BMI Coursework - A Bayesian Classifier with Linear Regression Decoder

Luis Chaves Rodriguez (*01128684*), Francesco Guagliardo (*00978390*), Daniele Olmeda (*01114530*)
Team name: *monkeysdecodemonkeys*

Abstract

A combined classification and regression model was built to decode multi-channel neural spikes from the motor cortex of a primate into the corresponding primate's upper limb position. The spikes were recorded from the cortex of a monkey while its arm aimed at a target. The classifier chosen is a Bayesian classifier and it is used to discriminate between movement directions. A Principal Component Regression method is used to estimate the hand position. A classification accuracy of 99.2% and a root-mean squared error (RMSE) of 11.64 for the movement trajectory was obtained with a 80:20 split of training to testing data. Other methods, such as neural networks, Error-correcting output code SVM and others were also tested but proved to be less efficient in terms of time and prediction accuracy.

I. Introduction

Traditional upper-limb prosthetic control is slow, unpredictable and it does not match its human equivalent. Despite tremendous efforts in the field, prosthesis user satisfaction needs to be increased and that will be mainly achieved by more intuitive and natural control [2]. Due to the electrical nature of neurons communication, we are able to connect humans and machines. Though the nature of neuron's communications is beneficial to neuroscience, the "neural code" yet needs to be broken. In this report, the data exploited comes from the brain of a monkey who has been trained to perform 8 different movements. Each movement was trialled 100 times whilst neuron spikes (from 98 electrodes) and hand position were recorded at every millisecond. The subject of this report is the decoding of this neuron recordings into movement trajectory (x and y-coordinates).

I.A. Understanding the data

The nature of the problem in question is of continuous estimation which implies regression: learning from a set of data points and predicting a response from new data. Given that the monkey is trained on movements at 8 different angles, this problem can also be treated as a classification one. The processing of the raw data reveals specificity of some electrodes to respective movements as it can be seen in figure 1. In this example, the so-called positively responsive electrode could be measuring an excitatory neuron while the negatively responsive one could be a inhibitory neuron and the non-responsive electrodes do not give us any information. Not such specificity is visible when looking at all the electrodes at the same time as expected in the case of neurons with different properties and functions. (figure A4). When looking at the tuning curves of all 98 electrodes to the

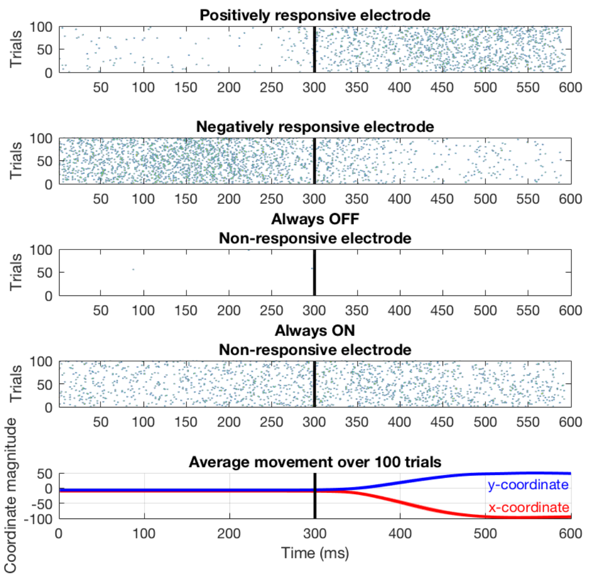


Figure 1: Raster plot for 4 different electrodes response to movement 4 over 100 trials and over time. At bottom average movement in the x and y-coordinates over a hundred trials. From top to bottom, these are electrodes: 27, 90, 38, 9. The black bold bar shows the start of the monkey movement

8 movements (figure A2), it becomes visible that over trials and over every millisecond sensitivity is low for most electrodes as the tuning curves show flat profiles. As shown in figure whilst these electrodes do not elicit strong responses, they still yield specificity to some movements. Whilst the

difference will be smaller for electrodes with less sensitivity, a classifier may still be able to differentiate classes using signals from those electrodes. Though electrodes with flat tuning curves will help fine-tune the classification but changes in these electrodes will be of less importance to those from more active electrodes. This is useful as it could be considered to use less data to improve execution time while maintaining correct model accuracy. Upon analysis of the raw data it was decided that for classification purposes, each electrode spike data would be used as a feature vector for classification. The problem is thus divided into two tasks: a classification one where each set of spikes from all electrodes is classified into one of the eight movements and a regression problem where a model learns to predict one x-coordinate and one y-coordinate from a set of spikes for each movement.

II. Methods

The characteristics and parameters values of the chosen classifier and regressor are described in this section. All the data presented in this report was obtained using a training:testing split of 80:20.

II.A. Classifier

The classifier chosen uses Bayes theory to calculate the probability of a certain data point x belonging to a specific class C_k given the data point $p(C_k|x)$, the class with the highest probability is then chosen. We call this a Bayesian classifier. The training of the classifier involves creating a multi-class model (8 classes in our case), which indicates the probability of a data point belonging to a class, given the class $p(x|C_k)$. The classifier assumes the data to be normally distributed since it uses a multivariate Gaussian function to create the model as shown in equation 1 [3].

$$p(\mathbf{x}|C_k) = \frac{1}{\sqrt{(2\pi)^d|\mathbf{\Sigma}|}} exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right) \tag{1}$$

Where Σ is the covariance matrix of the training data belonging to C_k and μ is the mean. We can then use Bayes theorem to obtain $p(C_k|x)$ as follows:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)} \tag{2}$$

Where $p(x)$ is the probability of a data point happening and can be omitted for classification purposes as it is constant in the data set. $p(C_k)$ is the prior probability of class C_k which depends on how many data points belong to that class over all the set.
We assess the performance of our classifier with a measure of accuracy calculated as the sum of true positive and true negatives divided by the total data points:

$$\text{Accuracy} = \frac{TP+TN}{P+N} \tag{3}$$

Accuracy alone is enough to assess the performance; F1 score was not needed since the data set is evenly distributed between classes [5].

For our classification problem we set $\mathbf{x}$ as the sum of the spikes for each electrode and for each training trial from 1 to 320, 340, 360, 380 and 400 milliseconds, we call this variable *train time*. The more times a neuron fires the more it is active, and this is reflected by the sum. The classifier is allowed to "change its mind" on what class a data point belongs to for the first 4 testing data points (since each testing data point is provided at time intervals of 20ms) and then make a final decision at 400ms. In figure 2 we can see how the variable *train time* does not have a great effect on classification accuracy. The rationale behind this idea is that the classifier will perform poorly on classifying spike data including values up to 560 ms if the classifier has only be trained on data up to 320 ms or some other low value. As shown in figure 2 the Bayesian classifier is robust to *train time* variability, though optimizing the *train time* variable shows significant improvement in other classifier such as SVM(figure). The covariance conditioning (*cov conditioning*) is a determining hyper-parameter for classification accuracy. The covariance matrix Σ of a correlated data set (such as ours as shown in section I.A) can be close to singular. Therefore, when performing its inversion in equation 1 we would get erroneous results. *Cov Conditioning* is a value that is added to the diagonal of Σ to make it non-singular.[1]. Figure 2 shows how accuracy changes as a function of *Cov Conditioning*. We can clearly see how for 0 or small values of *Cov conditioning* the accuracy is very

