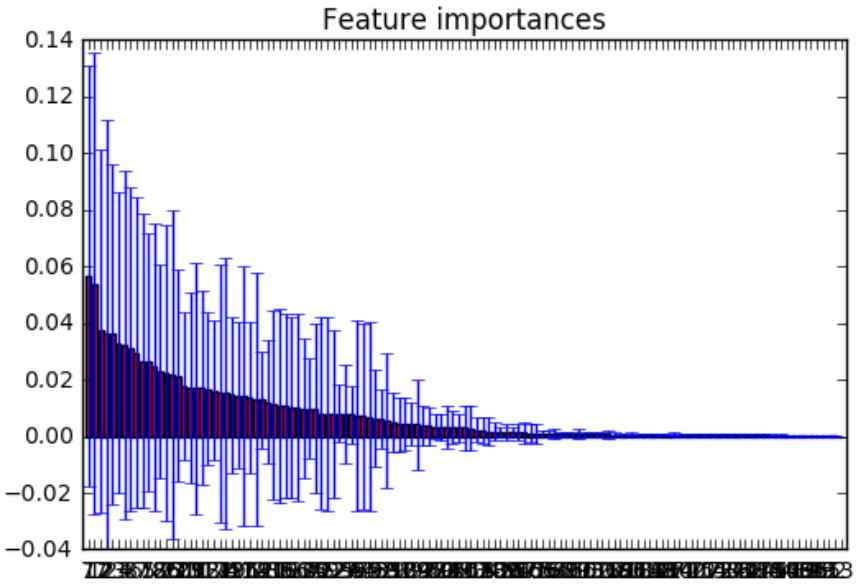

Fig.5. Outcomes of feature selection

Results

Model Evaluation and Validation

During model training stage, we split the dataset into two parts to help adjust parameters. Since we wanted to make full use of the collected data, we finally trained the model with the whole dataset. Table2 offers the confusion matrix; the last two columns show accuracy and precision for each class.

Table2: Classification results for user activity recognition

Confusion Matrix	sedentary	incar	running	walking	intrain	biking	accuracy	precision
sedentary	3541	33	0	0	247	79	90.79%	96.85%
incar	19	5411	0	5	107	98	95.93%	96.37%
running	0	0	3663	197	0	0	94.89%	96.37%
walking	34	1	138	5309	18	140	94.13%	91.14%
intrain	62	126	0	4	3642	146	91.51%	90.30%
biking	0	44	0	310	19	4687	92.63%	91.01%

The total accuracy is 93.49%. We had 100 trees with max depth of 15. When it comes to applying the model on Android phones, we selected 78 features (see the Refinement section), and trained with 50 trees with max depth of eight, the total accuracy decreased to 91.67%. However, it does not make any difference in practical application scene.

Justification

User activity recognition in real-time mode is not so meaningful in our daily

life. Besides, there can always be some wrong predictions. So we transformed to the history trace mode, which records user activities for a long time, and returns the traces of different activity of that day. By this means, we can help users get to know the amount of activities in their daily life. The result was improved upon by using the following techniques:

- ✧ The algorithm outputs an assigned probability for each class; we preset a threshold for each class to reduce the number of false positives. If the probability is less than the threshold, we output label “unknown”.
- ✧ We merged and smoothed the time-series results with a preset window (e.g.3min). The output label for each window is the one whose number of occurrences is the maximum and larger than forty percent of the window length.

The results are in JSON format as below:

```
[
    {
        "userId": "",
        "deviceId": "",
        "activityId": "",
        "startTime": "",
        "endTime": ""
        "duration": ""
    },……
]
```

Conclusion

Free-Form Visualization

Fig.6 and Fig.7 are the screenshots of real time mode and history trace mode respectively.

In Fig.6, The top icon shows the current activity of the user. It is updated by taking the activity of the biggest probability. The bottom statistics are calculated by adding up the time of different activities.

