Table of Contents


```
% EE239AS Homework 4

clc
clear
close all
```

Problem 4: Real Neural Data

```
ps4_data = importdata('ps4_realdata.mat');
```

Part A: ML Parameters

```
train_data = ps4_data.train_trial;
test_data = ps4_data.test_trial;
% since sizes of test and train data sets are the same, extract number of
% spikes in the same loop

n_neurons = 97;           % number of neurons per trial
n_class = size(train_data,2);       % number of classes
n_trial = size(train_data,1);       % number of trials
n_spikes_train = cell(1, n_class);
% number of spikes per spike train (train data)
n_spikes_test = n_spikes_train;
% number of spikes per spike train (test data)

% sum the total number of spikes per trial from spike train data

for i = 1:n_trial
    for j = 1:n_class
        n_spikes_train{1,j} = [n_spikes_train{1,j}, sum(train_data(i,j).spikes(:,3
        n_spikes_test{1,j} = [n_spikes_test{1,j}, sum(test_data(i,j).spikes(:,351:
        % only include the spikes in the middle of the trial due to
        % experimental procedure
    end
end
```

Part A: Model(i) Classification

```
% Model (i) Gaussian, Shared Covariance
```

```
N_k = n_trial;      % number of neurons per class is equal for all classes
N = N_k*n_class;    % total number of neurons
P_Ck = N_k/(n_class*N_k);
% prior probabilities of each class (equal for all classes)

mu_i = zeros(n_neurons, n_class);
S_k_i = cell(1, n_class);
sigma_i = zeros(n_neurons, n_neurons);
% preallocate ML parameter vectors

% calculating ML parameters for classification boundaries
for i = 1:n_class
    mu_i(:,i) = 1/(N_k)*sum(n_spikes_train{1,i},2);
    % calculate mean using results of problem 1
    cov_trial_i = zeros(n_neurons, n_neurons);
    % (x-mu)*(x-mu)' matrices for each trial
    for j = 1:n_trial
        cov_trial_i = cov_trial_i + (n_spikes_train{1,i}(:,j)-mu_i(:,i))*(n_spikes
        % sum the (x-mu)*(x-mu)' matrices for each trial
    end

    S_k_i{i} = 1/N_k * cov_trial_i;
    % calculate the S_k for each class and store into cell
    sigma_i = sigma_i + N_k/N * S_k_i{i};
    % calculate sigma (weighted sum of S_k)
end

k = zeros(n_trial, n_class);
errors = zeros(1,n_class);
% preallocate the class prediction and error vectors

% classification of test data
for n = 1:n_class
    xy = n_spikes_test{n}';
    % matrix of neuron firing rates across all trials (97x91) transposed
    for j = 1:n_trial
        for i = 1:n_class
            k(j,i) = log(P_Ck)+ mu_i(:,i)'*inv(sigma_i)*xy(j,:)'-0.5*mu_i(:,i)'...
                *inv(sigma_i)*mu_i(:,i);
            % decision function for each neuron a_k(x)
        end
    end
    [m,idx] = max(k, [], 2);
    % take the maximum of this to decide which class the neuron belongs in
    errors(n)= length(idx(idx~=n));
    % number of incorrect classifications (not matching the specified reach
    % angle)
end
accuracy = 1-sum(errors)/(n_trial*n_class);
fprintf('Model (i) Gaussian, Shared Covariance Accuracy:\n')
disp(accuracy)
% calculate accuracy of classification
```

Model (i) Gaussian, Shared Covariance Accuracy:

0.9602

Part B: Model (ii) Classification

```
% Model (ii) Gaussian, Class Specific Covariance

% The code above was run using the ML parameters for Model (ii) (same mu,
% and class-specific sigmas). However, the error:

% Matrix is close to singular or badly scaled. Results may be inaccurate.

% occurs. This happens because there are neurons that do not fire very much
% across all the trials, causing the covariance matrix to be close to
% singular (due to the determinant being very close to zero). When the
% covariance matrix is singular, it is not invertible and therefore the
% classification calculations cause this error to occur.

% We can remove these neurons in order to ensure a positive definite
% covariance matrix by setting a threshold on the minimum number of spikes.
% If a neuron does not spike above the threshold across all trials, it must
% be removed. Furthermore, if one row is removed from a class, it must
% be removed from all other classes in order to preserve the dimensions.
```

