Optimized Newspaper Delivery Using Markov Chains

Çağatay Yıldız cagatay.yildiz1@gmail.com
Computer Engineering Department
Boğaziçi University
Istanbul, Turkey

Editor: -

Abstract

This paper explains possible approaches to optimize distribution of newspapers. The number of newspapers supplied forms a number of time series for each sale point. These time series can be used to train a Hidden Markov Model, which is later used to predict sales. Moreover, higher order Markov models and factorial models are good candidates to be trained. It turned out that it is possible to distribute newspapers more efficiently using one of these models but this also causes the demand not to be met.

Keywords: Markov Chains, Mixture Models, Bayesian Networks

1. Introduction

In this work, I tried to solve a real-world problem, that is, how many newspapers should be delivered to different sale points throughout the country. More generally, this problem can be seen as the prediction of demand for a particular product and therefore, there are a huge number of problems that are very similar to it. What's more, the production and supply of products such as daily milk and newspapers are crucial because such products have really short life spans. One or a few days after the production, they are simply garbage.

2. Dataset

The data is received from "YAYSAT Dağıtım", a company that distributes media products. It was preprocessed by Taylan Cemgil and ready to be studied. What Taylan Hoca gave me was the number of newspapers (just Hürriyet, no other newspapers) that is delivered to and returned from more than 2500 sale points in Turkey for about 250 weeks (You can see the a heat map, drawn by Taylan Cemgil, for a sale point in Figure 1). The goal of YAYSAT is simply to reduce the number of returns while meeting the demand.

One interesting point is that demand is observable if and only if some newspapers are returned. In sold-out case, all we know is the lower bound for demand. Also, a simplifying assumption is made here, that is, demand is not a distribution but just a number. This is obviously not correct in general but it makes analysis much easier.

In addition to that, I generated synthetic data for each model implemented. You can see the source code in `generate_data.m`

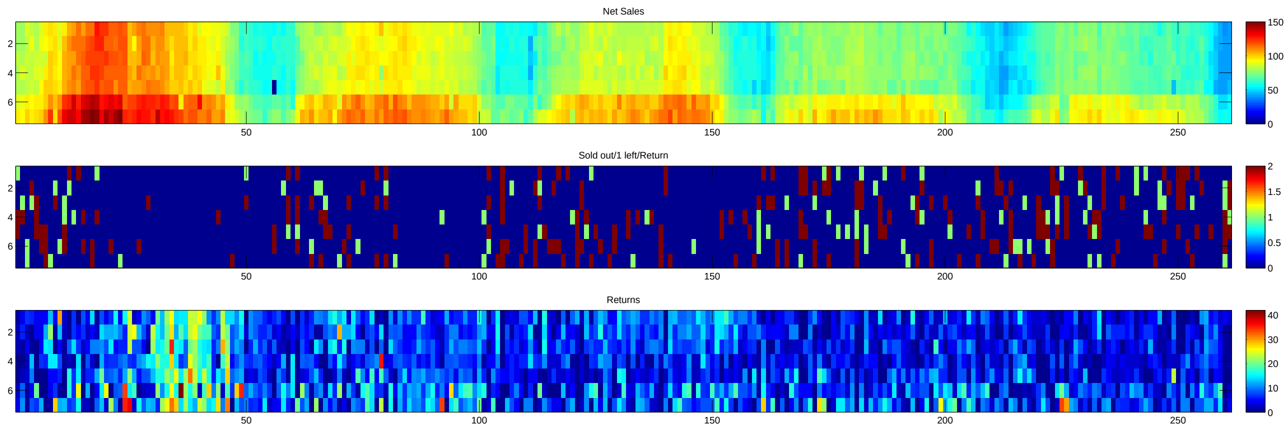

Figure 1: (i)Net sale, (ii)Return status (sold out (red), just one return (green), more returns (blue)), (iii)Return count. Each row represents a day of the week and weeks are on horizontal axis.

3. Models

The very natural way of modelling this data is through Markov chains. For each sale point there are 7 time series, each of which corresponds to different days of one week. Depending on the method of solution, it is possible to consider these time series in relation or not.

Since I have the observable demand assumption, **observable Markov chains** could fit well to the problem. However, it seemed a bit problematic considering the state space, or the greatest number of newspapers sold in whole time frame for the sale point of interest, can be very large (up to $N = 400$). Another model that is applicable to this problem is the **hidden Markov models**. Using HMM's, it could be possible to model the data in a more realistic way. Since we have already implemented EM for Markov-1 chains, I went ahead and implemented HMM using a Markov-2 chain.

