MotionSense

Detecting Mobile Movement with Smartphone Sensors

Data Science Workshop 2018

Ido Calman [308353499]

Ofri Kleinfeld [302893680]

Shachar Hirshberg [305624207]

Part 0: Overview

Our main goal is to analyze how analytical data from mobile sensors can be used to identify what a user is actually doing in real-life. We first downloaded a dataset from Kaggle and chose the best performing model. Then, we gathered our own data and checked if the model still evaluates well. Next, we developed our own mobile app to see our model in action. Finally, we used a boosting method to improve real life predictions. Our **point of focus** here is to demonstrate the whole cycle: How a data science analysis of an outsourced dataset turns into a working product.

Part 1: The Dataset

Our dataset is retrieved from Kaggle, and is consisted of experiments, where participants were performing different kinds of activities around Queen Mary University of London's Mile End campus. The participants used smartphones sensors to retrieve analytical records of their activities.

- This dataset includes time-series data generated by accelerometer and gyroscope sensors
- It is collected with an iPhone 6s kept in the participant's front pocket
- A total of 24 participants performed 6 activities in 15 trials in the same environment and conditions.
- The activities: downstairs, upstairs, walking, jogging, sitting, and standing

Understanding The Features

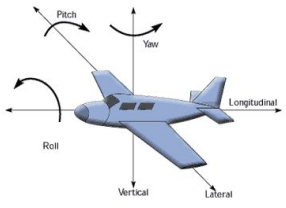
Accelerometer & Gyroscope

Accelerometers handle axis-based motion sensing. The sensor's goal is to **figure out how a phone is moving** and **which direction** it's pointing in. The gyroscope helps the accelerometer out with understanding which direction a phone is **orientated**.

Original Features

1. **Attitude -** *Roll*, *Pitch* and *Yaw*

   Attitude provides information about an object's orientation with respect to the local level frame (horizontal plane) and true north.



2. **Gravity & User Acceleration**

   The accelerometer sensor in fact measures the sum of two acceleration vectors: gravity and user acceleration. In our data, there are two separate features - the user acceleration and gravity acceleration.

3. **Rotation Rate**

   Using the Gyroscope, this feature measures the how much the phone is rotated around a certain axis (x,y,z).

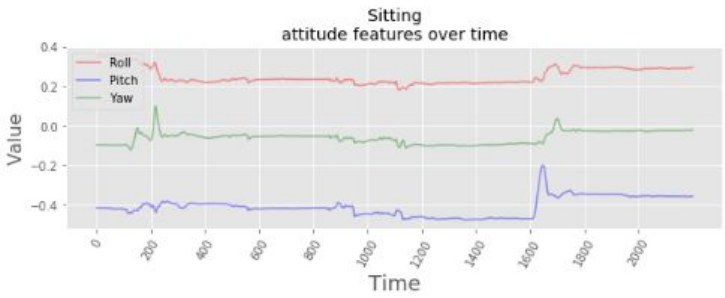
Part 2: Dataset Analysis

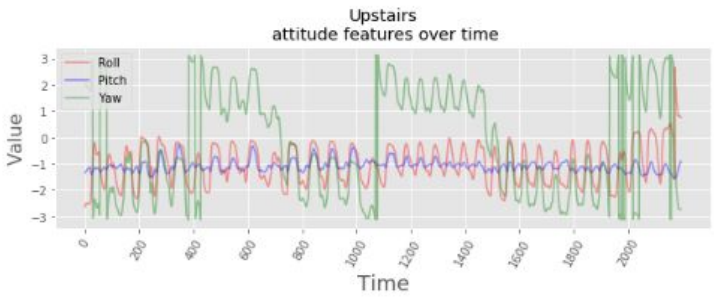
Exploration & Visualization

Our main goal here is to visually understand if there exists some correlation between the labels and our analytical features. After glimpsing on our data, we see that each data point is a vector $p \in R^{12}$ which determines a static point in time, which does not make a lot of sense when thinking about a continuous activity.

