Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет «Высшая школа экономики»

Факультет компьютерных наук

Магистерская программа *Науки о данных*

Департамент *Анализа данных и искусственного интеллекта*

КУРСОВАЯ РАБОТА

На тему *Анализ данных о поведении человека, полученных с сенсоров мобильного телефона \ Human Motion Analysis from Mobile Sensors Data*

Студент группы № м15НоД_ИССА

Киреев Руслан Юрьевич

Руководитель КР

Attila Kertesz-Farkas, PhD, Associate Professor

Москва, 2016

Table of contents

Introduction

Sensors in mobile phones may be considered as yet another input source, which provides information about the environment and the user's motions. There are a wide variety of them: magnetic field sensor, temperature sensor, acceleration sensor, light sensor etc.

The motion sensors have a vast amount of applications in motion games, virtual reality, controlling devices, navigation and gesture recognition.

The gesture recognition provides an interface, that might be convenient in some cases, for example, a gesture can be stand for some command, that opens an application, or, a unique gesture can be used as a password, instead of using the touchscreen. So, the problem is, how to predict a gesture using the signal from a device. There are many obstacles for real-life usage. First of all, different users have different manner of gestures performing, so, the accuracy of prediction really depends on it. Second, we need to develop the recognizer to be fast enough and efficient to deal with a continuous stream of signals. Finally, there is a problem of isolation a gesture from the stream.

There are a plenty of researches, that solves these problems with varying degree of success, using different approaches. For example, in [7] studied the case of the high variability of the input data, that comes from different users, different devices, different orientation. The authors investigate different Machine Learning approaches (SVM, RVM, kNN) and different feature extraction techniques, such as PCA. Some systems [5, 15] successfully use special distance metrics, like Dynamic Time Wrapping, and clustering method of affinity propagation. On the other hand, due to the similarities with the speech recognition problem, the method of hidden Markov models is also popular. The papers describe different ways to implement HMMs to the problem: discrete HMMs with cepstral analysis [11], spectral analysis [16], continuous HMMs [10]. The great advantage of this method is that it can easily handle stream of signals [18]

Hence, there are two main approaches to handle the task: based on Machine Learning, using various distance metrics, and based on the generative model of HMMs. Of course, some of them could be combined to produce better results.

The articles [7, 10, 11, 15, 16, 18] consider only a small amount of gestures, so all of the methods are used focus on the problems of variability of data, but when the number of gestures increases, all of them show a fall in the prediction accuracy [5]. On the contrast, in this paper, we try to tackle only the case of a quite bigger gesture dictionary: the letters of the English alphabet. Particularly, the case of user-dependent and isolated gesture is considered.

The popular techniques were tested and implemented are (i) Support Vector Machine with different distances and (ii) hidden Markov models.

The goal of this project is an Android application, which solves the problem of the English letters recognition via gestures.

Data acquisition

The last decade with the intense growth in Microelectromechanical systems (MEMS) development, it has become possible to fit portable devices up with cheap, compact and energy-saving sensors.

The phone that was used in experiments is Motorola XT910. It has 3-axis accelerometer mounted with resolution of $0.00479\frac{m}{s^2}$.

The Android platform has several sensors that let one track the motion of a device. Two of these sensors are hardware-based (the accelerometer and gyroscope). The sensory data would include only accelerometer measurements, since the phone doesn't have the gyroscope. However, when the values from several sensors are complemented algorithmically somehow, which known as sensor fusion, better performance (time of response, accuracy, noise reduction) can be achieved in task such as dead reckoning, motion games, air mouse, etc.

The accelerometer has 3 axes. The x-axis is parallel to the bottom of the screen, the y-axis vertically goes from the bottom side to the top side of the screen, the z-axis points perpendicularly from the screen. So, when device moves from left to right it shows positive values of x-axis, when it moves away from the user with screen faced, the negative values of z-axis appear, etc.

The accelerometer assesses the acceleration given to the phone with the gravitational force included.

Hence, if the phone is laying still on the table, the force of gravitation influences on the accelerometer values: z-axis would show about $9.8\frac{m}{s^2}$ (gravitational acceleration). The same way, if the device is falling and nothing applies any force, all axis values are equal to 0. Consequently, to get the real magnitudes, we need to subtract the force of gravity from raw data.

This can be achieved by applying a high-pass filter. On the contrast, a low-pass filter can isolate the gravitation. With a digital low-pass filters, the signals, caused by the gravity last all the time (because the force applied permanently), so our goal is to make them pass through and exclude signals that exist over short periods of time. To do so, the weighted moving average can be used to allow fluctuations in the signal build up slowly:

$$y_t = y_{t-1} + \alpha(x_t - y_{t-1})$$

where α is a weight, x_t is a signal value at time t.

In order to collect data an android application has been implemented which acquires signals from the mobile device while the user draws letter in the air holding the phone in the hand.

The main screen can be seen on Figure 1. Due to the task is just acquisition the data, the functionality is straightforward. In the middle one can see the text field, where one can write a character you are going to draw, also it largely duplicated underneath, to see clearly what you are doing. Besides, it shows current linear acceleration, and it has 3 buttons: start record, stop record, and clear file with all previous records. The application's source code is available on the author's github: https://github.com/rkireev/human-motion-analysis

Figure 1. *The main screen of data acquisition application*

The procedure of collecting the data is following:

First, the letter must be chosen via entering it in the text field, one hits the record button and draw it. Every 1/20 of a second the application reads values from the accelerometer. Afterwards, one must stop recording by hitting the rec off button. Then all collected data is being written in the file.

