Open your mind. LUT.
Lappeenranta University of Technology
Fall 2019

Pattern Recognition

BM40A0702

8.12.2019

Pattern Recognition

Practical Assignment

Digits 3D dataset

Note: I bit of misunderstood the purpose of this assignment from the beginning as the goal was to just create a digit_classify.m function which handles the whole process from one single digit. I started the work with MATLAB notebook to design the classifier and use the designed classifier in "production" for single digits only. So the digit_classify.m function does not load the whole dataset in or calculate weights of the neural network, that work is done in the notebook file. (In my opinion it makes no sense to calculate NN weights each time in digit_classify.m function, and I have no time to fix this)

Joni Kettunen XXXXXXX

Contents

Introduction

This paper is a documentation for the practical assignment done for the Pattern recognition course in Lappeenranta university of technology. The first part of the paper is about data pre-processing and feature extraction. Second part is about implemented classification models and the last part is performance analysis of the chosen model.

The workflow for this project was mostly trial and error, as a simple classifier was easy to make, but the accuracy was not adequate. Different data pre-processing methods and steps (PCA, FPCA, different normalizations, time series smoothing etc.) were tried and based on results they were implemented in the "final" model. Total of three different algorithms (FKNN, Similarity classifier, Neural Network) were tested for the dataset. Since code written for previous courses was not allowed in the final model, Neural network was the chosen algorithm for the digit classification function.

Since my background is in Industrial Engineering this assignment proved to be very challenging. However in the end a lot was learnt, and I'm satisfied with the decision to take this course as extra.

Main work on data pre-processing and developing the classifier can be found from the "Practicalassigment.mlx" live script (MATLAB notebook). Only low-level MATLAB functions are used in the solutions. I used templates from Pattern recognition course for the multi-layer NN classifier and classifier templates from Fuzzy data analysis course for similarity and FKNN classifier. Exponential smoothing function template is from Kamil Wojcicki (2012).

Data pre-processing and feature extraction

The given data is in timeseries form containing positional location of index-finger in x y z axes. Each number has timeseries in different file, so first step is to read the different files to larger structure. After this was done, trial and error with pre-processing could be started. The Z (depth) was dropped based on graphical analysis of the data, using common sense as argument. Depth axis would not hold information value for a human, who would be doing classification for the data manually. Including depth in the wrangled data was tested, and it added only extra noise and therefore decreased the classification accuracy.

Marzinotto (2017) has an excellent GitHub repository that includes documentation and steps made on classification on similar leap motion time series data. Marzinotto suggested seven steps on pre-processing, out of which two were used in this project. PCA for the timeseries and resampling of digits were left out, since PCA for some reason only seemed to worsen the results and resampling of digits would be difficult in this case due to missing information on observation frequency or acceleration (Leapmotion sensors captures also these, but this information was not provided in the assignment files).

Some research was done on why PCA seemed to worsen the results. Plots on example digits 1 and 3 timeseries can be found from figures 1 and 2.

