STA 5107 Final Project:

Neural Decoding using an Inhomogeneous Poisson Model

Department of Statistics, FSU

Xue Huang

April 24, 2012

1. **Problem Statement**

Neurons are remarkable among the cells of the body in their ability to propagate signals rapidly. Neural coding and decoding is a neuroscience-related field concerned with how sensory and other information is represented in the brain by networks of neurons. The main goal of studying neural coding is to characterize the relationship between the stimulus and the individual or ensemble neuronal responses and the relationship among electrical activity of the neurons in the ensemble.

In this report, we will perform a statistical modeling on the data observed from research animal’s brain cortex to understand the brain mechanism and make some inferences about the external behaviors.

1. **Methodology**

**2.1 Inhomogeneous Poisson Process Model**

Both for the train set and test set, they have two variables: *kin* and *rate*. Let in denote *x-position, y-position, x-velocity* and *y-velocity* of a 2-d hand movement at time, and in denote the spiking rate of C neurons in the primary motor cortex at the same time, where the rate at each time is the number of spikes within 70*ms*.

For, we assume a simple linear Gaussian model over time. That is,

, where  (1)

where A and W can be estimated using the close form of Maximum Likelihood Estimation (MLE):

 (2)

 (3)

For , we assume a generalized linear model (GLM) with an inhomogeneous Poisson process condition on .That is



where , we can express this to be



* 1. **Model Identification**

In the train data set, both the hand state and neural activity are known. After centralized the kinematic data, we can use the close-form formula to estimate the model parameters A and W in (1), which is shown as (2) and (3).

After we get model (1), we can identify  using MLE.

For each, we assume the observation are . We need to maximize the likelihood:



Take the log-likelihood, we get:



where 

Then, we apply the Newton - Raphson method to solve this MLE. The first order derivative and second derivative are:





We get the recursive update estimate is:

 (4)

* 1. **Neural Decoding**

Once the parameters are identified, we can perform the neural decoding on the testing data. That is, we will use neural activity to infer the movement behaviors of the hand. Two inference methods are used in this report:

1. *Point Process Filter*

An efficient, deterministic estimation method to estimate is Point Process Filter. This method is based on Laplace approximation by approximate the posterior at each time using a Gaussian distribution.

We use the recursive formula:



Assume that conditioned on, all components in are independent. Therefore,

 (5)

The algorithm for Point Process Filter can be divided into two parts: time update and measurement update.

1. Time update

We approximate the posterior using a Gaussian distribution at each time *k*.

 

Then,

is also normally distributed.

The mean is computed as:



The covariance is computed as:

where 

Therefore,



 (6)

1. Measurement update

The logarithm of the posterior is

 (7)

We can approximate this posterior by a Gaussian distribution

 (8)

Then, combine (7) and (8) we can get,

 (9)

Differentiate (9) w.r.t to, we have





Differentiate again,



Let after the second differentiation, we have



Let after the first differentiation, we have



Therefore,



1. *Sequential Monte Carlo Method*

The Sequential Monte Carlo aims to estimate the sequence of hidden parameters  based only on the observed data. The algorithm of Sequential Monte Carlo is in this form:

1. Generate n samples. Set t=0.
2. Prediction: Generate the prediction set using:



1. Update: Compute the weights, and normalize them using.

* Estimate using.
* Resample from the set with probabilities n times to obtain the samples 

1. Set t=t+1, and return to Step (ii).
2. *Comparison between Point Process filter and Sequential Monte Carlo Method*

* Estimation accuracy

Error is commonly used as a criterion to measure the estimation accuracy:

Let denote the true state and denote the estimate. Then,



We compute the estimation accuracy of the positions using Error.

* Computation time

1. **Experimental Result**
2. *Point Process Filter*

Using the Point Process Filter, we get the estimate for the hand movement. Figure 2 shows the comparison between the estimation from Point Process Filter and the true data value; we can see that the estimated value is very close to true hand position.