I also made use of ideas presented by Saul and Jordan (1998). In their work, different techniques to represent transition matrices as convex combinations are proposed, which are called as mixed memory Markov models. First two of the three methods they presented seemed applicable to the problem:

- **Higher Order Markov Model:** This mixture model attempts to parametrize transition matrix of a Kth order Markov model as a convex combination as such:

$$P(i_t|i_{t-1}, i_{t-2}, ...i_{t-K}) = \sum_{k=1}^{K} \alpha(k) a^k(i_t|i_{t-k}) \tag{1}$$

  where I's represent observations, $\alpha(k) \geq 0$, $\sum_k \alpha(k) = 1$ and $a^k(i'|i)$ are K elementary n x n transition matrices. Here, the number of parameters reduces from $O(n^{K+1})$ to

$O(kn^2)$. Using this model, it is possible to look deeper in the sale history of a sale point.

- **Factorial Markov Model:** Factorial representations seemed quite applicable to this problem as well. Two simplifying assumptions made by Saul and Jordan are that (i)each observation at time t are conditionally independent given the vector I_{t-1} and (ii)conditional probabilities can be expressed as a weighted sum of "cross transition" matrices:

$$P(I_t|I_{t-1}) = \prod_{k=1}^{K} P(i_t^v|I_{t-1}) \tag{2}$$

$$P(i_t^v|I_{t-1}) = \sum_{k=1}^{K} \alpha(v,k) a^{vk}(i_t^v|i_{t-1}^k) \tag{3}$$

where $\alpha(v,k) \geq 0$, $\sum_k \alpha(v,k) = 1$ and $a^{vk}(i'|i)$ are K elementary n x n transition matrices. This model can be used to project the relationship between time series, or days in my data set.

4. Results

4.1 Synthetic Data

The reason why synthetic data was generated is to understand whether implementations of algorithms are correct or not. You once told me that I can check my algorithms by comparing the prior, transition and observation matrices of the generative model with those that are produced at the end of EM. I did this and saw that parameters *look like* one another but not exactly same. It was usually the case that parameters returned by EM contain more *zero* values whereas no zero occurs in the generative model parameters. I thought this is a convergence issue and attempted another way of debugging my implementation. The way I proceeded is as follows:

First, I generated some data using all generative models (4 different models and 4 dataset in total). While generating data, I considered model parameters to be as simple as possible. If they are set randomly, results were disaster. After that, I chose one dataset at a time and trained all models using 80 percent of the data. Finally, I tried to predict next 20 percent of the observations. It turned out that all models yielded best results on datasets produced by their generative models. In other words, if you have some data generated by an HMM, you can get the best predictions using an HMM.

Hidden Markov model and higher order Markov model approximation were especially great. They successfully predicted 83 and 88 percent of the remaining observations. Observable Markov model also performed great if the data is generated by its generative model and it produced some ridiculous results in other cases. Factorial Markov model was not that good, it predicted only 53 percent of the remaining data correctly. I also tried to train