Part C: Model (ii) Classification (Removing Neurons)

```
% Model (ii) Gaussian, Class Specific Covariance

% Calculate the total number of spikes per neuron for all of the classes
n_spikes_total = zeros(n_neurons, n_class);
for i = 1:n_class
    % sum the number of spikes for each neuron across the entire train
    n_spikes_class = sum(n_spikes_train{1,i},2);
    % sum the number of spikes for each neuron across all trials
    n_spikes_total(:,i) = n_spikes_class;
end

% set the minimum spike threshold number to be 10
% find the neurons which do not spike at least 10 times in each class, and
% find the unique rows
% the accuracy increases slightly with higher threshold
thres = 10;
[row,col] = find(n_spikes_total<=thres);
% find the unique rows that fall below the threshold, so they can be
% removed from all classes
row_del = unique(row);

% create a second spike train matrix to store the neurons above the
% threshold
n_spikes_train_ii = n_spikes_train;
```

```
n_spikes_test_ii = n_spikes_test;

% calculate the total number of neurons left after thresholding
n_neurons_ii = n_neurons-length(row_del);
n_spikes_total_ii = zeros(n_neurons_ii, n_class);

for i = 1:n_class
    % remove the below-threshold neurons across all classes
    n_spikes_train_ii{1,i}(row_del,:) = [];
    n_spikes_test_ii{1,i}(row_del,:) = [];

    % check that removing these neurons worked
    n_spikes_class_ii = sum(n_spikes_train_ii{1,i},2);
    n_spikes_total_ii(:,i) = n_spikes_class_ii;

end

% preallocate ML parameters
mu_ii = zeros(n_neurons_ii, n_class);
S_k_ii = cell(1, n_class);
sigma_ii = zeros(n_neurons_ii, n_neurons_ii);

% calculate ML parameters (same as above)
for i = 1:n_class
    mu_ii(:,i) = 1/(N_k)*sum(n_spikes_train_ii{1,i},2);
    cov_trial_ii = zeros(n_neurons_ii, n_neurons_ii);
    for j = 1:n_trial
        cov_trial_ii = cov_trial_ii + (n_spikes_train_ii{1,i}(:,j)-mu_ii(:,i))...
            *(n_spikes_train_ii{1,i}(:,j)-mu_ii(:,i))';
        % sum the (x-mu)*(x-mu)' matrices for each trial
    end
    S_k_ii{i} = 1/N_k * cov_trial_ii;
    % calculate the S_k for each class and store into cell
end

k_ii = zeros(n_trial, n_class);
errors_ii = zeros(1,n_class);

% classification (same as above)
for n = 1:n_class
    xy = n_spikes_test_ii{n}';
    for j = 1:n_trial
        for i = 1:n_class
            k_ii(j,i) = log(P_Ck)+ mu_ii(:,i)'*inv(S_k_ii{i})*xy(j,:)'-0.5*mu_ii(:
                inv(S_k_ii{i})*mu_ii(:,i) - 0.5*xy(j,:)*inv(S_k_ii{i})*xy(j,:)';
        end
    end
    [m_ii,idx_ii] = max(k_ii, [], 2);
    % classify with max argument of decision function
    errors_ii(n)= length(idx_ii(idx_ii~=n));
    % save number of incorrect classifications
end
accuracy_ii = 1-sum(errors_ii)/(n_trial*n_class);
```

```
fprintf('Model (ii) Gaussian, Class-Specific Covariance Accuracy:\n')
disp(accuracy_ii)
% report accuracy

% The accuracy for Model (ii) is lower than for Model (i). In theory, this
% should be higher but this result is likely due to overfitting. The
% classifier weighs the activity of the low-firing rate neurons too much,
% assigning significance to their activity because it is rare. However,
% since these neurons are most likely noise sources, this decreases the
% accuracy of the classifier.

% This explanation is consistent with the observation that accuracy
% increases slightly when a higher threshold is set, meaning that more of
% these neurons are notincluded for training.
```

```
Model (ii) Gaussian, Class-Specific Covariance Accuracy:
    0.5357
```

Part D: Naive Bayes

```
% Model (iii) Poisson

k_iii = zeros(n_trial, n_class);
errors_iii = zeros(1,n_class);

for n = 1:n_class
    xy = n_spikes_test_ii{n}';
    for j = 1:n_trial
        for i = 1:n_class
            k_iii(j,i) =  xy(j,:)*log(mu_ii(:,i)) - sum(mu_ii(:,i)') ;
            % decision function for Poisson model
        end
    end
    [m_iii,idx_y(:,n)] = max(k_iii, [], 2);
    % find max argument for classification
    idx_iii = idx_y(:,n);
    errors_iii(n)= length(idx_iii(idx_iii~=n));
end

accuracy_iii = 1-sum(errors_iii)/(n_trial*n_class);
fprintf('Model (ii) Poisson (Naive Bayes) Accuracy:\n')
disp(accuracy_iii)
```

```
Model (ii) Poisson (Naive Bayes) Accuracy:
    0.8929
```

Published with MATLAB® R2014b