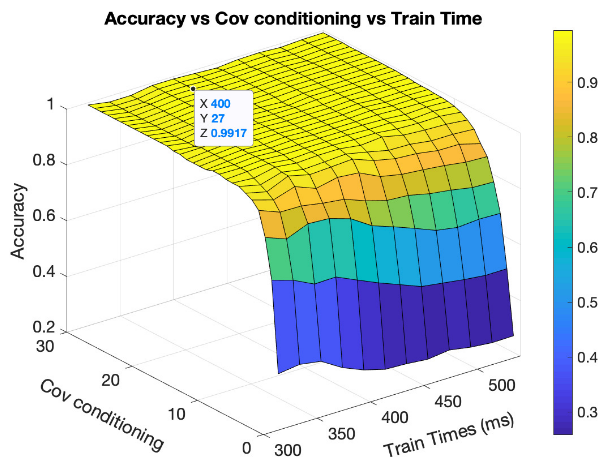


Figure 2: Classification Accuracy as a function of the hyper-parameters *cov. conditioning* and *train times*

low ($< 40\%$) but for values of *Cov Conditioning* > 5, accuracy jumps to 95% and keep on increasing up to 99.2% at a value of 27.

II.B. Regressor

The regressor chosen uses the the concepts of least square regression and principal component analysis (PCA) to provide an accurate prediction of the hand position given the neural spikes. The method is called Principal Component Regression [4]. Linear regression can be summarised as a problem of calculating some regression coefficients $\mathbf{B}$ such that

$$\mathbf{Y} = \mathbf{X}\mathbf{B} + \mathbf{C} \tag{4}$$

Where $\mathbf{Y}$ is the response that we want to estimate, in our case the hand x and y positions. $\mathbf{X}$ is the input or independent variable, in our case all the neural spikes of all test trials and $\mathbf{C}$ is random noise. The bold symbols indicate that the variables are column vectors. Ordinary Least Squares (OLS) (equation 5) method could be used to calculate the regression from our training data. Unfortunately, our set of data is highly dimensional (98 electrodes) and as we have seen in section I.A, some electrodes' response is very low or nonexistent which means that some of the data will be redundant (i.e. correlated). This in turn means that the matrix $\boldsymbol{X}^T\boldsymbol{X}$ would be close to singular, its inverse would be inaccurate, hence equation 5 should not be used.

$$\hat{\boldsymbol{B}}_{OLS} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y} \tag{5}$$

To circumvent this problem we can calculate the principal components of the 98-dimensional data set (using singular value decomposition) and use only $r < 98$ principal components. In this way we would get rid of the correlated and less meaningful dimensions, this is called principal component regression (PCR). Equation 6 shows the formulation of PCR.

$$\hat{\boldsymbol{B}}_{PCR} = \boldsymbol{V}_{1:r}(\boldsymbol{\Sigma}_{1:r})^{-1}\boldsymbol{U}_{1:r}^T\boldsymbol{Y} \tag{6}$$

Where $\boldsymbol{U}_{1:r}\boldsymbol{\Sigma}_{1:r}\boldsymbol{V}_{1:r}^T = \boldsymbol{X}$ is the singular value decomposition of $\boldsymbol{X}$ by retaining the best r principal components. The choice of the value of r is essential for a good regression. Figure 3 shows how changing r and another hyper-parameter of the regressor *fix position* affects the final RMSE of the decoding problem. The parameter *fix position* is a value that is subtracted to the final estimate of the PCR in case this value exceeds the maximum training x or y position, i.e. in case the regressor overshoots. We can see that increasing r the RMSE goes up, since we are introducing more and more correlated dimensions that add noise to the data; we also see that a value of $r = 2$ provides already a good RMSE which confirms the redundancy in the data set, thus confirming our hypothesis on from section I.A. In terms of *fix position* the best values lays at 8 where we obtain a minimum for every value of r, this suggests that

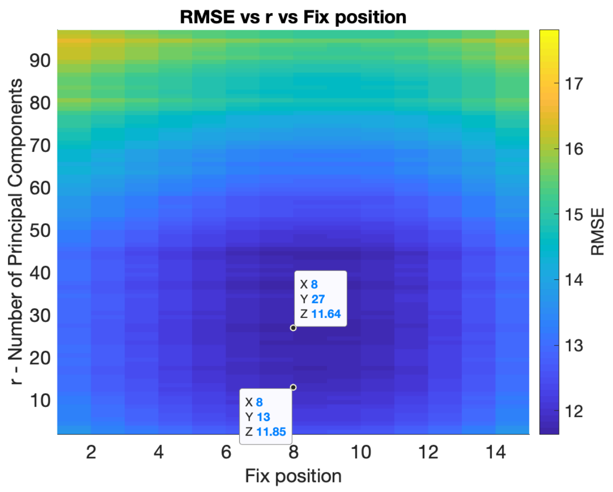


Figure 3: RMSE as a function of r (principal components used in PCR) and the *fix position* hyper-parameter. *The RMSE was calculated using the provided test function, the Bayesian classifier with classification accuracy of 99.17% and the PCR as in equation 6.*

the PCR overshoots in average by 8cm. To choose the best r we also have to look at the training computational time used to perform operations on a highly dimensional data set; figure 4 shows how computational times changes when changing the value of r. This shows that an optimal value of r would be around 3-6 since it has low computational time (around 28 seconds) and still provides good RMSE of 12.3 (at $r = 13$) compared to the absolute minimum achieved of 11.64 (at $r = 27$).

The model uses a *train times* value of 400, a *cov conditioning* value of 27, an r value of 27 and a *fix position* of 8.

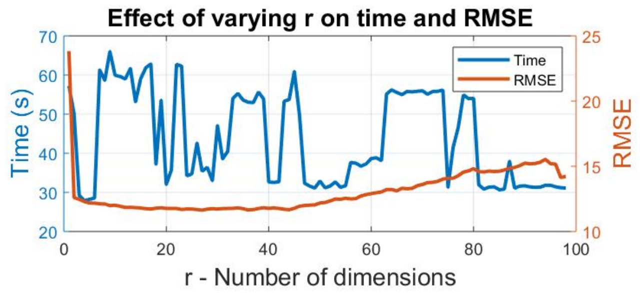


Figure 4: Execution time and RMSE for different values of principal components in PCR

III. Results

Here we analyse the results of our chosen approach and compare them with other methods tried.

III.A. Bayesian classification and Linear regressor

Figure A5 shows the confusion chart of the Bayesian classifier with optimal hyper-parameters as calculated in section II.A. We have an accuracy of 99.2% and almost all the misclassification involves class 5 misclassified as class 6. This suggests that the two movements originate from very similar neural responses.

Regarding the regressor, we make 8 regressor models, one per class, then a model is used following the classifier decision. We obtained an RMSE of 11.85 cm after using the best hyper-parameters chosen in section II.B.