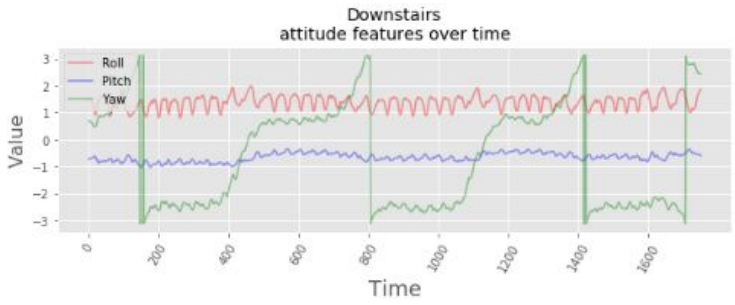
Analyzing Attitude Features

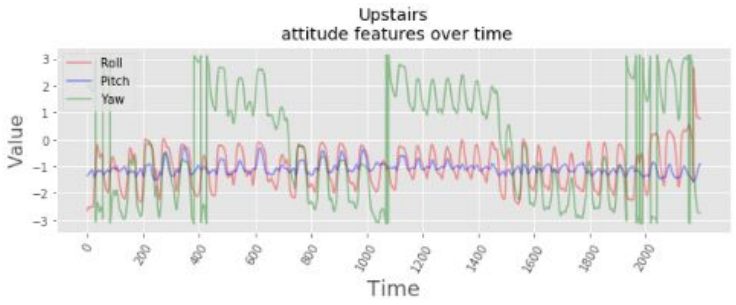
Next thing we would like to do is to figure out how our data changes in time. We pick one feature (i.e. attitude) and observe the correlation between the feature change in time over the three axes (*roll*, *pitch*, *yaw*) and it's labeled activity. We can see a great distinction between the attitude features values and trends between the two activities. the Attitude Yaw and Roll features fluctuates over time when climbing up-stairs but relatively stable while sitting. Also their actual value ranges quite differently between the two activities.





Next, we will try and see if we can get such a great distinction between two more related activities. We will still visualize the same Attitude features:

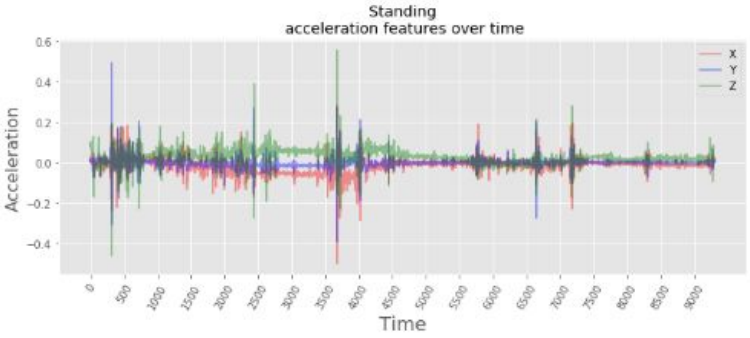


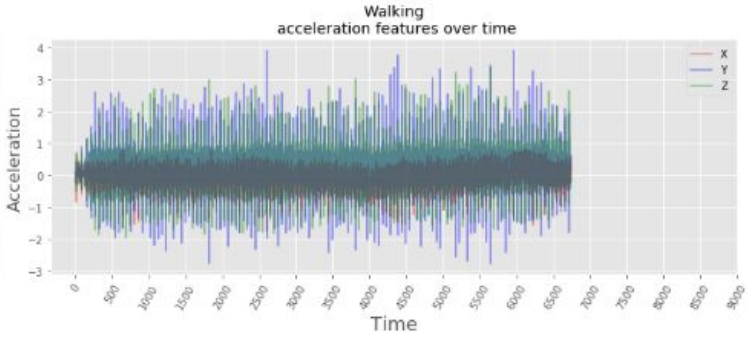


The distinction is not as great as in the last example, but still we can see magnitude differences in the Roll feature. Although while going downstairs the Yaw feature also fluctuates, we can still see different fluctuation trends between the two activities.