Figure 2. *Examples of the curves were drawn with the phone*

The letters were drawn as shown on Figure 2, basically, the following analogy in acceptable: the phone is a pen, and the table is a sheet of paper, where a user writes huge letters in his habitual manner. The curve lies principally in two dimensions: x-axis and y-axis, in order to get more clear picture and eliminate the factor of the axis swapping.

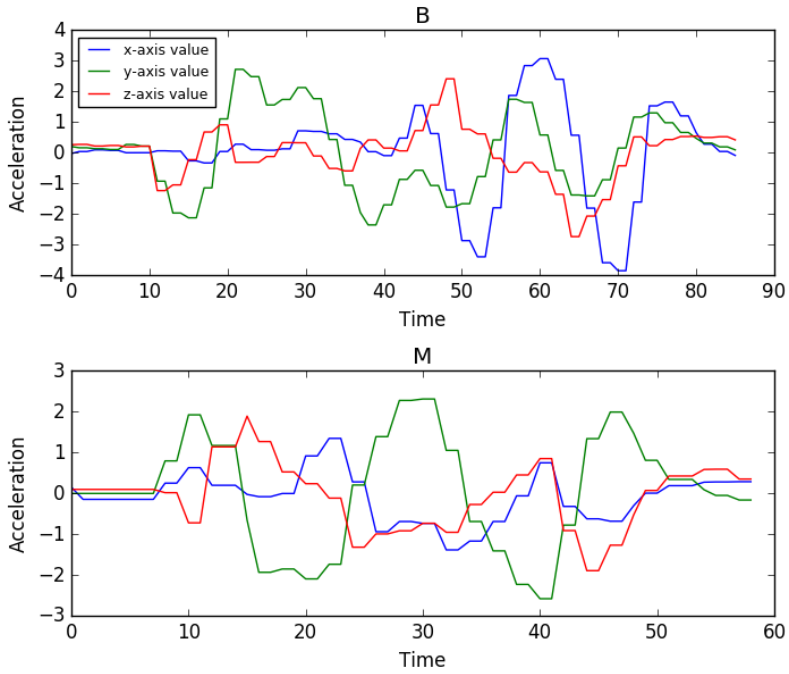

Figure 3. *Acceleration values corresponding to the curves on Fig.2. The top plot corresponds to the letter "B", the bottom plot corresponds to "M"*

On Figure 3 one can see results – values acquired from the accelerometer. First thing to note is that different character takes different time to write. While for "M" it takes about 2.5 seconds, for "B" it's up to 4 seconds. Obvious, but interesting thing is that one can track, how one's movements correspond to the plot. Here, for example, if we consider the "M" character, when the phone moves in the upper-right direction, the x-axis and y-axis acceleration are growing, when it stops, they fall down in time of 10, y-axis keeps decreasing as the device goes down, when you turn in the middle of M and move upwards, correspondently y-axis starts growing again and so on.

Consequently, we have the intuition of every plot characterizes each letter enough well to recognize characters from the values of acceleration.

Data analysis

As was mentioned before, the data consists of observations corresponded to a letter, each observation is a set of three time series – sequences of the acceleration values came from the accelerometer along x, y and z axes. The dataset contains about 30 observations for each letter, so these classes are balanced and on the whole it has exactly 800 instances.

Figure 4 shows plots of the x-axis accelerometer data observations categorized by labels. It can be seen, that the waves corresponding to a letter are commonly overlap each other constructing a general pattern. Some of them have a very clear and precise form, for example, “Z”, “J”, “C”, etc., while some other have more variance, mostly, where movements along x-axis are minor.