III.B. Performance comparison to ready-made algorithms

Method	Bayes	NN	ECOC-SVM	KNN
Accuracy(%)	99.17	99.37	98.9	95.05
Runtime(s)	1.39	188.16	49.3	1.29

Table 1: Performance of various classifiers. NN: Neural Network, ECOC-SVM: Error Correcting Output Code Support-Vector Machine, KNN: k-nearest neighbours

Method	PCR	OLMS	Mul. Gaussian	Mean values
RMSE	11.64	12.38	25.44	17.63
Runtime (s)	36.54	21.54	6.52	6.74

Table 2: Performance of various regressors calculated with 100% classification accuracy. PCR: Principal Component Regression, OLMS: Ordinary Least Mean Square method, Multi. Gaussian: Multivariate Gaussian

IV. Discussion

IV.A. Model advantages, limitations and improvements

The results presented in this report show that it is possible to decode and predict hand position from neural cortex inputs. This opens the potential of hand neuro-prosthetics. A real life application of a position estimator algorithm would use all past data as training data and would only need to predict one movement at a time. This model is good as its execution time is low (1.39 seconds for 20 predicted movements).

The presented model allows for the hand position estimation in 8 movements in 2-dimensional plane. These movements are linear patterns which do not match real-life movement patterns where a movement may stop or change direction at any chosen time. A more complex algorithm will need to be developed to generalise movement patterns to continuous set of movements in a 3-dimensional space and the possibility of stopping a movement. One limiting factor of the constructed model is the multi-class classification step which bounds the model to fit any set of neural data to the 8 movement patterns it has been trained on. The question remains if a model could further learn from a limited set of n movements and correlate neural data to movement patterns and expanding beyond the n learned movements. Another limitation of the model is the high RMSE. Human movements are very precise, it is valuable to acknowledge the best model obtained still predicts movement with an RMSE of 11 cm. Finally, this study was also limited by the scarcity of the available data for such a complex problem as a multi-output time-dependent neural decoding. More data would improve prediction accuracy and in-vivo implementation of neuro-prosthetic would yield an increase in performance over time.

The presented model would be improved by increasing the data available for training as well as by the means of a regression-only position estimator as opposed to the presented classification-regression position estimator.

V. Conclusion

A combined classification regression model for hand position estimation was built with a RMSE value of 11.8 cm. Future work would need more data and a regression only model for improved accuracy of real-life movement patterns.

VI. Authors contributions

FG wrote code for Bayesian and KNN classifiers and PCR regressor, wrote code to organise the data to be fed to the classifiers and regressors. Wrote corresponding parts in the report. **LCR** wrote code for OLS and FitECOC-SVM classifier (using MatLab Toolbox), wrote code for preliminary analysis of the raw data & wrote abstract, introduction, discussion and conclusion. **DO** performed parameter optimisation code, worked on NN classifier (using MatLab Toolbox) and plotted the RMSE graphs.

References

[1] J. D. Cook. Making a singular matrix non-singular. Link here.

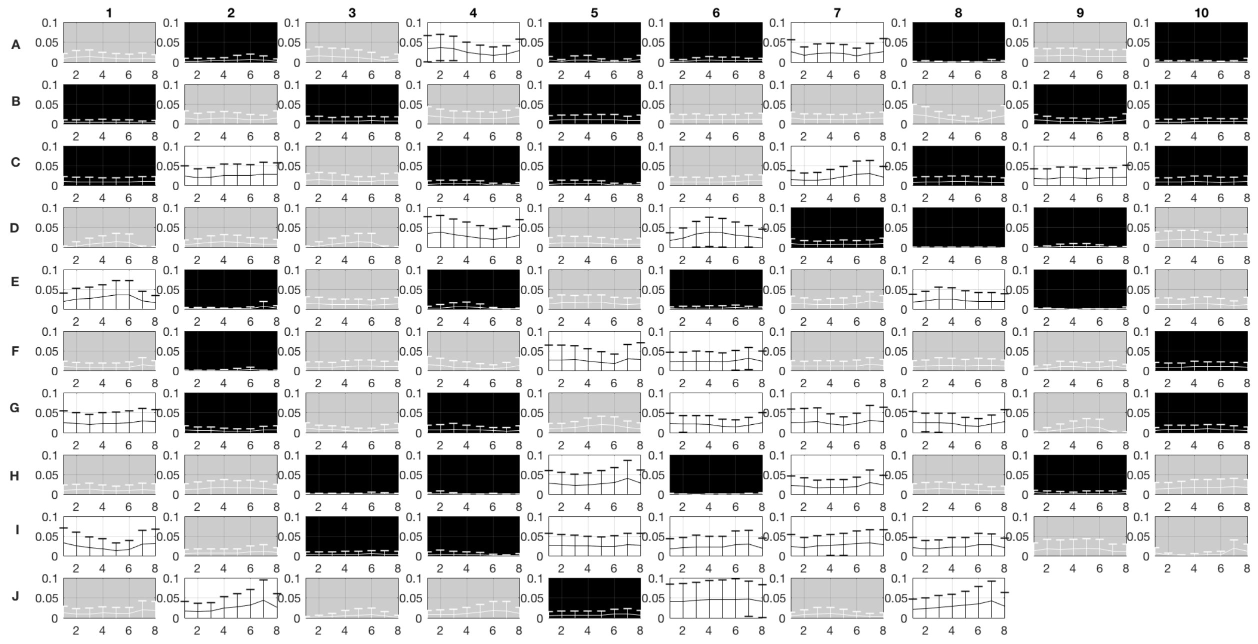
[2] F. Cordella, A. L. Ciancio, R. Sacchetti, A. Davalli, A. G. Cutti, E. Guglielmelli, and L. Zollo. Literature Review on Needs of Upper Limb Prosthesis Users. *Frontiers in neuroscience*, 10:209, 2016.

[3] A. Faisal. Machine learning and neural computation lecture notes. *Imperial College London*, 2018.