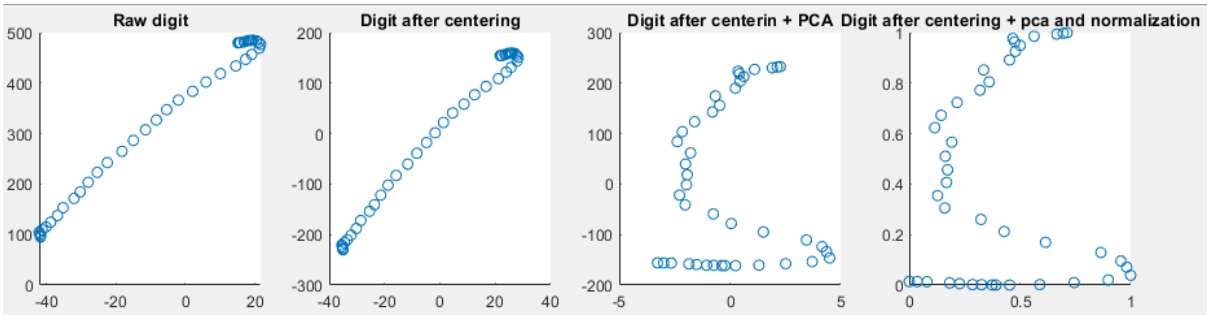

Figure 1 Pre-processing of digit 1 timeseries

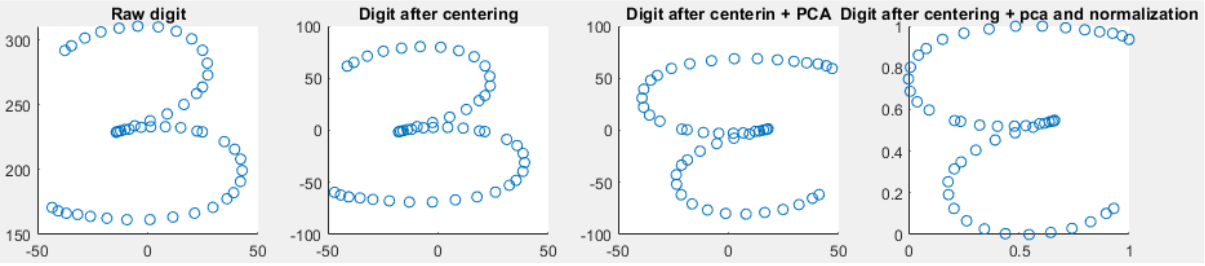

Figure 2 Pre-processing of digit 3 timeseries

PCA on each timeseries digit separately seems to make digit 1 into form that is not recognizable by human eye. This would perhaps not be a problem to classification algorithm, but nevertheless the accuracy was lower on all tested algorithms when PCA for timeseries was done. Since it was expected that y axis has most variation, the order of principal components had to be redefined in the graphical presentation. Also PCA seems to flip some number threes and some not, this becomes a problem since the next step in chosen way of pre-processing was to transform each timeseries into a 2-dimensional pixel matrix. So in the end PCA for timeseries was dumped and the only pre-processing methods for time-series data was centering, smoothing and rescaling each timeseries separately. Dimensionality reduction was done simply by just removing Z axis from the data, therefore projections provided by PCA were not needed.

As an alternative to resampling the digits according to frequency of observations, exponential smoothing was used. Exponential smoothing is common a common pre-processing technique for timeseries data used in many engineering fields. (Stanley, 2010) Exponential filter function requires smoothing constant, which was decided upon training neural network multiple times and choosing the constant that provided best accuracy in test set. Figure 3 shows the effect of exponential smoothing on digit 7.

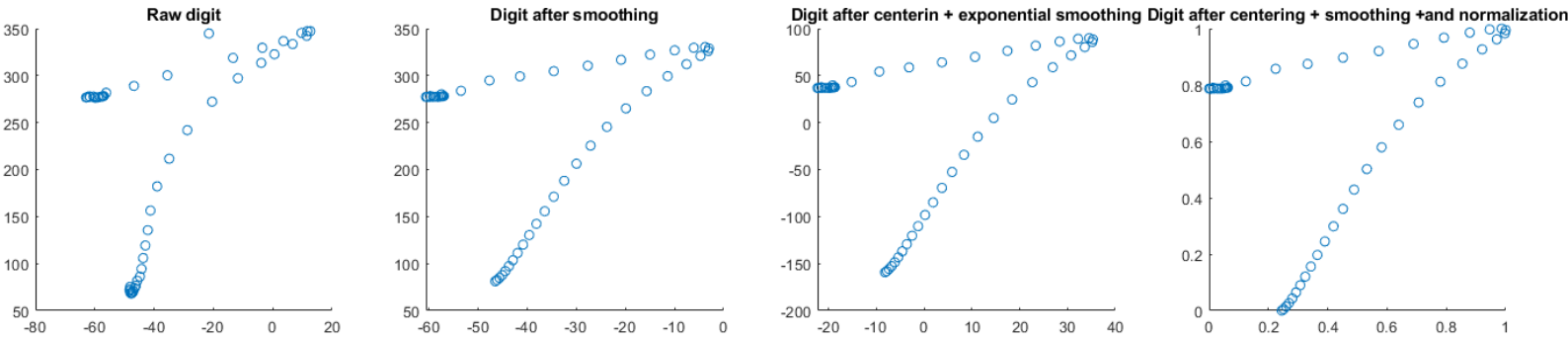

Figure 3 Effect of exponential smoothing.