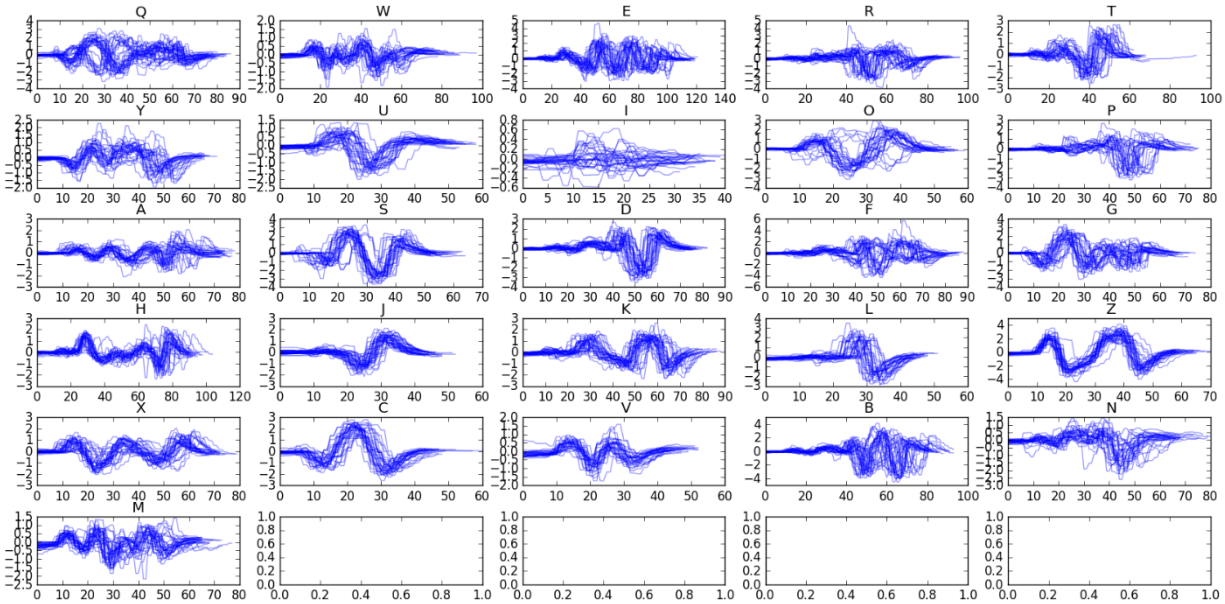

Figure 4. *Plots of each x-axis sequence for each letter in the dataset*

Figure 5 shows the data with respect to y-axis acceleration. The picture gives the motivation to use all points along all axes together. The reason is simple, for example, the x-axis plot of “I” looks cluttered, but the y-axis plot demonstrates the distinct pattern.

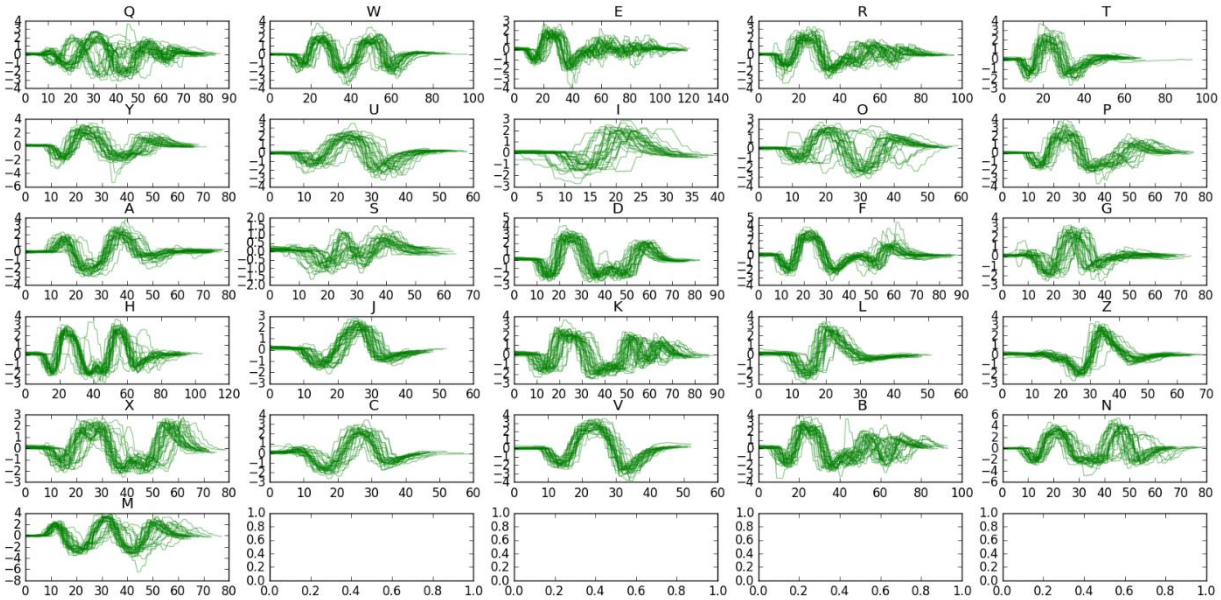

Figure 5. *Plots of each z-axis sequence for each letter in the dataset*

Plots of z-axis are illustrated on Figure 6. Since the dataset is collected in X-Y plane, there is mostly nothing, but noise. In the methods used in this project, experimentally stated, that it's better to exclude this part from the dataset.