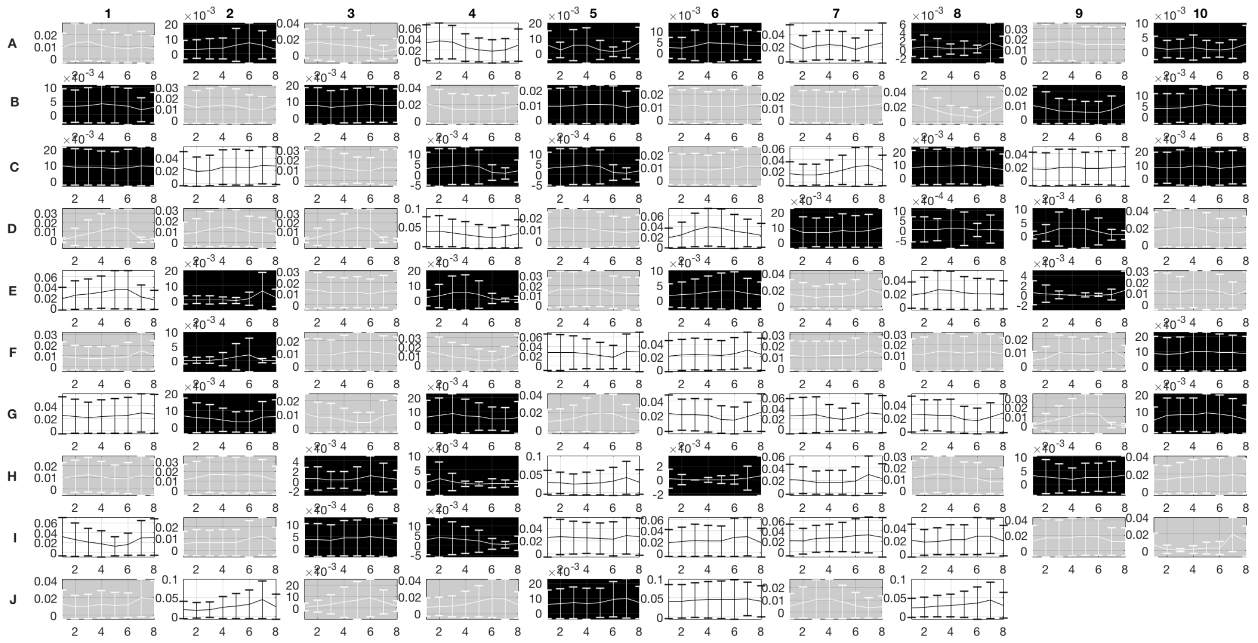
[4] T. Jolliffe, I. A note on the use of principal components in regression. New York: John Wiley & Sons, 1982.

[5] K. P. Shung. Accuracy, precision, recall or f1? 2018.

A. Appendix - Supplementary plots

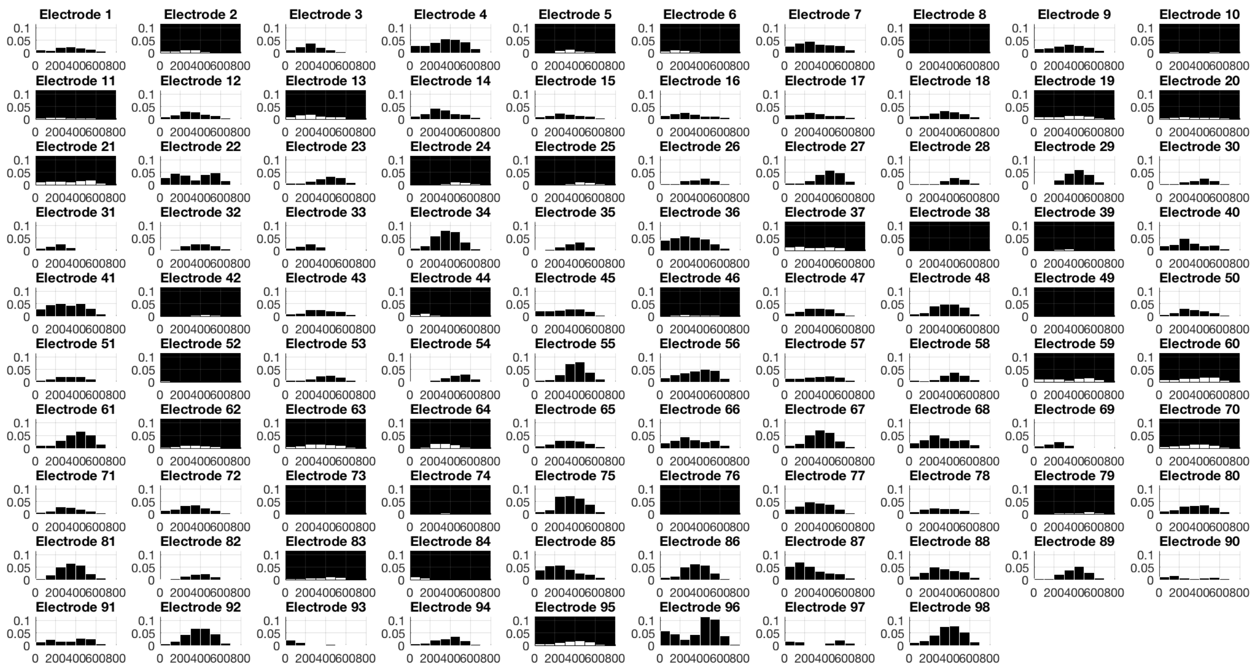


(a) Normalised Tuning curves (with fixed y-axis)



(b) Not normalised tuning curves (with varying magnitudes on the y-axis)

Figure A1: (a) The lessen signal intensity sensed by some electrodes becomes evident when looking at the normalised tuning curves, though these electrodes may still sense specifically the signal for a a given pattern as seen in (b). In black are the electrodes which maximum signal sensed is less than 0.025 spikes/trial/ms, in grey are the electrodes which sense signal of less than 0.05 spikes/trial/ms and in white are the others.


(a) Normalised peri-stimulus time histogram(PSTH)