Feature extraction

After the timeseries data is pre-processed, the next step is to extract features out of the data in order to use "generic form" classification algorithm functions (generic as in columns as features and rows as observations). In order to achieve this, the timeseries data was transformed into pixelmap of 12*12 pixels. The pixel density had a major impact on classification accuracy, and upon several iterations 12*12 pixelmap was chosen as it seemed to provide constantly highest accuracy on multilayer perceptron algorithm. In the pixelmap the pixel is either activated or disabled, depending on whether the wrangled timeseries data goes through the pixel.

On top of the pixelmap, several other features were calculated from the timeseries. Table 1 consists a list with explanations on the captured features. Each timeseries has minmax normalized data, so the positions of the timeseries locations are comparable.

Table 1. Extracted features

Feature	**Explanation**
startX	Start position X on the time series data
startY	Start Y position on the time series data
afterstartX	Position X of the timeseries data, when 33% of the datapoints are gone through
afterstartY	Position Y of the timeseries data, when 33% of the datapoints are gone through
beforeendX	Position X of the timeseries data, when 66% of the datapoints are gone through
beforeendY	Position Y of the timeseries data, when 66% of the datapoints are gone through
endX	End position X on the time series data
endY	End position Y on the time series data
startDirX	Direction of the time series on x axis at starting point
startDirY	Direction of the time series on Y axis at starting point
afterstartDirX	Direction of the time series on X axis, 33% of the timeseries values are gone through
afterstartDirY	Direction of the time series on Y axis, 33% of the timeseries values are gone through
beforeendX	Direction of the time series on X axis, 66% of the timeseries values are gone through
beforeendY	Direction of the time series on Y axis, 66% of the timeseries values are gone through
Eigenvector11	Highest Variability direction of scaled + smoothed time series data (no normalization)
Eigenvertor12	Highest Variability direction of scaled + smoothed time series data (no normalization)
Eigenvector21	Second highest Variability direction of scaled + smoothed time series data (no normalization)
Eigenvector22	Second highest Variability direction of scaled + smoothed time series data (no normalization)
Eigenvalue1	Variability amount on highest variability direction
Eigenvalue2	Variability amount on second highest variability direction

Upon testing the features in the previous table seemed to improve classification accuracy 3-4% depending on the run. Digit 1 classification accuracy improved significantly, probably due to high variability in one eigenvector direction and low variability second eigenvector direction.

The dataset was split into training and testing sets with 80% of the data in training and 20% of the data in testing set.

Classification and performance evaluation

Since the assignment stated it is forbidden to use templates from other courses, those algorithms are not described. However FKNN algorithm provided 91% accuracy in testing set and similarity classifier gave 94% accuracy in testing set.

Multilayer perceptron neural network was chosen as the method for classification. Briefly said, neural network is an algorithm that is fits to any continuous function given sufficient network architecture. Neural network consists of neurons in input, hidden and output layers. Synapses connect the neural network neurons together, normally a neuron in one layer is connected with a synapse to each neuron in previous layer and to each neuron in next layer. At input layer data is inserted, and at each hidden layer neuron the input is multiplied with specific weights that are learned as a part of the neural network training process. The neural network output represents the combined input of all the neurons. (ML-cheatsheet, 2019)

Neural network neuron is the basic object within neural network that gets input from either source features or previous layer and multiplies that input with weights and runs the results through nonlinear activation function. After the neuron has done its job, the output is sent forward to next hidden layer neuron or to the output layer neuron. (ML-cheatsheet, 2019)

The network training progress goes through the following steps on each iteration (ML-cheatsheet, 2019):

1. Forward propagation
    a. Layer by layer multiply inputs with weights and run through activation function
2. Backpropagation
    a. At end layer calculate the prediction error
    b. Layer by layer go back in the network structure, and use derivative of activation function and layer weights to get deltavalues on each layer
3. Update weights on each layer based on the error

Multiplayer perceptron with two layers and ten nodes on each layer was chosen as the algorithm to be used in the final classification implementation. On table 2 results of training NN with different number of nodes in each layer is represented.