Analyzing Acceleration Features

We will now examine changes in the Acceleration features through time over its three axes (x,y,z) between different labeled activities. We will use participant #12 and compare the activities "standing" and "walking":





As we might have expected, we can see a great distinction between these two activities, especially in the magnitude of all the features and on the y-axis magnitude and fluctuation. (For more data visualizations, please see **Notebook #1**).

Problem Formulation

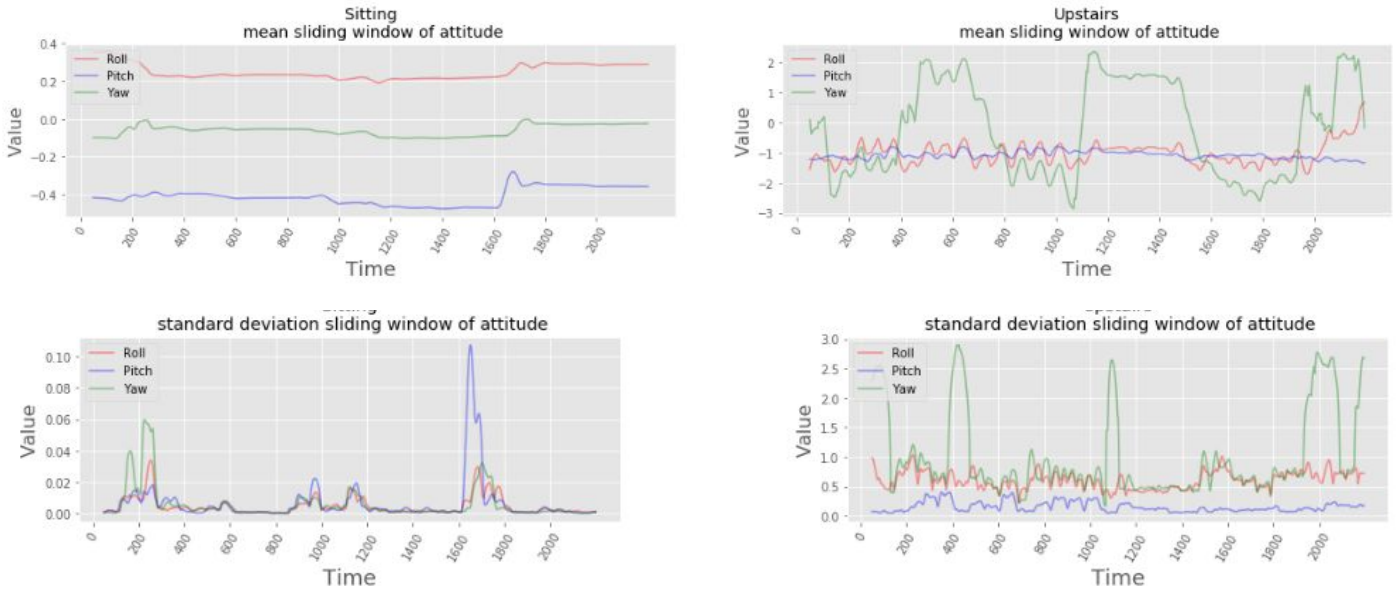
The above leads us to our precise problem formulation: **predict user's activity using smartphone sensors data**.

Part 3: Feature Engineering

Cleaning Outliers Using Data Smoothing

Manually identifying outliers in time series data, especially such that is originated from sensitive devices is quite problematic. Since our data is naturally fluctuated, it is hard to identify which entries are 'authentic outliers'' which we would like to learn, and which ones are just noise that would negatively affect our model generalization error.

One way to overcome the aforementioned problem is to smooth the data. This method allows us to aggregate our data over a number of observations together. First, we define a window size $w \in \mathbb{N}$ which determines the number of preceding observations to take into account while aggregating our current timestamp. Below are visualizations for the attitude features, with $w = 50$ with respect to the **mean** and **standard deviation** functions:



The aggregated version of the data looks visually more identifiable, which led us to the conclusion that we should learn the new features instead of the originals. We will next present a more generalized idea which we call "**Sliding Window**".