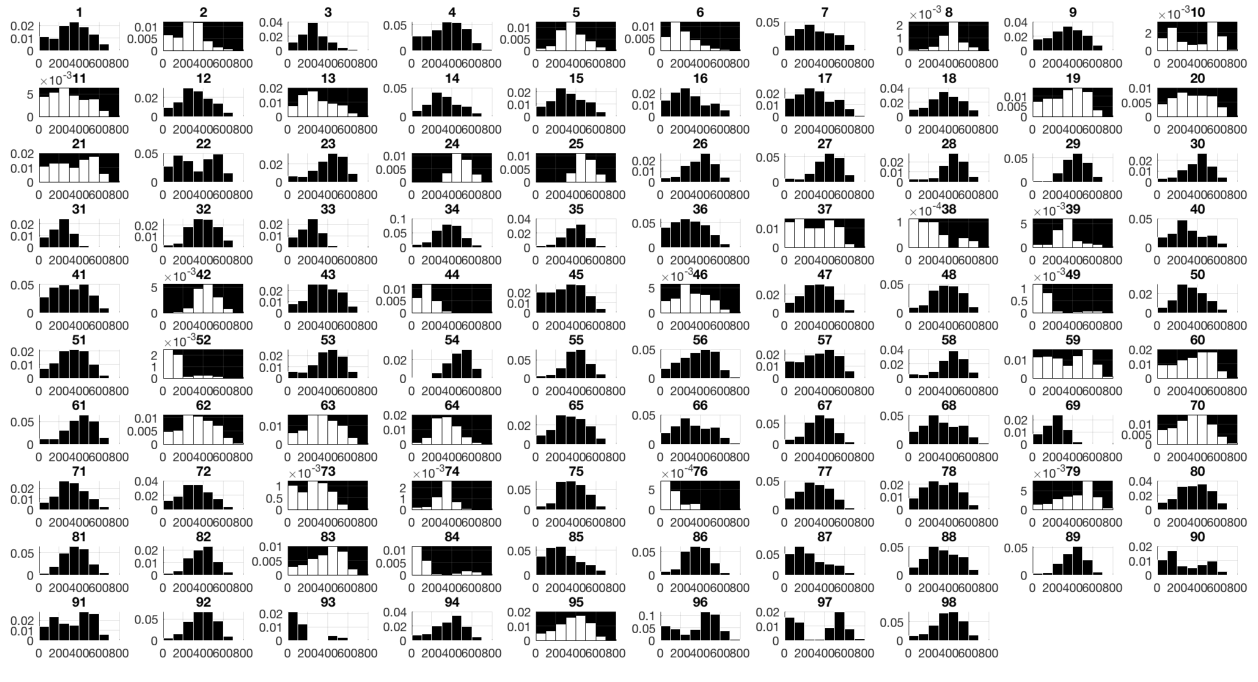

(b) Not normalised PSTH

Figure A2: Both figures show the average PSTH for all 8 movements for every one of the 98 electrodes with time windows of 100 ms. (a) Shows normalised plots where the y-axis is fixed to fit all PSTH graphs. (b) shows all PSTH graphs where the axis is dependent on each PSTH. It may initially seem that some electrodes are inactive but those electrodes still can sense varying signals that definitely come from the activation/inactivation of recorded neurons. *In black are the electrodes which signal maximum is smaller than 0.025 spikes/trial/ms*

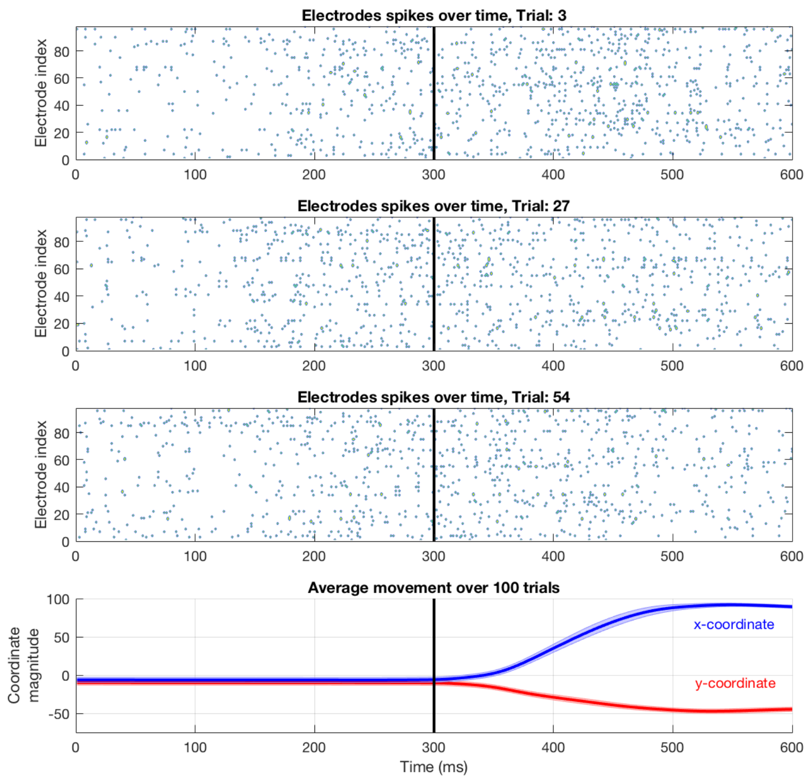


Figure A3: Raster plot of all electrodes over time for distinct trials for movement 3

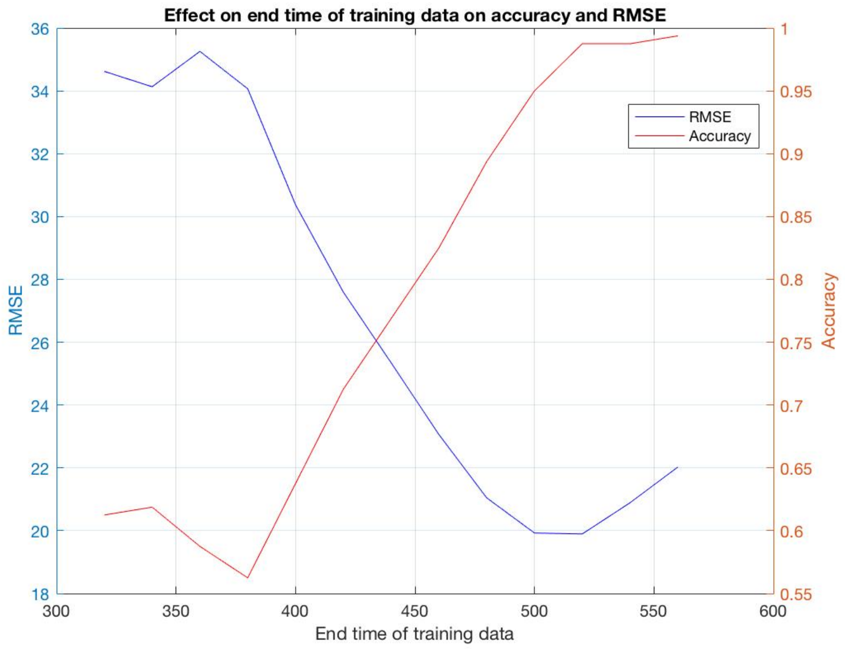


Figure A4: Accuray and RMSE plotted over several *train time* values for the Error correcting Output Code - SVM. It is clear that the best RMSE is achieved for an endtime of 380, in other words the classifier performs at its best when it has been trained on data up to 380 ms.

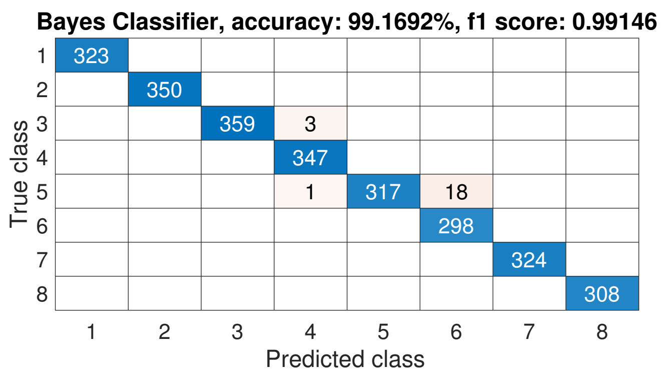


Figure A5: Confusion Chart of the optimised Bayesian classifier

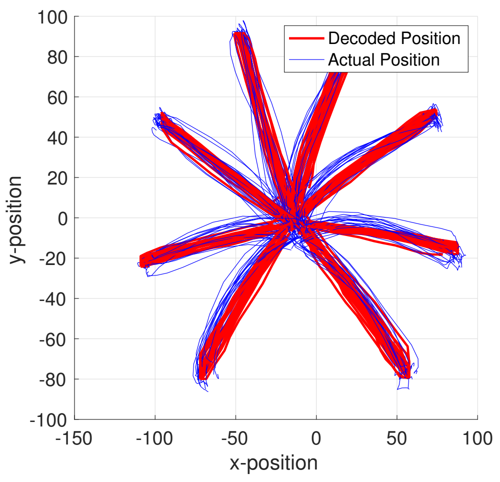


Figure A6: Position decoded using the Bayes classifier and linear regressor approach plotted together with the actual position

B. Appendix - Code

All code used in the making of this report can be found in the following *GitHub* repository: *BMImonkeysdecodemonkeys*. The **positionEstimator.m** and **positionEstimatorTraining.m** functions code can be found below.