Table 2 Neural Network performance with different configurations. Epochs limited on defined error level

Nodes Layer 1	**Nodes Layer 2**	**Nodes Layer 3**	**Test Set Accuracy**
10			93.08%
20			94.70% error seemed to go lower
30			94.04% error seemed to go lower
60			Unable to calculate, error rockets to the sky at some moment. Weird error.
10	10		95.36% error seemed to go lower
20	10		94.04% error seemed to go lower
30	10		94.70%
10	20		95.36%
20	20		94.70%
10	10	10	90.07% Needed significantly more epochs, seems to be overlearning

It seems that increasing the number of nodes in each layer makes little to no difference in the test set accuracy. However two-layer perceptron seemed to provide consistently higher accuracy that one-layer perceptron. After the number of layers in the network was decided, next step was to choose epoch (iteration) cut off level to prevent overlearning. The classification error was used to cut off iteration loop. Several cut-off levels for classification error were tried, and the one was chosen for the chosen method that provided consistently highest accuracy on the testing set. Figure 4 contains information on the error upon iterations.

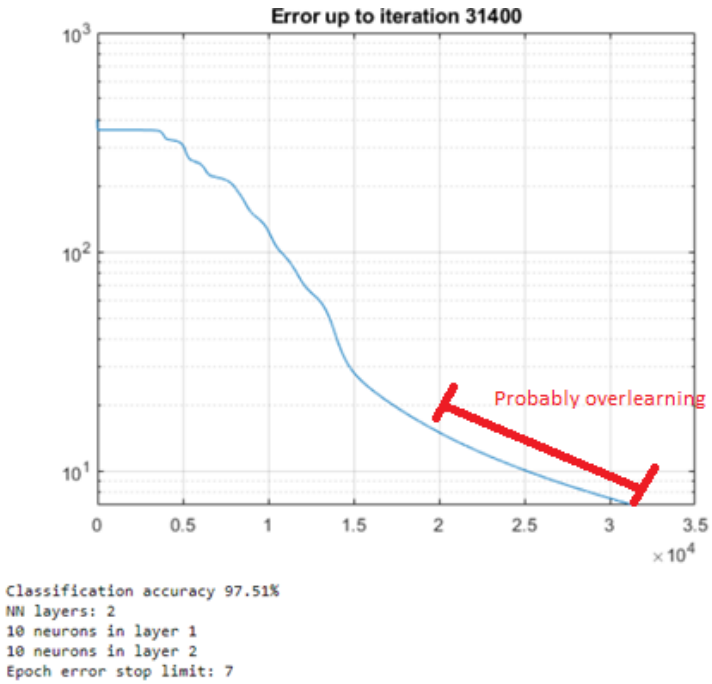

Figure 4 Error development based on iterations

After the final digit_classify.m function was done, upon inspection it had 100% accuracy on the train set. So the cut off level was probably chosen way too low. However due to time limitations and sufficient classification performance on testing set the performance of the classifier was stated as satisfactory. With 200 numbers in the testing set only 5 digits were misclassified this means 97.51% accuracy on test set. The misclassified digits can be found on figures 5 and 6.

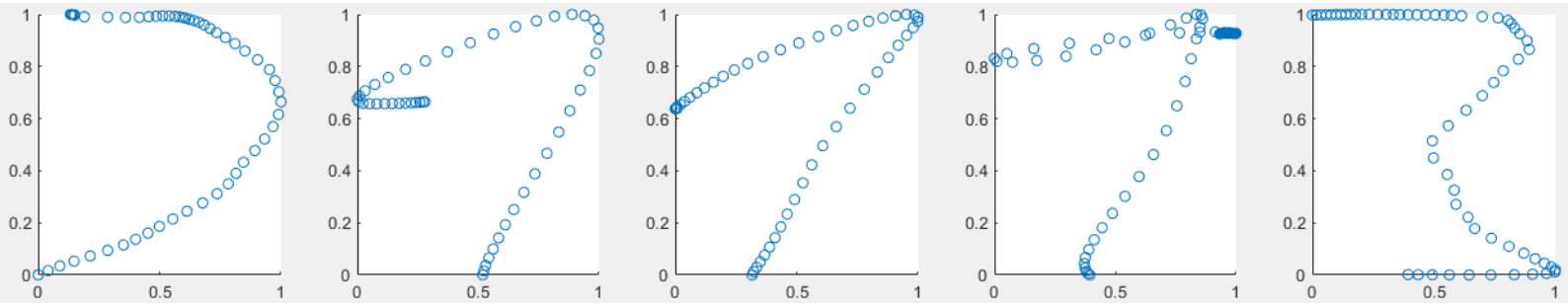

Figure 5 Misclassified digits (pre-processed)