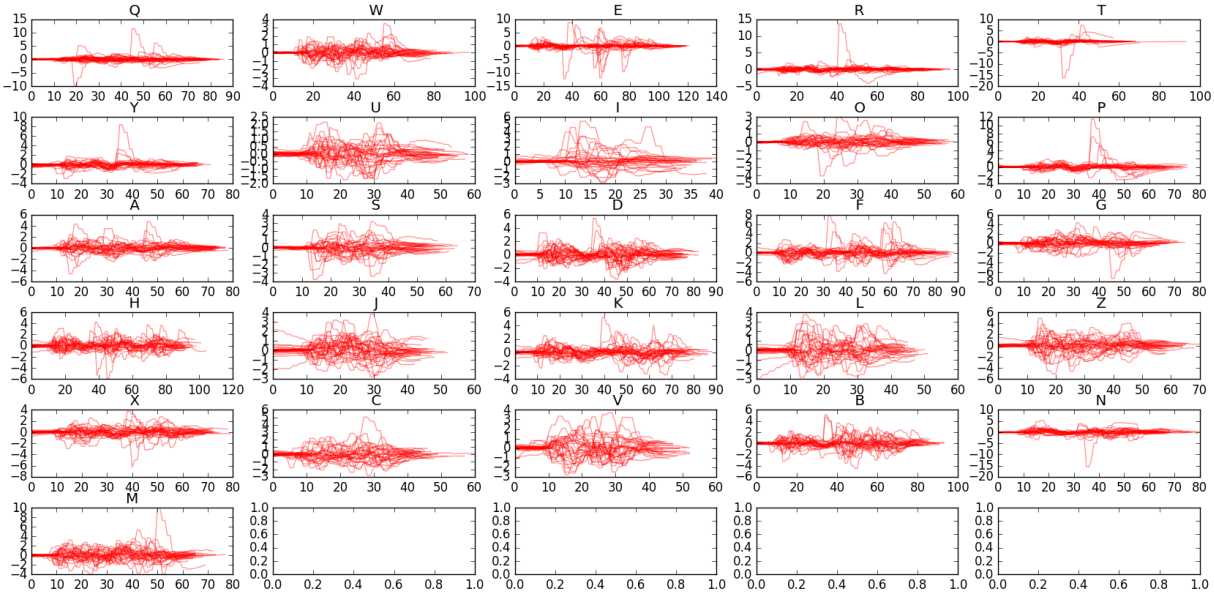

Figure 6. *Plots of each x-axis sequence for each letter in the dataset*

As it has been noticed previously, these sequences are not equal in length. Many standard classification methods require a fixed number of features. Besides, the high length value may imply low speed. Moreover, a different number of points may effect on classification efficiency. To overcome these drawbacks every sequence was linearly re-interpolated into the sequence of fixed length of l. This parameter could be explicitly varied to find the optimal value, when there is no significant lack of the original observation's pattern. Starting from $l = 1$ and steadily increasing the

value, we discover, that there is a point, after which information loss is almost zero. This step was proceeding with every model.

Packages used

The experiments require different toolboxes to handle the data and successfully apply in practice the methods described further. As the main program language Python 2.7 was used, by reason of many packages written for a wide variety of data analysis problems. For testing popular classification tools the scikit-learn package [3] seems suitable. The majority of algorithms are implemented with compiled Cython extensions; therefore, they show the computational performance of C.

There is an implementation of the type of DTW method, called fastDTW [1], that uses the algorithm presented in [14], works in linear time and space.

Regarding HMM, the Python package hmmlearn [2] implements EM algorithm and Viterbi algorithm. It has a scikit-learn interface and support continuous multivariate data, which is very important in our task.

Dynamic Time Wrapping approach

Dynamic time wrapping is a well-known method used to find the most appropriate alignment between two time series when one of them is non-linearly warped along the time axis. It found its major use in the field of speech recognition, since a word, for example, spoken by different speaker can vary in speed and manner, but the pattern should stay similar. So, this method allows to explore, if time series have common region or to find the similarity measure between them. Also it does not require two time series to be of the same length.

The DTW becomes a good substitute for previously widely used Euclidean metric [4, 8, 17], which has obviously a serious disadvantage – sensitivity to non-linear warps in time axis. If we consider two identical waves, but one of them is shifted in time, then Euclidean distance between them will be much larger than it should be. The Figure 7 demonstrates, how poorly Euclidean distance could perform relatively the DTW method.

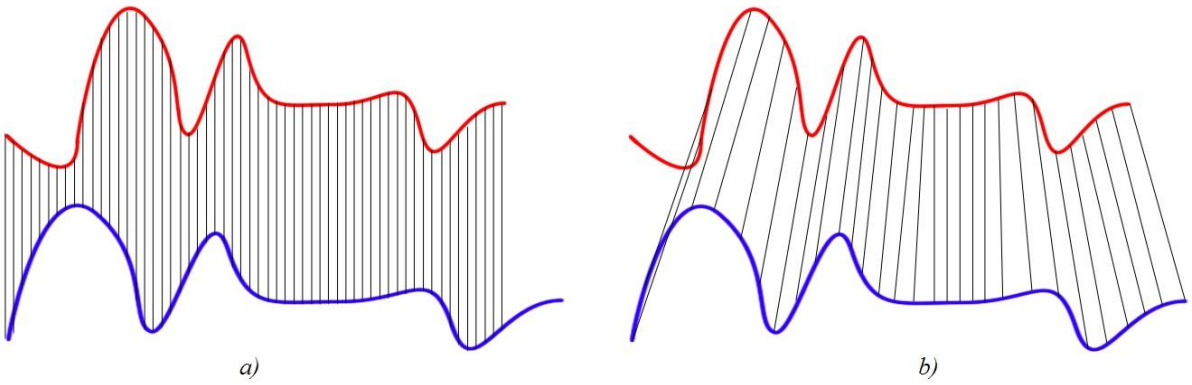

Figure 7. *Comparison between mapping in Euclidean case (a) and DTW (b)*

In practice it is shown that this method is successfully used in many other domains besides speech recognition, such as bioinformatics [4], robotics and more importantly gesture recognition.

Formally let S^1, S^2 be two time series of lengths n, m respectively.

$$S^1 = s_1^1 s_2^1 \dots s_n^1$$

$$S^2 = s_1^2 s_2^2 \dots s_m^2$$

The problem is to find a path $P = p_1 p_2 \dots p_K$, where K is the length of the path and $p_k(i,j)$ is a pair of indexes of the given sequences. It starts in (1,1) and ends in (n,m) to make sure there are all points of both series are involved. Besides, the path should be constructed the way, when indexes are monotonically increasing to ensure the path is not going backward.

