Computational Methods in Statistics I - STA 5106: Final Project

TANMOY DAS

1.0 Introduction

The main purpose of the project is to use dynamic programming method to reconstruct a binary function from a noisy observation.

A binary & marcovian sequence is considered & the reconstruction of this function by maximum a posteriori method is going to be conducted. In this project, I introduce a reconstruction technique to filter data considering noisy observation. In Bayesian statistics, a maximum a posteriori probability (MAP) estimate is a mode of the posterior distribution.

For given probability, binary function is simulated and noisy observation is also simulated with probable variance. Then, dynamic programming approach is applied using Matlab to estimate MAP.

2.0 Methodology

Dynamic programming:

Dynamic programming is a method for solving complex problems by breaking them down into simpler sub-problems. It is designed to solve optimization problem. Dynamic programming also divides a problem into several sub-problems. However, some sub-problems may share sub-problems. Dynamic programming saves time by computing each sub-problem only once. To do so, it must be conducted in a bottom-up fashion.

Dynamic programming takes advantage of the duplication and arrange to solve each sub problem only once, saving the solution (in table or in a globally accessible place) for later use. The underlying idea of dynamic programming is: avoid calculating the same stuff twice, usually by keeping a table of known results of subproblems. Dynamic Programming has emerged as an important tool in finding optimal paths in a variety of situations. This tool is useful in finding an optimal trajectory in situations where the decisions are made in stages and decisions made in past stages have future consequences.

Dynamic Programming is a powerful technique that can be used to solve many problems in time $O(n^2)$ or $O(n^3)$ for which a naive approach would take exponential time.

Maximum A Posteriori:

The MAP estimation framework provides a way of incorporating prior information in the training process, which is particularly useful for dealing with problems posed by sparse training data for which the ML approach gives inaccurate estimates. Suppose, we have a model with two parameters A and B and we achieve high posterior probability by either setting A to a high value and B to a low value or the other way around, setting A to a low and B to a high value. In a MAP estimate, we are not getting an understanding of such parameter correlations, but we can measure such correlations using posterior sampling.

MAP estimation can be applied to two classes of applications, namely, parameter smoothing and model adaptation, both related to the problem of parameter estimation with sparse training data.

In order to compute the discrete pixel values in the digital image, it is necessary to sample the original continuously defined image function, which is successfully accomplished in this project.

Problem Formulation:

$x(t)$ is discretized as $x = (x_1, \ldots, x_N)$

Let $x = (x_1, \ldots, x_n)$ be a given binary and Markovian sequence. In particular,

$$x_1 = \begin{cases} 0 \text{ with probability } 0.5 \\ 1 \text{ with probability } 0.5 \end{cases}$$

and

$$x_i = \begin{cases} x_{i-1} & \text{with probability } p \\ 1 - x_{i-1} & \text{with probability } 1 - p \end{cases}$$

Let $y = (y_1, \ldots, y_n)$ be a noisy observation of x, that is, for any $i = 1, \ldots, n$

$$y_i = x_i + e_i, \qquad \text{where } e_i \sim N(0, \sigma^2).$$

We want to incorporate knowledge of noise statistics. Because the sampling and reconstruction process involves approximation, it introduces error. These errors occur because the sampling process is not able to capture all of the information from the continuously defined image function.

Likelihood function for basic problem:

$$\Pr(\{y_i\} \mid \{x_i\}) = \prod_{i=1}^{N} \Pr(y_i \mid x_i) = \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{(y-x)^2}{2\sigma^2}\right)$$

Prior: Pr({x$_i$}) = $\Pr(x_1)\prod_{i=1}^{N}\Pr(x_i|\, x_{i-1}) = \frac{1}{2}\prod_{i=1}^{N} p^{1_{x_i = x_{i-1}}} + (1-p)^{1_{x_i \neq x_{i-1}}}$

Posterior: Pr({x$_i$} | {y$_i$}) = Pr({y$_i$} | {x$_i$}) Pr({y$_i$}) / Pr ({y$_i$})

Maximum A Posteriori (MAP) Estimate:

{ $\hat{x}_i$ } = argmaxPr({x$_i$}|{y$_i$}) = argmaxlog(Pr({x$_i$}|{y$_i$}))

log(Pr({x$_i$}|{y$_i$}))

$$= \log \left(\prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{(y-x)^2}{2\sigma^2}\right)\right) + \log \left(\frac{1}{2}\prod_{i=1}^{N} p^{1_{x_i = x_{i-1}}} + (1-p)^{1_{x_i \neq x_{i-1}}} \right)$$

$$= \sum_{i=1}^{N}\left(-\frac{(y-x)^2}{2\sigma^2}\right) + \sum_{i=2}^{N}\sum_{i=2}^{N} \log(1_{x_i = x_{i-1}}p + 1_{x_i \neq x_{i-1}}(1-p)) + const$$

This maximum can be computed by Dynamic Programming, which is done in later section.