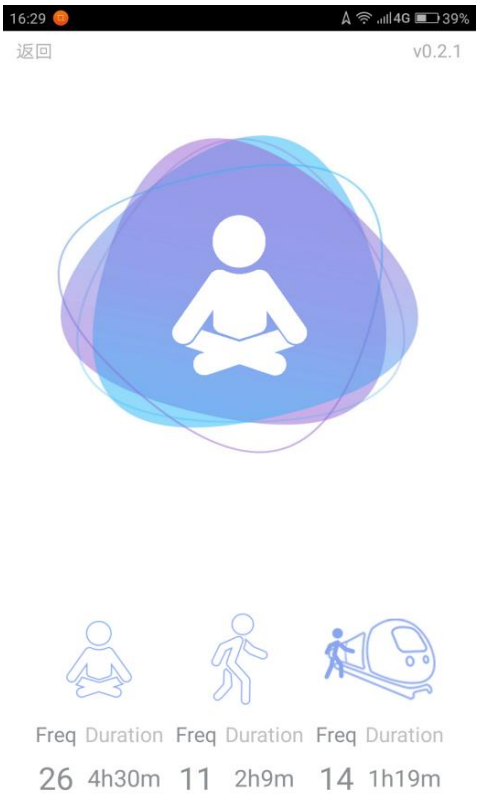

Fig.6. Screenshot of real time mode

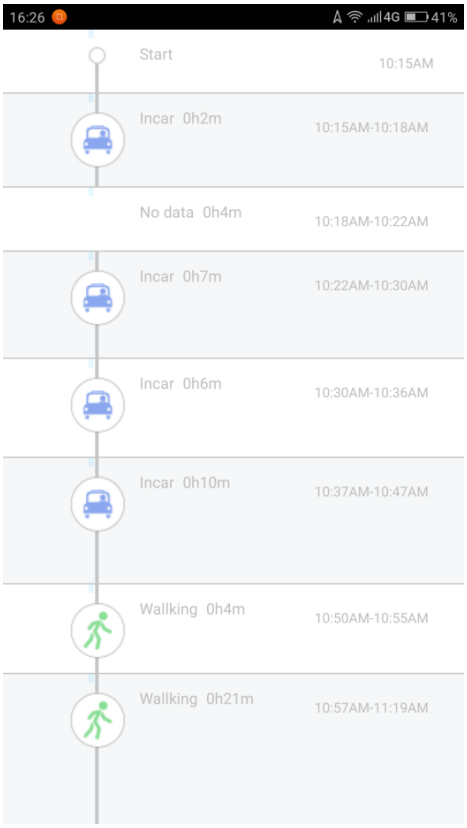

Fig.7. Screenshot of history trace mode

In Fig. 7, the screenshot shows the history statistic of the sensed activities of a day. The trace is generated from the JSON file: The icon is chosen by the activityId. The start time and stop time of the activity are set by startTime and endTime respectively. The duration of the activity is set by duration. One element correspond to a section, including activity, start time, end time and duration. If time difference between the endTime of the earlier section and the startTime of the latter section is more than 3min, a "No data" section is inserted between the activities in the demo.

This function could be used to record user's daily life and exercise automatically, and it can provide follow up service accordingly.

Reflection

In the model training stage, it is important to find the appropriate features and the model. It may take us a long time to make improvement repeatedly.

The applicable scene is limited to the scenario that the phone is placed in the pants, the clothes or in the bag, because of the limitation of training

dataset.

Improvement

We should continue to collect sensor data to increase the amount of training dataset. The transient period where the user activity changes from one class to another is not classified as accurate as the rest parts. Therefore, in the next stage we shall add “random” label to enable proper handle for the scenes that have not been met.

References

[1] https://software.intel.com/en-us/documentation/context-sensing-sdk-for-android-states-datasheet

[2] Ortiz J L R. Smartphone-Based Human Activity Recognition [M]. Springer International Publishing, 2015.

[3] https://www.cs.waikato.ac.nz/ml/weka/

[4] http://www.dice4dm.com/

--The End