The cost of an element of the path $p_k(i,j)$ is denoted as $c_k\left(s_i^1, s_j^2\right)$, commonly it takes as the Euclidean distance. Consequently, the cost of the whole path

$$C(P_{S^1 S^2}) = \sum_{k=1}^{K} c_k\left(s_i^1, s_j^2\right)$$

The optimal path is the path that minimizes the cost function over all possible paths.

$$DTW(S^1, S^2) = \min_{\mathrm{P}} C(P_{S^1 S^2})$$

Dynamic programming is successfully used to find this warping path. Solving this recurrent equation:

$$D(i,j) = c\left(s_i^1, s_j^2\right) + \min\{D(i-1,j), D(i,j-1), D(i-1,j-1)\}$$

where D is the dynamic programming table, $c\left(s_i^1, s_j^2\right)$ is the cost found in the current cell, and $D(i,j)$ is the cumulative cost of $c\left(s_i^1, s_j^2\right)$ and the minimum cumulative costs from the three adjacent cells.

Since the data has three dimensions we need to incorporate the information of all axis to make a prediction. There are two possible ways to do that. One of them is to calculate the cost function between two data points as the sum along three axes: $c(s_i^1, s_j^2) = (s_{xi}^1 - s_{xj}^2)^2 + (s_{yi}^1 - s_{yj}^2)^2 + (s_{zi}^1 - s_{zj}^2)^2$ (squared Euclidean distance seems appropriate here). The other way is to compute separately distances within each dimension, range them as in descending order and compare the best results. But the weakness of this method is when the letter is best characterized by one axis, but poorly by another. We tested both. The second method comes with some adjustments, which work well in practice.

In [8] it is shown that nearest neighbor classifiers with DTW distance instead of Euclidean show good performance. Therefore, the whole classification procedure is following:

We have a training set of labeled observations: each observation is three time series along x, y and z axes. When a new data is observed, we find a list of distances with every sequence in training set in all dimensions separately, as a result it gives three lists of distances. To incorporate this to one ranged list we sum the lists up: first items with first items, second with second, etc. The motivation for this is quite simple: if the letters are very different, then the data would have bigger distance along every dimension; if the letters are different along one axis, but pretty similar along the other (for example, “L” and “J” look alike on y axis, but different on x axis), then the sum would help as to catch out such cases and avoid the mistake; if the letters are same or similar, then the sum would be small.

The next step is where kNN becomes helpful. From the sorted list we got previously, we take top k candidates with smallest distances, and by simple voting choose the most frequent.

The great and distinct property of this distance – it doesn’t require the sequences to have same length. Although, in practices, application to raw time series costs computational time. Moreover, it has been shown that there are no significant benefits from using raw data [13]. The experiments with the acquired data shown, that re-interpolating sequences has positive impact on time and accuracy, as well. So the sequences were re-interpolated into sequences with the optimal length of 20 points.

To test this method and evaluate the accuracy of this classifier the 10-fold cross validation was used. The best results were achieved with k=3 and excluding z axis from the process, since the data was acquired in 2d dimension (on the table). It showed the accuracy at about 96.6%.

The method where Euclidean distance were used, was tested similar way, except this time the process was more straightforward. With the same settings the method showed 97.1% of accuracy.

Hidden Markov Models approach

As mentioned before, just looking at the plot of some letter we can correspond the slopes to the actual movements, for example, if the phone is laying on the table, when we move it upwards, then x and z accelerations stay still, on the contrast y acceleration starts to increase. When the phone goes from bottom-left corner to top-right, x and y axis increase and so on. Every letter can be described by a set of such movements and their order. For example, "T" is can be characterized by sequence of two movements: "top to bottom", "bottom to top-left", "left to right". We don't know what the user did, we only see acceleration data. Hence this observation gives us a motivation to use Hidden Markov Models.

A hidden Markov model (HMM) is a statistical generative model first described in 1960s by Baum and his colleagues [6]. Since then the model has successfully been applied to a wide variety of problems including speech recognition, DNA sequences analysis, text mining, gesture recognition.

Taking into account the great efficiency of the model in gesture recognition [16] it seems reasonable to apply this method to our problem, since the letters might be considered as gestures.

We now try to define a HMM (Gaussian emission case). It can be characterized by the following [12]:

1. n, the number of hidden states. The states describe what we cannot see, but we guess they exist, and they construct the sequence $\{S_1, S_2, \dots, S_n\}$, and state in time t we denote as q_t.
2. A, the transition probability matrix, $A = \{a_{ij}\}$, where a_{ij} is a probability to go from state S_i to state S_j.
$$a_{ij} = P\left(q_{t+1} = S_j \middle| q_t = S_i\right), \;\; 1 \le i, j \le n$$
3. The emission probabilities to characterize each state $b_i(O) = P(O|q_t = S_i)$: $B = \{b_i(O)\}$, in case of Gaussian emission: $b_i(O) = N(O, \mu_i, U_i)$, where O is an observation, μ_i, U_i are mean and covariance matrix, i.e. parameters of the distribution.
4. The distribution of the initial state $\pi = \{\pi_i\}$, where $\pi_i = P(q_1 = S_i)$

In our case the states $S_1, S_2, \dots, S_n$ are might be "top to bottom", "right to left", etc. It's worth noting, that it's not necessary to attach each movement to a state, that is

we don't have to know, how the letters were drawn, but important, they were drawn differently.

Denote a model by θ.