Simulate x for given p

For given initial probability of 0.5, binary function x is generated in Matlab. As value of x should be binary, value is rounded to 1.

Simulate y conditioned on x for given σ^2.

Function y is calculated by adding an error e with marcovian sequence x. Error is normally distributed with mean 0, & variance σ^2

Use dynamic programming method to estimate MAP z = (z_1, …, z_n)

Suppose, A(x_1, …,x_n) = $\sum_{i=1}^{n}\left(-\frac{(y-x)^2}{2\sigma^2}\right) + \sum_{i=2}^{N} \log(1_{x_i = x_{i-1}}p + 1_{x_i \neq x_{i-1}}(1-p))$

Our aim is to Maximize this A, which is done in Matlab.

3.0 Matlab programs:

Matlab Code: 4(a)

```
clear all
% simulate x for given p
p = .99;
sigma_square = 1;
n = 1000;
X = zeros (n,1);
X(1) = round(rand(1));
for i = 2:n
    sample = rand(1);
    X(i) = X(i-1)*(sample < p ) + mod(X(i-1)+1, 2)*(sample > p);
end

% Simulate y conditioned on x for given σ^2.
e = normrnd(0, sigma_square, n,1);
Y = X + e;
% dynamic programming method to estimate MAP z
x = zeros(n,1);
S = zeros(n,2);
S(1,1) = -Y(1)^2./(2*sigma_square);
S(1,2) = -(Y(1)-1)^2./(2*sigma_square);

for k = 2:n
    for i = 0:1
        h(1) = S(k-1,1) -((Y(k)-i)^2)/(2*sigma_square) + log(p*(i==0) + (1-
p)*(i~=0));
        h(2) = S(k-1,2) -((Y(k)-i)^2)/(2*sigma_square) + log(p*(i==1) + (1-
p)*(i~=1));
        if  h(1) > h(2)
            x(k,i+1) = 0;
            S(k,i+1) = h(1);
        else
            x(k,i+1) = 1;
            S(k,i+1) = h(2);
        end
    end
end

z = zeros(n,1);
if S(n,1) < S(n,2)
    z(n) = 1;
else
    z(n) = 0;
end

for k = n-1:-1:1
    z(k) = x(k+1,z(k+1)+1);
end

% Plot x, y and the estimate z
plot(Y,'g')
hold on
plot(X,'LineWidth',2)
plot(z,'r','LineWidth',2)
hold off
axis([0 1000 -5 5 ])

h_legend = legend('Observed','True','Reconstructed');
set(h_legend,'FontSize',9);
title('Figure 4(a): Plot x, y & Estimate z; consider p = 0.99 & \sigma =
1','Fontsize',11)
xlabel('Time')
ylabel('value')
```

Matlab Code: 4(c) Perform reconstruction for different value of *p*

```
clear all
% change value of p as instructed in question
p = .9;
sigma_square = 1;
n = 1000;
X = zeros (n,1);
X(1) = round(rand(1));
for i = 2:n
    sample = rand(1);
    X(i) = X(i-1)*(sample < p ) + mod(X(i-1)+1,2)*(sample>p);
end
e = normrnd(0, sigma_square, n,1);
Y = X + e;
x = zeros(n,1);
S = zeros(n,2);
S(1,1) = -Y(1)^2./(2*sigma_square);
S(1,2) = -(Y(1)-1)^2./(2*sigma_square);

for k = 2:n
    for i = 0:1
        h(1) = S(k-1,1) -((Y(k)-i)^2)/(2*sigma_square) + log(p*(i==0) + (1-
p)*(i~=0));
        h(2) = S(k-1,2) -((Y(k)-i)^2)/(2*sigma_square) + log(p*(i==1) + (1-
p)*(i~=1));
        if  h(1) > h(2)
            x(k,i+1) = 0;
            S(k,i+1) = h(1);
        else
            x(k,i+1) = 1;
            S(k,i+1) = h(2);
        end
    end
end
z = zeros(n,1);
if S(n,1) < S(n,2)
    z(n) = 1;
else
    z(n) = 0;
end

for k = n-1:-1:1
    z(k) = x(k+1,z(k+1)+1);
end
plot(Y,'g')
hold on
plot(X,'LineWidth',2)
plot(z,'r','LineWidth',2)
hold off
axis([0 1000 -5 5 ])
h_legend = legend('Observed','True','Reconstructed');
set(h_legend,'FontSize',9);
title('Figure 4(c-1): Plot x, y & Estimate z; consider p = 0.9 & \sigma =
1','Fontsize',11)
xlabel('Time')
ylabel('value')
```