Feature Encoding

Recall that our data is a time series, that is, a sequence of measurements over time. Thus, extracting value for a single data point depends on its context. Classic ML classifiers predict output for a single input data point, independently to an adjacent input data point. So, in order to use our data to train classic ML model we will have to encode our features to represent context data.

Sliding Window

In this method, we will encode each data sample as a concatenation of analytical functions calculated over a predefined size of previous samples. This allows us to represent each entry together with its context.

Let $s \in \mathbb{R}^d$ be a sliding window single input, and define which analytical functions we want to aggregate by. In our case, we chose the functions F to be: [mean, median, std, min, max, sum]. Denote the number of our original features n Then we can calculate d : $d = n \cdot \mid F \mid +1 = 12 \cdot 6 + 1 = 73$. Using sliding window is also better for **preserving a large amount of data** after aggregating (See **Notebook #2** for more).

Raw History Encoding

In this method, we will simply encode each data point as a concatenation of the original features of its previous w (window size) data points. Let $r \in \mathbb{R}^d$ be a raw history single input. Then $d = (w+1) \cdot n + 1 = (10+1) \cdot 12 + 1 = 133$ for $w = 10$ Note that in this encoding d is linearly dependent on w .

Part 4: ML Algorithms & Statistical Evaluation

Evaluation Metrics

Our evaluation metrics will be **precision**, **recall** and their harmonic average the **F1 score**. These metrics are much more relevant to our problem compared to the model total accuracy for few reasons:

- Our problem is **imbalanced**. The labels walk sit and stand are x2 time more frequent than going up/down stairs.
- We also consider our assumption that some activities will be much **harder** to predict compared to others. i.e separating "sit" from "walk" should be much easier than separating between "upstairs" and "downstairs".
- That is why we'll be interested in the model performance for each activity by its own.

Linear Model: Multilabel Logistic Regression

We will start with a simple linear model and evaluate its performance using our two different encodings. We will use logistic regression with L2 loss function (MSE).

	precision	recall	f1-score	support
wlk	0.61	0.82	0.70	34613
sit	0.99	0.99	0.99	33674
std	0.97	0.98	0.98	30465
ups	0.59	0.44	0.50	15697
jog	0.85	0.83	0.84	13496
dws	0.50	0.21	0.30	12981
avg / total	0.79	0.80	0.78	140926

Sliding Window

	precision	recall	f1-score	support
wlk	0.48	0.68	0.56	34050
sit	0.87	0.96	0.91	33932
std	0.55	0.76	0.64	30745
ups	0.45	0.17	0.25	15631
jog	0.54	0.10	0.17	13509
dws	0.45	0.14	0.21	13059
avg / total	0.59	0.60	0.56	140926

Raw History

The results of the same classifier with the raw history encoding are much worse compared to the sliding window encoding - even with the relatively "easy" to predict activities ‘sit’ and ‘stand’. This leads to a conclusion that the data is **not linearly separable** with the raw history encoding, and might be even the same with the sliding window aggregation encoding.

Non-Linear Model: Random Forest

We will evaluate the performance of a non-linear random forest model over our two different encoding datasets. This model can be trained much **faster** due to options to parallelize the training of independent trees. Below is the optimal (See **Notebook #2**) hyper parameterized results for sliding window and raw history:

	precision	recall	f1-score	support
wlk	0.98	0.99	0.99	34613
sit	1.00	1.00	1.00	33674
std	1.00	1.00	1.00	30465
ups	0.97	0.96	0.97	15697
jog	0.99	0.99	0.99	13496
dws	0.97	0.96	0.97	12981

	precision	recall	f1-score	support
wlk	0.96	0.99	0.97	34050
sit	1.00	1.00	1.00	33932
std	1.00	1.00	1.00	30745
ups	0.95	0.92	0.94	15631
jog	0.99	0.97	0.98	13509
dws	0.95	0.92	0.93	13059

Sliding Window Raw History

Problems and the Need for 'Real World' Data

As we can see, even on our left aside test data the performance of the model is **too good to be true**.