There are three main problems associated with HMMs, solutions for two of them are very helpful for our task. These problems are:

1. Given the model θ and the observation sequence $\hat{O}$, compute the probability of the sequence is being generated by the model, i.e. $P(\hat{O}|\theta)$
2. Given $\hat{O}$ and n, find parameters of the model, that maximizes the probability of $\hat{O}$, i.e. $\max_{\theta} P(\hat{O}|\theta)$

The second problem can be considered as a training step to fit a model with data. Then the first is where we test, if a new sequence can be generated by the model.

The whole algorithm is straightforward. Let $L = \{'A', \dots 'Z'\}$ be a set of letters. For each letter $l \in L$ adjust the model θ_l, when a new observation arrives to make a prediction we find a model most likely generates the observation: $\max_{\theta_l} P(O_{new}|\theta_l)$.

There are several algorithms to solve these problems effectively, avoiding the full search, such as Baum-Welch and Forward-Backward algorithms. That makes it possible to apply this approach in practice.

To test this method and evaluate the accuracy of such classifier the 10-fold cross validation was used. The grid search was used for evaluating the number of hidden states and the number of points in each observation. The experiment with the collected data showed the best parameters to be 12 hidden states and 20 data points. The achieved accuracy is 97.1%.

Support Vector Machine approach

Another popular technique for sequence classification problem is Support Vector Machine (SVM) based on the idea of finding a hyperplane (decision surface) between two classes, that maximizes the margin between them. Formally, it's a minimization problem of empirical regularized risk, we have to find vectors w, w_0 that satisfy:

$$\min_{w,w_0} \sum_{i=1}^{l} (0{,}1 - y_i(\langle w, x_i \rangle - w_0)) + \lambda \left|\left|w\right|\right|^2$$

where l is the number of elements in training set, x_i are objects in training set, y_i labels of corresponding x_i. A binary classification considered above. The solution of the problem can be expressed in terms of dot product between objects of the training set with some coefficients.

Since in our task calculating features of the sequences is a quite computationally expensive problem, which can result in loss of information, the dot product of two sequences can be found by measuring the similarity between them. We could replace the dot product by some function, that is still a dot product, but not necessarily in the space, where x_i lie. This function is called kernel $K(x_i, x_j) = \langle \varphi(x_i), \varphi(x_j) \rangle$, where $\varphi: X \to H$, H is a new space where the dot product defined. It has been proved [19] that the function $K(x_i, x_j)$ could replace the dot product, if $K(x_i, x_j) = K(x_j, x_i)$ and its positive semi-definite. There are several kernels widely used in practice: linear, polynomial, sigmoid, etc. The popular choice is the Radial Basis functions (RBF) kernel, that we going to use in the experiments, since it shows steadily better results with the collected data. However, as the kernel function the similarity measure DTW could be used [7], this case would be considered further.

Radial basis function is defined as $K(x_i, x_j) = \exp(-\gamma \left|\left|x_i - x_j\right|\right|^2)$, where $\left|\left|x_i - x_j\right|\right|^2$ is the squared Euclidean distance, and γ is a hyper parameter.

The technique of Support Vector Machine has several drawbacks, counting the high sensitivity to noise and the absence of built-in feature selection methods. These problems are usually solved separately, before applying SVM. Although, looking back at our data, we conclude, that these steps are not necessary in this case. Outliers and incorrect noisy observation were not included in the dataset, while the acquiring procedure, and we cannot proceed any feature selection, since we don't have features, only time series. Note, that in real-life application this approach should be reconsidered with the robust solution of these drawbacks [7].

To test this method and evaluate the accuracy of the model the 10-fold cross validation was used. The grid search was used for evaluating the parameters C and γ. The first parameter is a tradeoff between misclassification of training examples and simplicity of the decision surface. As lower the parameter C smoother the decision surface is, making the model more general. On the contrast the more vectors are becomes support as C grows, making the surface more complex, which leads to memorizing the training data. The classifier reacts very responsive to γ as well. If it has a small value, the model can't see properly the shape of the data. When γ is high, the more support vectors impact on classification, and it causes overfitting. This testing setting provide us an opportunity to say the optimal values of the parameters, that is, $C = 5$ and $\gamma = 0.05$ with corresponding accuracy score of 97%. Z-axis data was excluded. The data was re-interpolated to 20 points.

The result shown are similar to the results obtained with DTW and HMM.

Recall, in RBF kernel there is used the Euclidian distance, which is as discussed above has a serious drawback compering to DTW: a very high sensitivity to

distortions along the time axis. Hence, it gives us the motivation to replace Euclidian distance with DTW distance. Formally, the kernel function becomes [7]:

$$K(x_i, x_j) = \exp(-\gamma DTW(x_i, x_j))$$

Although, the obstacle here is the computational time. fastDTW works in linear time of the input, but still slow to perform quickly enough in the real-time letter recognition task with a huge trainset. Since a new sequence arrived we have to find distances with every item in the training set.

In the sake of time, the kernel was precomputed on the full set of data, that is, for every pair x_i and x_j $K(x_i, x_j)$ was found before the cross validation.

C and γ was searched as before to get maximal prediction accuracy, so the optimal values are $C = 5$ and $\gamma = 0.067$. Accuracy is 99.3%. Z-axis data was excluded. The data was re-interpolated to 20 points.

Android application and testing

To implement these methods and test, how well they work in real life, an Android application was developed. Since we consider only separate gesture recognition problem, the user himself marks, what part of continuous stream acceleration signals should be used for prediction. To do that, the application uses two buttons, one is to start the sequence recording, the other is to stop.