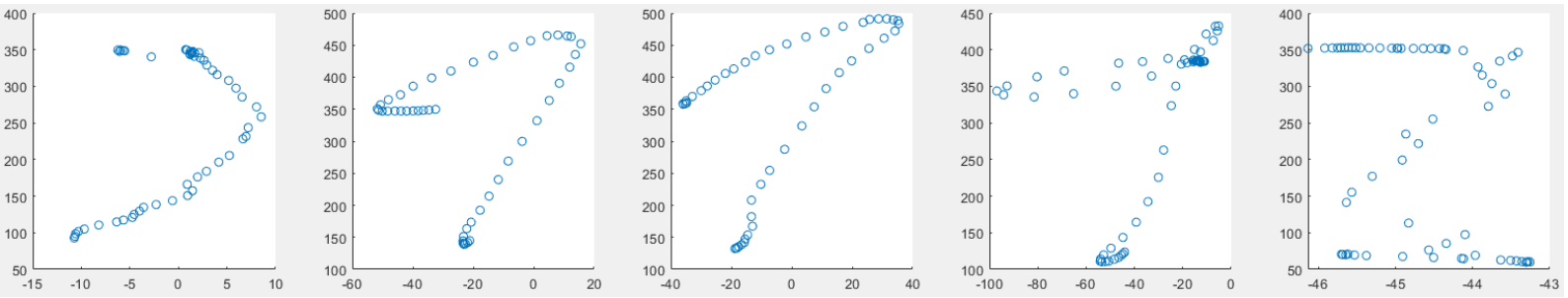

Figure 6 Misclassified digits (raw-digits)

The true classes for the misclassified digits are [1 1 1 9 1]. Even using human eye, I would classify these digits differently.

Confusion matrix

Since there are only 5 misclassified digits in the dataset with a NN trained with 800 digits on train and 200 digits on test, building a confusion matrix with this split is useless. So in order to get insight on the "wellness" of chosen pre-processing methods and neural network architecture the neural network is trained again with 500 digits on training set and 500 digits in testing set in order to get more misclassified digits. Using this split, the accuracy on test set was 95.61%. The confusion matrix is visible on figure 7.

Predicted \ Actual	0	1	2	3	4	5	6	7	8	9
0	50	0	0	0	0	0	1	0	0	0
1	0	39	0	1	0	0	0	0	0	0
2	0	2	47	0	0	0	0	0	0	0
3	0	3	0	49	0	0	0	0	0	1
4	0	0	0	0	47	0	0	0	0	0
5	0	1	0	0	0	50	0	0	0	1
6	0	0	1	0	0	0	49	0	0	0
7	0	3	1	0	0	0	0	50	0	1
8	0	0	1	0	0	0	0	0	50	0
9	0	2	0	0	3	0	0	0	0	48

Figure 7 Confusion matrix done with 50% of the data in testing set

It seems that the algorithm works pretty bad with digit 1 but surprisingly well with digit 7. Worse results for digit 1 may be due to chosen pre-processing or feature extraction method. The classifier performance with other digits is adequate. After graphical analysis it turns out that the exponential smoothing factor may be too high, since the smoothing "straightens the curves" of the numbers, and make them look more like digit 1.

References

Marzinotto (2017) LeapMotionGestures. Github repository
Available: https://github.com/GMarzinotto/Leap-Motion-Gestures

Brownlee (2016) Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras. Web article
Available: https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/

Kail Wojcicki (2012) Exponential smoothing function
Available: https://www.mathworks.com/matlabcentral/fileexchange/34743-exponential-smoother

Greg Stanley (2010) Exponential filter
Available:
https://gregstanleyandassociates.com/whitepapers/FaultDiagnosis/Filtering/Exponential-Filter/exponential-filter.htm

ML-cheatsheet (2019) Webpage with lots of information on machine learning. Cited 6.12.2019.
Available: https://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html