Figure 2: Plot of the true and estimated hand positions using Point Process Filter

1. *Sequential Monte Carlo Method*

Figure 3-6 shows the plots of the true data value and estimated hand positions and velocity using SMCM with sample size n= 20, 50, 100 and 500. From the evolution of the estimate of the hand positions, we can see that as the sample size increase, Sequential Monte Carlo Method performs better.



Figure 3: Plot of the true and estimated hand positions and velocity

using SMCM with n=20



Figure 4: Plot of the true and estimated hand positions and velocity

using SMCM with n=50



Figure 5: Plot of the true and estimated hand positions and velocity

using SMCM with n=100



Figure 6: Plot of the true and estimated hand positions and velocity

using SMCM with n=500

1. *Comparison between Point Process filter and Sequential Monte Carlo Method*

From Table 1 and 2, we can see Point Process filter method over perform sequential Monte Carlo methods both in estimation accuracy and computation time, which is a more accurate and fast method.

* Estimation accuracy

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | X\_pos | Y\_pos | X\_velocity | Y\_velocity |
| Point Process Filter | 0.3955 | 0.6542 | 0.4751 | 0.7571 |
| Sequential MCM |  |  |  |  |
| n=20 | 0.2378 | 0.6626 | 0.3525 | 0.6549 |
| n=50 | 0.2887 | 0.6614 | 0.4525 | 0.7133 |
| n=100 | 0.4012 | 0.6588 | 0.4411 | 0.739 |
| n=500 | 0.3155 | 0.6542 | 0.4843 | 0.7497 |

Table 1: Estimation accuracy of Point Process filter and SMCM (n=20, 50, 100 and500)

* Computation time

|  |  |
| --- | --- |
|  | time |
| Point Process Filter | 0.5594s |
| Sequential MCM |  |
| n=20 | 26.7221s |
| n=50 | 68.3409s |
| n=100 | 137.5117s |
| n=500 | 695.2096s |

Table 2: Computation time of Point Process filter and SMCM (n=20, 50, 100 and500)

1. **Summary**

In this report, we perform statistical modeling on the data to make statistical inferences about the external behaviors of the research animals. Two inference methods are used: Point Process filter and Sequential Monte Carlo Method. Both of them are good estimate for the hand movement.

At last, we make comparisons between these two methods from two aspects: estimation accuracy and computation time. We find that Point Process filter method performs better and consistent result than Sequential Monte Carlo Methods. But as sample size increase, Sequential Monte Carlo Methods has increasing higher estimation accuracy. Besides, Point Process filter compute faster than Sequential Monte Carlo method, which shows Point Process filter is a more efficient modeling analysis method for an inhomogeneous Poisson Model.

1. **Appendix (Matlab Code)**

clear all; close all;

%%%%%%%%%% Model Identification %%%%%%%%%%%

load('C:\Users\xue\Desktop\final\_train');

[M,C] = size(rate); % y

[M,d] = size(kin); % x

% centralize the kinematic data

kin = kin - ones(M,1) \* mean(kin);

% MLE close formula to estimate the model parameters A, W

a = kin(2:M,:)';

b = kin(1:M-1,:)';

A = a \* b' \* inv(b\*b');

W = (a - A \* b) \* (a - A \* b)'/(M-1);

% Inhomogenous Poisson Process

kin = [ones(M,1), kin];

for i = 1:C

% initial

theta\_old = zeros(d+1,1);

theta\_new = theta\_old + 1;

% Newton-Raphson Method

while (norm(theta\_new - theta\_old)>1e-2)

theta\_old = theta\_new;

first\_deriv = kin'\* rate(:,i) - kin' \* exp(kin\*theta\_old);

second\_deriv = - kin'.\*(ones(d+1,1) \* exp(kin\*theta\_old)')\*kin;

theta\_new = theta\_old - inv(second\_deriv) \* first\_deriv;

end

theta(:,i) = theta\_new;

end

%%%%%%%%%%%% Neural Decoding with Point Process Filter %%%%%%%%%%%%%%%%

clear('M','C','d','kin','rate');

load('C:\Users\xue\Desktop\final\_test');

tic;