Training Function: `positionEstimatorTraining.m`

```
%%% Team Members: Francesco Guagliardo, Luis
%%% Chaves Rodriguez, Daniele Olmeda, Arun Paul
%%% Bayes implementation
function  [modelParameters] = positionEstimatorTraining(trainingData)

% Arguments:

% - training_data:
%     training_data(n,k)              (n = trial id,  k = reaching angle)
%     training_data(n,k).trialId      unique number of the trial
%     training_data(n,k).spikes(i,t)  (i = neuron id, t = time)
%     training_data(n,k).handPos(d,t) (d = dimension [1-3], t = time)

%
%time_range = 1:360;%280:480;
%[data_formatted, labels] = tidy_spikes(trainingData,time_range);
[n,k] = size(trainingData);
[i,t] = size(trainingData(1,1).spikes);
cov_condintioning = 27;
train_times = 320:20:400;
data_formatted_per_train_time = struct;
tic
for end_t = 1:1:length(train_times)
    [data_formatted, labels] = tidy_spikes(trainingData,1:train_times(end_t));
    data_formatted_per_train_time(end_t).data_formatted = data_formatted;
    num_train_pts = size(data_formatted,1);
    % param(1).smth is not class 1, pram(2).smth is class 1, param(3) is
    % not class 2 and param(4) is class 2 and so on
    parameters = struct;
    parameters.num_classes = k;
    id = 1;
    for c = 1:k
        logical_arr = logical(labels==c);
        %divide classes (NC is no class,
        train_NC = data_formatted(~logical_arr,:); % no class is NC
        train_C = data_formatted(logical_arr,:); % yes class is C

        % priors: p(class)
        parameters(id).prior = size(train_NC,1)/num_train_pts;
        parameters(id+1).prior  = 1-parameters(id).prior;

        %means
        parameters(id).mu = mean(train_NC);
        parameters(id+1).mu = mean(train_C);

        %covariances
        parameters(id).s = cov(train_NC)+eye(size(train_NC,2))*cov_condintioning;
        parameters(id+1).s = cov(train_C)+eye(size(train_NC,2))*cov_condintioning;

        %upadte id for next class
        id = id+2;

    end
    data_formatted_per_train_time(end_t).parameters = parameters;
end
toc
% regressor
data_formatted_train = prepare_regressor_data(trainingData,'train');
r = 36;
fix_pos = 6;

for ang = 1:k
    %get x positin from processed training data
    x_position = data_formatted_train(ang).out(:,1);
    %get y position from processed training data
    y_position = data_formatted_train(ang).out(:,2);

    % get length of data
    length_data_in = length(data_formatted_train(1).in);
    %get processed spike data and concatenate to a colum of ones to prepare it
    %for the regress function
    processed_electrodes = [ones(length_data_in,1),data_formatted_train(ang).in];

    params_x = train_regressor(x_position,processed_electrodes,r,1);
    params_y = train_regressor(y_position,processed_electrodes,r,1);
    %store coefficient for this movement
    coeffs(:,:,ang) = [params_x,params_y];

    % get max and mins for x and y in order to later bound estimations

    max_x = max(x_position);
    if max_x < 0, max_x = max_x +fix_pos;
    else,  max_x = max_x -fix_pos; end
    max_y = max(y_position);
    if max_y < 0, max_y = max_y +fix_pos;
    else,  max_y = max_y -fix_pos; end
    min_x = min(x_position);
    if min_x < 0, min_x = min_x +fix_pos;
    else,  min_x = min_x -fix_pos; end
    min_y = min(y_position);
    if min_y < 0, min_y = min_y +fix_pos;
    else,  min_y = min_y -fix_pos; end

    maxs_mins(:,:,ang) = [ min_x max_x;min_y max_y];

end
modelParameters.train_in = data_formatted_per_train_time;
modelParameters.labels = labels;
% regressor
modelParameters.coeffs = coeffs;
modelParameters.extremes = maxs_mins;
modelParameters.new_dim = r;
end

% format the data in a way
function [data_formatted, labels] = tidy_spikes(data_to_format,range)
[n,k] = size(data_to_format);
[i,t] = size(data_to_format(1,1).spikes);

% output in train_trials trials x 98
labels = zeros(n*k,1);
dimensions = 1:i;%[3,4,7,18,27,31,33,34,36,41,55,68,69,75,81,90,92,98];
i = length(dimensions);
data_formatted = zeros(n*k,i);
count = 1;
for a = 1:k
    for t = 1:n % number of trials
        for el = 1:i
            data_formatted(count,el) = red_dim(data_to_format(t,a).spikes(
                dimensions(el),range));
        end
        labels(count,1) = a;
        count = count +1;
    end
end

end

% function to agglomerate the data
function reduced_dimension_data = red_dim(data_in)

reduced_dimension_data = sum(data_in);

end

function b = train_regressor(y,X,r,option)
% this linear regressor takes as an input your feature space, concated to a
% vector of ones as the constant term which will give the bias or
% "y-intercept"
% we consider y and X given as column vector where time is in the rows

if option == 0
    b = inv(X'*X)*X'*y;
else
    [Ur,Sr,Vr] = svds(X,r);
    b = Vr/Sr*Ur'*y;
end
end

function data_out = prepare_regressor_data(data_to_format, train_or_test)
% train_or_test = 'train' prepares training data, train_or_test = 'test'

% get data size: n: trials(100), k: movements/angles(8), i: electrodes (98), t:
```

```
% time (variable length)
[n,k] = size(data_to_format);
[i,t] = size(data_to_format(1,1).spikes);

%use only "useful" electrodes
dimensions = 1:i;%[3,4,7,18,27,31,33,34,36,41,55,68,69,75,81,90,92,98];
%[3,4,7,18,27,31,33,34,36,41,55,68,69,75,81,90,92,98];
%[3,4,18,34,36,96];%1:i; %electrodes used, some are useless so we shouldn't use
    them

end_time = 540; %ms
start_time = 320; %ms
step_time = 20; %ms
%time vector over which spikes will be calculated, they will be calculated
%in the form: sum(spikes(1:320)), sum(spikes(1:340))...
times = start_time:step_time:end_time;
% dim_reducer "combines" electrodes by adding the spikes of every
% consecutive 3, dim_reducer MUST BE A FACTOR OF THE NUMBER OF ELECTRODES
% WE ARE USING, OTHERWISE TROUBLE
dim_reducer = 1;%14; % final dimensions will be initial dimensions / dim_reducer
if strcmp(train_or_test,'train')
    % .in(20,30) contains the sum of the spikes up to time 320ms of
    % electrode number 30 for trial 20.
    % .in(120,30) contains the sum of the spikes up to time 340ms
    % (if step_time = 20) electrode 30 for trial 20
    % .out(20,:) contains the x and y position for trial 20 at time stamp
    % 320ms, .out(120,:) contains the x and y for trial 20 at time stamp
    % 340 ms and so on.
    %                     Electrode 1 | Electrode 2 | Electrode 3 ...
    %Trial 1 - 1:320ms   | sum(spikes)|
    %Trial 2 - 1:320ms   |
    %Trial 3 - 1:320ms
    %        .
    %        .
    %        .
    %Trial 100 - 1:320ms
    %Trial 1 - 1:340ms
    %Trial 2 - 1:340ms
    %        .
    %        .
    %        .
    %Trial 100 - 1:540ms

    for a = 1:k
        %cumulative sums: initialise with 100 (# of trials)*12 (recording
        %times for every trial) rows and as many columns as you are using
        %electrodes
        data_formatted(a).in = zeros(n*length(times),length(dimensions));
        %data_formatted(a).out = zeros(length(times),2); %x,y
        %data_out(a).in = zeros(n*length(times),reduced_dimensions);
        %output is the x,y trajectories over time
        data_out(a).out = zeros(length(times),2); %x,y
        count = 1;
```

```matlab
        %for all times (every 20 ms)
        for tim = times
            for t = 1:n % number of trials
                %store handposition for every trial and every movement at
                %precised time (320, 340, 360 ... 540)
                data_out(a).out(count,:) = data_to_format(t,a).handPos(1:2,tim);

                %for all electrodes sum input data (data_to_format) from 1
                %to tim (1 to 320, 1 to 340, 1 to 360...)
                for el = dimensions
                    data_formatted(a).in(count,el==dimensions) = sum(
                        data_to_format(t,a).spikes(el,1:tim));
                end
                %increase handpos storage vector
                count = count +1;
            end
        end
        % reduce data by combining data for every consecutive electrodes,
        % this will be part of the function output along with the x and y
        % positions. reduce_feat_dim takes as input the formatted data for
        % all original dimensions and the dim_reducer factor
        data_out(a).in = reduce_feat_dim(data_formatted(a).in,dim_reducer);

        %[data_out(a).in, coeff_pca] = reduce_feat_dim(data_formatted(a).in, 8);
        %data_out(a).coeff_pca=coeff_pca;
        data_out(a).coeff_pca=0;
    end

    %if only preparing data for testing regressor then just sum spikes over
    %dimensions (# of electrodes) and then reduce dimensions
elseif strcmp(train_or_test,'test')
    data_formatted = zeros(1,length(dimensions));
    for el = dimensions
        data_formatted(el==dimensions) = sum(data_to_format.spikes(el,:));
    end
    % reduce data
    data_out = reduce_feat_dim(data_formatted,dim_reducer);%data_formatted;%
        reduce_feat_dim(data_formatted,0.65);
    %data_out = data_formatted;
else
    warning('Insert either train or test')
end
end

function reduced_features = reduce_feat_dim(features,sum_int)
%features is a obervations x dimensions vector and the dimensions are
%reduced by summing over dimensions sum_int by sum_int
new_dim = size(features,2)/sum_int;
%reduced feature space is created
reduced_features = zeros(size(features,1),new_dim);
start_idx = 1;
for i = 1:new_dim
    reduced_features(:,i)= sum(features(:,start_idx:start_idx+sum_int-1),2);
    %index changes every sum_int (in example case = 3),in this case
    %new_dimension(1) = old_dimension(1)+old_dimension(2)+old(dimension(3)
    start_idx = start_idx+sum_int;
end

end
```