The training step is excluded, the application uses pre-trained models, the reason for this is trivial: to train the model the user should provide many learning examples, it could take hours to have enough sample to make the models work properly.

There are two methods implemented: 1-Nearest-Neighbor (1NN) and Support Vector Machine (SVM). Also two metrics were used: Euclidian and Dynamic Time Wrapping. Therefore, there are 4 possible options: 1NN with Euclidean distance, 1NN with DTW distance, SVM with Euclidean distance, SVM with DTW distance. The SVM model uses RBF kernel, since it showed best results in the experiments.

Pre-trained multiclass SVM classification model consists of the dual coefficients, support vectors and intercepts. It implements the approach “one versus one” there are $\frac{(n-1)n}{2}$ classifiers (n is a number of classes) trained from data of a pair of classes. The decision function calculates for each classifier, then by voting the most frequent class is chosen. The decision function is:

$$\text{sign}\left(\sum_{i=0}^{m} y_i \alpha_i K(x_i, x_{new}) + b\right)$$

where the product $y_i\alpha_i$ are coefficients corresponding to support vectors, $K(x_i, x_{new})$ the kernel value between support vectors and the new observation, b – intercept.

The kernel function is being calculated on the phone, so the computational time depends on the chosen distance.

1NN is being implemented in the following sense. The distances with every item in the training set are being computed (using Euclidean or DTW). Then it founds the nearest train sample and returns its label.

As the preprocessing we re-interpolate the input sequence to have the length of 20 points according to the length of samples in the training set.

Every implemented method predicts labels in decent time. Interpolation works in linear time. This is the only thing that depends on input, after the signal been re-interpolated, the rest works in fixed time.

The main screen of the application is shows on Figure 8. There are buttons for stop/start recording the signal, also buttons for selection the distance and model. In the center, there is the letter predicted the last time.

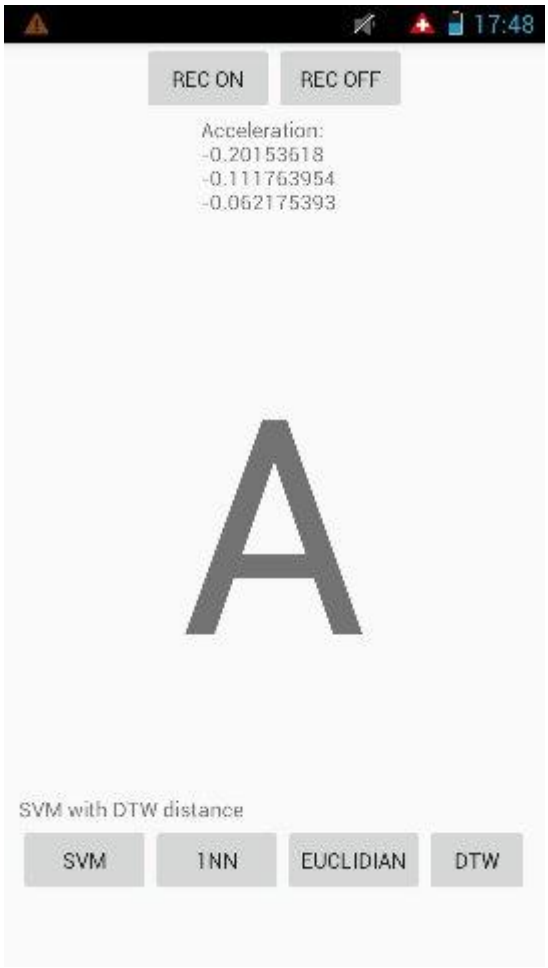

Figure 8 *The main screen of data prediction application*

Results

There were many limitations on the acquisition method: (i) it was collected in two dimensions, (ii) the same letters have the same sequence of movements with similar speed, shape and size. This has led to very small variance of signals of similar letters, as it can be seen on Figures 4 and 5. Consequently, in our experiments it showed very good accuracy, even without sophisticated tuning of the models.

Several different widely used approaches presented in [4, 5, 7, 8, 15] towards the gesture recognition problem were investigated in the essay. The application of HMMs to the problem showed about 97% of accuracy. The kNN methods with the DTW distance achieved the same results as well. Support vector machine with the RBF kernel with Euclidean distance demonstrated the accuracy slightly above 97.1%. Finally, the highest accuracy score of 99.3% has been achieved using SVM with RBF kernel and DTW distance.

Comparing two types of distances (Euclidean and DTW), one may conclude, that DTW outperforms the Euclidean distance. Indeed, the DTW method deals with time distortions, therefore the speed of gestures should not affect the recognition ability. However, in practice there is an obstacle, caused by the nature of acceleration. Consider, for example, two cases. The first case is when one writes the letter “M” very fast. The letter consists of four movements, every movement starts with the impact, that gives high acceleration values. The second case is when one writes the letter slowly. Each movement starts with less force, therefore less acceleration, moreover, if the speed is constant, then the values of acceleration tend to zeroes. This is when one of the drawbacks of DTW reveals [9]. The algorithm faults sometimes to find obvious alignments in two sequences because a point in one sequence is slightly higher or lower than its corresponding in the other sequence. In practice, this causes very high variability of predictions.

It has become obvious, that a correct prediction is very sensitive to the user, speed and shape not even with Euclidean distance, but also with DTW.