Matlab Code: 4(c) Plot average success rate w.r.t. *p*

```
% Plot averaged (over 5 repetitions) success rate w.r.t. p
p = [.9 .8 .7 .6];
r = [80.4 70 74 67];
plot(p,r,'--rs','LineWidth',2)
axis([.5 1 60 90])
title(' Figure 4(c): Averaged success rate w.r.t. p','Fontsize', 12)
xlabel('value of p')
ylabel('success rate, r')
```

Matlab Code: 4(d) Plot average success rate

```
% Plot averaged (over 5 repetitions) success rate w.r.t. \sigma
clear all
sigma = [.5 1 2 5];
r = [98.67 95.33 64.8 51.14];
plot(sigma,r,'-ro','LineWidth',2)
axis([0 6 40 100])
title(' Figure 4(d): Averaged success rate w.r.t. \sigma','Fontsize', 12)
xlabel('value of \sigma')
ylabel('success rate, r')
```

4.0 Experimental Results

(a) Implement the dynamic programming in Matlab. Choose p = 0.99 and σ = 1. Plot x, y and the estimate z. Matlab codes to implement dynamic programming is noted in Matlab program section of this report file. Matlab command to find success rate through command window:

sum(X==z)

Success rate*:* $\frac{total\ Succeeded\ value}{number\ \ of\ iteration} = \frac{959}{1000}$ = 95.9 %

We got the success is 959 by 1000, hence success rate is 95.9 by 100, i.e 95.9%

(b) Step (a) is repeated 5 times, and the averaged success rate is reported.

We assume, n = 1000

σ	1	1	1	1	1
p	.99	.99	.99	.99	.99
Reconstruction	967	987	979	955	976

Hence, Average success rate = (967+987+979+955+976)/5 = 972.8 = 97.28%

Comment: Average success rate is high when variance is 1 & *p* is 0.99

(c) Perform reconstruction for p = 0.9, 0.8, 0.7, 0.6 with σ = 1. Plot averaged (over 5 repetitions) success rate w.r.t. p.

Matlab codes to implement dynamic programming is noted in Matlab program section 4(c). Reconstruction is continued for different value of *p*. Matlab program is run several times & associated success rate is noted in the table below. Average success rate is finally plotted to figure 4(c)

Only four cases are shown in Figure 4(c-1,2,3,4) from several graphs of reconstruction found from running the Matlab program for different value of *p* as noted .9, .8, .7 & .6 . The value of σ is fixed this time.

Sigma	p	Success Rate
1	.9	(85+81+81 +81+77+77) /6 = 80.6
1	.8	(64+69+83+ 65+ 75+64) /6 = 70
1	.7	(58+71+79 + 71+90 + 75) /6 = 74
1	.6	(64+ 70+69 +60+71+ 68) /6 =67

Comment: Average success rate will be higher when value of *p* is more than .9, but before that success rate can be low or high. Reconstruction's success rate is fluctuating with the changing value of *p.*

(d) Perform reconstruction for σ = 0.5, 1, 2, 5 with p = 0.99. Plot averaged (over 5 repetitions) success rate w.r.t. σ.

Matlab program is run several times & associated success rate is noted in the table below. Average success rate is finally plotted to figure 4(d). Only two cases are shown in Figure 4(d-1,2) from several graphs of reconstruction found from running the Matlab program for different value of σ as noted .5,1,2,5. This time, value of *p* is fixed to 0.99

Sigma	σ^2	p	Success Rate
.5	.25	.99	(100+100+100 +100+95+97)/6 = 98.67
1	1	.99	(100+100+90 +92+100+90)/6 = 95.33
2	4	.99	(65+99+16+ 100+21+80 +88+50)/8 64.8
5	25	.99	(82+18+76 + 74+17+27 +64)/7 =51.14

Comment: Average Success rate is directly disproportional to the value of σ, that means success rate is decreasing with the increase of sigma when value of *p* is constant.

But, with high value of sigma, success of individual case is too random, sometimes it is too high, sometimes it is severally low. In case of lower value of σ, success is pretty high, nearly 100%, but as σ's value is increasing, reconstruction behaves randomly.

5.0 Conclusion

In the project work, the process of taking a collection of sample values and converting them back to a continuous function is developed. Reconstruction success rates in different scenario are also analyzed, like varying the occurrence probability with same variance & different variance value.

From the output, we gather knowledge about behavior of reconstructing function with a predefined probability density function & a given variance. From this project, we learnt how to deal with noisy observations. Dynamic programming is an efficient tool to reconstruct binary function from noisy cases which is successfully applied to solve the project's problem.