Testing Function: `positionEstimator.m`

```
%%% Team Members: Francesco Guagliardo, Luis
%%% Chaves Rodriguez, Daniele Olmeda, Arun Paul
%%% Bayes
function [x, y, new_param] = positionEstimator(test_data, modelParameters)

% **********************************************************
%
% You can also use the following function header to keep your state
% from the last iteration
%
% function [x, y, newModelParameters] = positionEstimator(test_data,
   modelParameters)
%                  ^^^^^^^^^^^^^^^^^^
% Please note that this is optional. You can still use the old function
% declaration without returning new model parameters.
%
% *********************************************************

% - test_data:
%     test_data(m).trialID
%         unique trial ID
%     test_data(m).startHandPos
%         2x1 vector giving the [x y] position of the hand at the start
%         of the trial
%     test_data(m).decodedHandPos
%         [2xN] vector giving the hand position estimated by your
%         algorithm during the previous iterations. In this case, N is
%         the number of times your function has been called previously on
%         the same data sequence.
%     test_data(m).spikes(i,t) (m = trial id, i = neuron id, t = time)
%     in this case, t goes from 1 to the current time in steps of 20
%     Example:
%         Iteration 1 (t = 320):
%             test_data.trialID = 1;
%             test_data.startHandPos = [0; 0]
%             test_data.decodedHandPos = []
%             test_data.spikes = 98x320 matrix of spiking activity
%         Iteration 2 (t = 340):
%             test_data.trialID = 1;
%             test_data.startHandPos = [0; 0]
%             test_data.decodedHandPos = [2.3; 1.5]
%             test_data.spikes = 98x340 matrix of spiking activity

[i,t] = size(test_data(1,1).spikes);
input_len = size(test_data,1);
```

```matlab

input_time = size(test_data.spikes,2);
% up_to = 360;
% if input_time < up_to
%     time_range = 1:input_time;%280:480;
% else
%     time_range = 1:up_to;%280:480;
% end
train_times = 320:20:400;
up_to = find(train_times==input_time);
if isempty(up_to)
    up_to = length(train_times);
end


%[test_data_formatted, ~] = tidy_spikes(test_data,time_range);
[test_data_formatted, ~] = tidy_spikes(test_data,1:train_times(up_to));
label = zeros(size(test_data,1),1);
parameters = modelParameters.train_in(up_to).parameters; % parameters up to that
    point

for i = 1:input_len %iterate input datapoints
    id = 1;
    prediction = zeros(parameters(1).num_classes,1);
    for c = 1:parameters(1).num_classes % iterate classes
        %probabilty p( x | class) --> pxc
        pxc_NC = multivar_gauss(test_data_formatted(i,:),parameters(id).mu,
            parameters(id).s);
        pxc_C = multivar_gauss(test_data_formatted(i,:),parameters(id+1).mu,
            parameters(id+1).s);

        % posterior probability for data point i p(C | x)
        pcx_NC = pxc_NC * parameters(id).prior;
        pcx_C = pxc_C * parameters(id+1).prior;

        if pcx_NC < pcx_C % yes class x
            prediction(c,1) = pcx_C;
        end
        id = id+2;
    end
    [~, predicte_label] = max(prediction);
    label(i,1) = predicte_label;
end % iterate input data points

%label = direc;
% regressor
test_input = prepare_regressor_data(test_data,'test');
coeffs = modelParameters.coeffs;
maxmins = modelParameters.extremes;
r = modelParameters.new_dim;

params_x = coeffs(:,1,label);
params_y = coeffs(:,2,label);
```

```
[Urx,Srx,Vrx] = svds([1,test_input]',r);
prediction(1) = params_x'*Urx*Srx*Vrx';
[Ury,Sry,Vry] = svds([1,test_input]',r);
prediction(2) = params_y'*Ury*Sry*Vry';

 %prediction(1) = params_x'*[1,test_input]';
 %prediction(2) = params_y'*[1,test_input]';

% max min check
maxmins = modelParameters.extremes;
min_x = maxmins(1,1,label);
max_x = maxmins(1,2,label);
min_y = maxmins(2,1,label);
max_y = maxmins(2,2,label);
if prediction(1) > max_x,  prediction(1) = max_x; end
if prediction(2) > max_y,   prediction(2) = max_y; end
if prediction(1) < min_x, prediction(1) = min_x; end
if prediction(2) < min_y,  prediction(2) = min_y; end



x = prediction(1);
y = prediction(2);
%x = modelParameters.mean_vals(label).mean_pos(1,t);
%y = modelParameters.mean_vals(label).mean_pos(2,t);
modelParameters.test_label = label;
new_param = modelParameters;
end

% format the data in a way
function [data_formatted, labels] = tidy_spikes(data_to_format,range)
[n,k] = size(data_to_format);
[i,t] = size(data_to_format(1,1).spikes);

% output in train_trials trials x 98
labels = zeros(n*k,1);
dimensions = 1:i;%[3,4,7,18,27,31,33,34,36,41,55,68,69,75,81,90,92,98];
i = length(dimensions);
data_formatted = zeros(n*k,i);
count = 1;
for a = 1:k
    for t = 1:n % number of trials
        for el = 1:i
            data_formatted(count,el) = red_dim(data_to_format(t,a).spikes(
                dimensions(el),range));
        end
        labels(count,1) = a;
        count = count +1;
    end
end

end
```

```matlab

% function to agglomerate the data
function reduced_dimension_data = red_dim(data_in)