- This is a reason to suspect that although we tested our model on data that was not used to train it, we are over-fitting to the current experiment setup.
- Perhaps the generation process of the data was too "synthetic" , not representing "real world" data obtained from phone sensors.
- We used this experiment framework to extract real data obtained from our activities during the day and labeled them accordingly. (Next section)

'Real World' Data Evaluation

We use the best trained model so far, which is the **Random Forest** over Sliding Window encoding. Below are the results over our **manually collected** data:

	precision	recall	f1-score	support
wlk	0.65	0.48	0.55	52652
sit	0.98	0.69	0.81	35225
std	0.96	0.96	0.96	36561
ups	0.40	0.74	0.52	21800
jog	0.00	0.00	0.00	0
dws	0.40	0.45	0.42	19547
avg / total	0.72	0.66	0.68	165785

As predicted, the results on real world data are much worse compared to the results over our original test set. Feed forward NN did not perform much better (See **Notebook #3**) than random forest, and adding another layer did not improve much the F1-score. We decided to deal with the real world data results gap with an **applicative** solution.

Part 5: iOS Application & Applicative Boosting

Our next step was to think about the actual use case of our project. In order to examine how our models behave in the wild. We decided to develop our own iOS application, watch where we fail to predict an activity, and improve our estimations accordingly.

iOS App Development

We decided to build our app using Swift. One of our challenges there was importing the model we created and trained in Python to our app, and using it in it - we used CoreML to do so. After having the model in place, we implemented real-time majority voter in the app to boost our precision. Finally, we had many product decisions to make during this process, such as: How frequently should we show predictions, how to visually conduct a session, etc. Our two model options, as mentioned above, were **Random Forest** and **FFNN.** Since our results were quite unnoticeable we chose **RF** because it is a simpler model complexity-wise, and it provides the ability to predict in parallel.

Predictions Boosting

After observing our app performance in the 'real world', we noticed that we do not need to predict in short time intervals, but rather a reasonable choice would be to predict every **2-3 seconds**. In order to boost our predictions we faced a **trade-off** in choosing the window size w : On the one hand, smaller window size would enable us to boost over more model predictions, on the other hand, increasing window size improves each prediction accuracy.

Example:

- $w = 20$, prediction every 0.2 seconds, with boosted decision every 3 seconds.
- We take a **majority vote** of 15 model predictions.
- We can choose from 6 different labels - so label with **at least 3** votes will be chosen.
- Our worse label prediction is "upstairs" with precision rate 0.37
- Upper bound (upstairs predicted 3 times) : $precision = 1 - (0.63)^3 \approx 0.75$

Results:

	precision	recall	f1-score	support
wlk	0.91	0.62	0.74	52652
sit	0.99	0.70	0.82	35225
std	0.97	0.97	0.97	36561
ups	0.50	0.89	0.64	21800
jog	0.00	0.00	0.00	0
dws	0.40	0.60	0.48	19547
avg / total	0.83	0.75	0.76	165785

Part 6: Appendix

Project GitHub Repository

Notebook #1 - Business Understanding and Data Exploration

Notebook #2 - Feature Engineering and Model Selection

Notebook #3 - Evaluating on Real World Data

Application - First version video

Application - Final version video

Application Workflow

Part 7: References

1. MotionSense Dataset by Mohammad Malekzadeh, Richard G. Clegg, Andrea Cavallaro, Hamed Haddadi:
   GitHub - https://github.com/mmalekzadeh/motion-sense
   Original Paper - https://arxiv.org/abs/1802.07802
   Kaggle - https://www.kaggle.com/malekzadeh/motionsense-dataset/home
2. Smartphone Sensors information:
   https://fieldguide.gizmodo.com/all-the-sensors-in-your-smartphone-and-how-they-work-1797121002
   https://www.novatel.com/solutions/attitude/
   http://www.rotoview.com/attitude_sensor.html
   https://developer.android.com/guide/topics/sensors/sensors_motion.html
3. Predicting Human Activity Using LSTM:
   https://blog.goodaudience.com/predicting-physical-activity-based-on-smartphone-sensor-data-using-cnn-lstm-9182dd13b6bc