[M,C] = size(rate); % y

[M,d] = size(kin); % x

mu = theta(1,:);

alpha =theta(2:5,:);

W\_kk\_1(1,:,:) = eye(d);

W\_kk(1,:,:) = eye(d);

X\_kk\_1(:,:) = zeros(d,1);

X\_kk(:,:) = zeros(d,1);

for i = 2:M

%% time update

W\_kk\_1(i,:,:) = A \* squeeze(W\_kk(i-1,:,:)) \* A' + W;

X\_kk\_1(:,i) = A \* X\_kk(:,i-1);

%% measurement update

add1 = zeros(d,d);

for c = 1:C

add1 = add1 + alpha(:,c)\* exp(mu(c)+alpha(:,c)'\*X\_kk\_1(:,i))\*alpha(:,c)';

end

W\_kk(i,:,:) = inv(inv(squeeze(W\_kk\_1(i,:,:))) + add1);

add2 = zeros(d,1);

for c = 1:C

add2 = add2 + (rate(i,c) - exp(mu(c) + alpha(:,c)' \* X\_kk\_1(:,i)))\*alpha(:,c);

end

X\_kk(:,i) = X\_kk\_1(:,i) + squeeze(W\_kk(i,:,:))\* add2;

end

t1 = toc;

% R\_square error

kin\_mean = mean(kin);

kin = kin - ones(M,1) \* kin\_mean;

X\_kk = X\_kk + mean(kin)'\* ones(1,M);

R\_square = 1- sum((X\_kk'-kin).^2)./sum(kin.^2);

% plot for the true and estimated position and velocity

figure(1);

tl = ['px','py','vx','vy'];

for i = 1:d

subplot(4,1,i);

plot(1:M,X\_kk(i,:),1:M, kin(:,i)');

xlabel('time');

ylabel(tl(2\*(i-1)+1:2\*(i-1)+2));

end

%%%%%%%%%%%% Neural Decoding with Sequential Monte Carlo method %%%%%%%%%%%%%%%%

clear('M','C','d','kin','rate');

load('C:\Users\xue\Desktop\final\_test');

tic;

[M,C] = size(rate); % y

[M,d] = size(kin); % x

n = 500;

X\_h(:,:,1)=rand(d,n);

% X\_h(:,:,i)=rand(d,n);

for i = 2:M

% Prediction, generate the prediction set

X\_tilte(:,:,i) = A \* squeeze(X\_h(:,:,i-1)) + mvnrnd(zeros(d,1), W, n)';

% Compute the weights

for j = 1:n

LL = zeros(d,1);

for c = 1:C

lamda(i,c) = exp(mu(1,c) + alpha(:,c)' \* X\_tilte(:,j,i));

LL(c,1) = -lamda(i, c) + log(lamda(i, c))\* rate(i,c) -log(factorial(rate(i,c)));

end

w(i,j) = exp(sum(LL));

end

w2(i,:) = w(i,:)/sum(w(i,:));

weight(i,:) = cumsum(w2(i,:));

theta\_hat(i,:) = w2(i,:)\* squeeze(X\_tilte(:,:,i))';

% resample

for j=1:n

U = rand;

ind = find(U < weight(i,:));

X\_h(:,j,i) = X\_tilte(:,ind(1),i);

end

end

t2 = toc;

kin = kin - ones(M,1) \* mean(kin);

R\_square = 1- sum((theta\_hat' - kin').^2,2)./sum(kin'.^2,2);

figure(2);

tl = ['px','py','vx','vy'];

for i = 1:d

subplot(4,1,i);

plot(1:M,theta\_hat(:,i),1:M, kin(:,i)');

xlabel('time(n=500)');

ylabel(tl(2\*(i-1)+1:2\*(i-1)+2));

end