reduced_dimension_data = sum(data_in);

end

% multivariate gaussian
function phi = multivar_gauss(x,mu,covar)
k = length(covar);
A = 1/sqrt((2*pi)^k*det(covar));
phi = A*exp(-0.5*(x-mu)*covar^-1*(x-mu)');
%phi = A*exp(-0.5*(x-mu)*pinv(covar)*(x-mu)');
end

function data_out = prepare_regressor_data(data_to_format, train_or_test)
% train_or_test = 'train' prepares training data, train_or_test = 'test'

% get data size: n: trials(100), k: movements/angles(8), i: electrodes (98), t:
% time (variable length)
[n,k] = size(data_to_format);
[i,t] = size(data_to_format(1,1).spikes);

%use only "useful" electrodes
dimensions = 1:i;%[3,4,7,18,27,31,33,34,36,41,55,68,69,75,81,90,92,98];
%[3,4,7,18,27,31,33,34,36,41,55,68,69,75,81,90,92,98];
%[3,4,18,34,36,96];%1:i; %electrodes used, some are useless so we shouldn't use
    them

end_time = 540; %ms
start_time = 320; %ms
step_time = 20; %ms
%time vector over which spikes will be calculated, they will be calculated
%in the form: sum(spikes(1:320)), sum(spikes(1:340))...
times = start_time:step_time:end_time;
% dim_reducer "combines" electrodes by adding the spikes of every
% consecutive 3, dim_reducer MUST BE A FACTOR OF THE NUMBER OF ELECTRODES
% WE ARE USING, OTHERWISE TROUBLE
dim_reducer = 1;%14; % final dimensions will be initial dimensions / dim_reducer
if strcmp(train_or_test,'train')
    % .in(20,30) contains the sum of the spikes up to time 320ms of
    % electrode number 30 for trial 20.
    % .in(120,30) contains the sum of the spikes up to time 340ms
    % (if step_time = 20) electrode 30 for trial 20
    % .out(20,:) contains the x and y position for trial 20 at time stamp
    % 320ms, .out(120,:) contains the x and y for trial 20 at time stamp
    % 340 ms and so on.
    %                     Electrode 1 | Electrode 2 | Electrode 3 ...
    %Trial 1 - 1:320ms   | sum(spikes)|
    %Trial 2 - 1:320ms   |
    %Trial 3 - 1:320ms
    %        .
    %       .
    %       .
    %Trial 100 - 1:320ms
    %Trial 1 - 1:340ms
    %Trial 2 - 1:340ms
    %       .
    %       .
    %       .
    %Trial 100 - 1:540ms

    for a = 1:k
        %cumulative sums: initialise with 100 (# of trials)*12 (recording
        %times for every trial) rows and as many columns as you are using
        %electrodes
        data_formatted(a).in = zeros(n*length(times),length(dimensions));
        %data_formatted(a).out = zeros(length(times),2); %x,y
        %data_out(a).in = zeros(n*length(times),reduced_dimensions);
        %output is the x,y trajectories over time
        data_out(a).out = zeros(length(times),2); %x,y
        count = 1;
        %for all times (every 20 ms)
        for tim = times
            for t = 1:n % number of trials
                %store handposition for every trial and every movement at
                %precised time (320, 340, 360 ... 540)
                data_out(a).out(count,:) = data_to_format(t,a).handPos(1:2,tim);

                %for all electrodes sum input data (data_to_format) from 1
                %to tim (1 to 320, 1 to 340, 1 to 360...)
                for el = dimensions
                    data_formatted(a).in(count,el==dimensions) = sum(
                       data_to_format(t,a).spikes(el,1:tim));
                end
                %increase handpos storage vector
                count = count +1;
            end
        end
        % reduce data by combining data for every consecutive electrodes,
        % this will be part of the function output along with the x and y
        % positions. reduce_feat_dim takes as input the formatted data for
        % all original dimensions and the dim_reducer factor
        data_out(a).in = reduce_feat_dim(data_formatted(a).in,dim_reducer);

        %[data_out(a).in, coeff_pca] = reduce_feat_dim(data_formatted(a).in, 8);
        %data_out(a).coeff_pca=coeff_pca;
        data_out(a).coeff_pca=0;
    end

    %if only preparing data for testing regressor then just sum spikes over
    %dimensions (# of electrodes) and then reduce dimensions
elseif strcmp(train_or_test,'test')
    data_formatted = zeros(1,length(dimensions));
    for el = dimensions
```

```
        data_formatted(el==dimensions) = sum(data_to_format.spikes(el,:));
    end
    % reduce data
    data_out = reduce_feat_dim(data_formatted,dim_reducer);%data_formatted;%
       reduce_feat_dim(data_formatted,0.65);
    %data_out = data_formatted;
else
    warning('Insert either train or test')
end
end

function reduced_features = reduce_feat_dim(features,sum_int)
%features is a obervations x dimensions vector and the dimensions are
%reduced by summing over dimensions sum_int by sum_int
new_dim = size(features,2)/sum_int;
%reduced feature space is created
reduced_features = zeros(size(features,1),new_dim);
start_idx = 1;
for i = 1:new_dim
    reduced_features(:,i)= sum(features(:,start_idx:start_idx+sum_int-1),2);
    %index changes every sum_int (in example case = 3),in this case
    %new_dimension(1) = old_dimension(1)+old_dimension(2)+old(dimension(3)
    start_idx = start_idx+sum_int;
end

end
```