On the other hand, if one writes the letters the same way as in the training dataset, the accuracy remains high, comparing to the experiments. However, there are still frequent mistakes between similar letters, like “P” and “D”, “A” and “H”.

Conclusion

In this paper we have considered very restricted case of input data. It was user-dependent, there is only one user took part in the acquisition process, but there are many other ways to write the same letter, for example, some people write “Z”, while the other write with a bar in the middle. Another limitation is that the data were acquired in two dimensional space. And finally, the gestures were separated, therefore, solved only isolated gestures case.

Following these settings there is no wonder, that in the experiments all methods performed very well. Every method showed the accuracy score more than 96%. In particular, the DTW method itself showed already good results, but in combination with Support Vector Machine it achieved the score above 99%, so the average mistaken letter is less than one in the testing set out of 80 samples. The different, generative, approach of the hidden Markov models demonstrated good numbers as well, more than 97% correct predictions.

However, the experiments differ from practice. Testing the Android application revealed all the drawbacks of the constraints. In the test phase it predicts very well for the person who created the training dataset. Besides, the large amount of mistaken predictions with the similar letters is another issue, that occurred in the application testing, while in the experiments, the models learnt to deal with even such difficulties.

Nevertheless, these issues are unavoidable, since we have a large gestures dictionary, with some very similar signal waves, for example, “P” and “D”.

As the further work, there should be studied a lot more variable data input, to cover much more patterns. Increasing variability of data, the models should be adjusted more accurately, using some preprocessing techniques, for example, there is a different orientation problem, that could be solved by PCA [7]. Besides, different behavior of a user should be considered, while he is drawing letters, he could stand still or walk, or drive a car, etc. Finally, we have to automatically isolate the gesture signals out of the acceleration values stream, instead of making a user to do it manually.

Regarding to further Android application development. First, it is necessary to add the hidden Markov model, since a different approach could work better in practice, then other methods. Second, implement gesture isolation out of continuous stream of acceleration data. Besides, provide a convenient interface to make it useful in practice, for example, to add the text area, where a user could enter a text letter by letter via gestures and send it to another application. Finally, the phase of adjusting models’ parameters could be moved from the external device to the phone, in order

to provide autonomy of the application. However, it could increase the time of prediction, due to capabilities of mobile devices.

References

[1] https://pypi.python.org/pypi/fastdtw

[2] https://github.com/hmmlearn/hmmlearn

[3] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011

[4] Aach, J. & Church, G. (2001). Aligning gene expression time series with time warping algorithms. Bioinformatics. Vol. 17, pp. 495-508

[5] A. Akl, C. Feng, and S. Valaee. A novel accelerometer-based gesture recognition system. Signal Processing, IEEE Transactions on, 59(12):6197–6205, 2011.

[6] L.E. Baum, T. Petrie, G. Soules, and N. Weiss, A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chins," Ann. Math. Stat., Vol. 41, No. 1, pp. 164-171, 1970.

[7] G. Belgioioso, A. Cenedese, G. I. Cirillo, F. Fraccaroli and G. A. Susto, "A machine learning based approach for gesture recognition from inertial measurements," 53rd IEEE Conference on Decision and Control, Los Angeles, CA, 2014, pp. 4899-4904.

[8] Hui Ding , Goce Trajcevski , Peter Scheuermann , Xiaoyue Wang , Eamonn Keogh, Querying and mining of time series data: experimental comparison of representations and distance measures, Proceedings of the VLDB Endowment, v.1 n.2, 2008

[9] Keogh, E. & M. Pazzani. Derivative Dynamic Time Warping. In Proc. of the First Intl. SIAM Intl. Conf. on Data Mining, Chicago, Illinois, 2001.

[10] J. Liu, L. Zhong, J. Wickramasuriya, and V. Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. Pervasive and Mobile Computing, 5(6):657–675, 2009.

[11] C. Nickel, H. Brandt, and C. Busch. Classification of Acceleration Data for Biometric Gait Recognition on Mobile Devices. In BIOSIG 2011 - Proceedings of the Special Interest Group on Biometrics and Electronic Signatures, 2011.

[12] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE, 77(2):257–286, 1989.

[13] Ratanamahatana, C. A. and Keogh. E. (2005). Three Myths about Dynamic Time Warping. In proceedings of SIAM International Conference on Data Mining (SDM '05), Newport Beach, CA, April 21-23, pp. 506-510

[14] S. Salvador, P. Chan. "FastDTW: Toward accurate dynamic time warping in linear time and space." Intelligent Data Analysis 11.5 (2007): 561-580

[15] J. Wu, G. Pan, D. Zhang, G. Qi, and S. Li. Gesture recognition with a 3-d accelerometer. In Ubiquitous intelligence and computing, pages 25–38. Springer, 2009.

[16] J. Yang, Y. Xu and C. S. Chen, "Hidden Markov model approach to skill learning and its application in telerobotics," Proceedings of 1993 IEEE Inter. Conf. on Robotics and Automation, Vol. 1, pp.396-402, 1993

[17] Yi, B.K., Jagadish, H. & Faloutsos, C. (1998). Efficient retrieval of similar time sequences under time warping ICDE '98, pp. 23-27

[18] Y. Yin and R. Davis. Gesture spotting and recognition using salience detection and concatenated hidden markov models. In Proc. of ICMI, pages 489–494, 2013

[19] Vapnik V (2000) The nature of statistical learning theory. Springer, Berlin