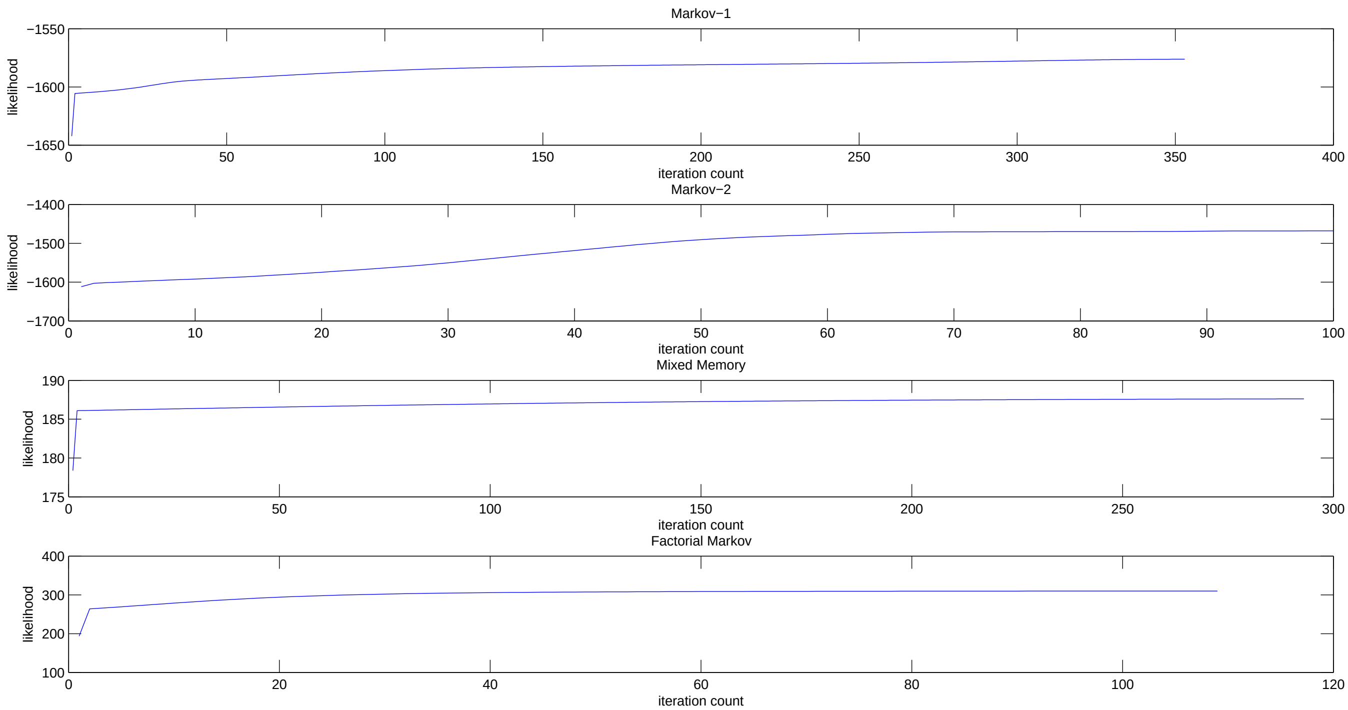

Figure 2: EM iterations vs Likelihood

a second order HMM but algorithms executes really slow when state space is not very small.

The source code for this process is in `test_data.m` I also noted the likelihood at the end of each EM iteration. You can see in Figure (2) that likelihoods never decreases.

4.2 Real Data

As noted before, for each sale point in my real data set, I have 7 time series, each having length 261. I used first 200 observation to train the model and the next 61 to test it and I did this for 25 different sale points. You can see the source code in `real_data.m`. Also, note that 23913 newspapers are returned in this period. This number is important because I am going to make comparisons with it.

- **Observable Markov Model:** Well, I did not give it a try to this model in real data set since it could not perform well even in synthetic data set.

- **Hidden Markov Model:** I considered each time series separately while training HMM's since results were really bad if one HMM is trained with 7 different time series. Using HMM reduced the number of returns to 19281, which may seem nice at the first glance. On the other hand, 14017 newspapers are missed, that is, we underestimated demand by this number in total. That is of course a huge negative. You can see `hmm_markov1.m` and `hmm1_next_sale_prediction.m` for the source code.

- **Higher Order Markov Model Approximation:** In contrast to training many HMM's, I trained only one higher order Markov model for each sale point. Results are noticeably better than HMM: Number of returns are reduced to 16453 and 9119 newspapers are missed. You can see `hmm_mm.m` and `mm_next_sale_prediction.m` for the source code. I also tried to fit a second order HMM but it executed so slow that I did not get any result. The source code is in `hmm_markov_full.m`
- **Factorial Markov Model:** In this model, I took cartesian product of each time series for a sale point. It performed slightly worse than the second model: 16963 newspapers are returned and 10309 are missed. I might have enlarged the cartesian space by considering all time series in not one but a few sale points. This would have reflected the relationships between different sale point. You can see `hmm_fm.m` and `fm_next_sale_prediction.m` for the source code.

5. Further Work

Results above are definitely not YAYSAT was looking for. The main goal of the company is to deliver newspaper, not to decrease sale. Thus, some other approaches to the problem need to be considered. A promising one would be adding hidden states to higher order Markov model approximation. In such a model, latent variables will form a Markov chain of some order and observations will be sales. What's more, I trained models just once and then made predictions but it was also possible to keep training models as new data come, that is, they can be trained not just using 200 weeks of data but using the whole time frame.

Appendix A. Derivation of the EM Algorithm for Higher Order Markov Model Approximation

This project went hand-in hand with another one that I dealt with in CMPE 547 Bayesian Statistics and Machine Learning course. Taylan Hoca asked me to derive the EM updates for higher order Markov model approximation. Here is my work:
First, see the notation: $p(x=k) \Leftrightarrow \alpha(k)$ and $A(i_t = i', i_{t-k} = i, x_t = k) \Leftrightarrow \beta(i', i, k)$

$$P(I,X) = \prod_t p(x_t) p(i_t | x_t, i_{t-1:t-K}) \tag{4}$$

$$= \prod_t \prod_k \alpha(k)^{[x_t=k]} A(i_t | i_{t-k}, k)^{[x_t=k]} \tag{5}$$

$$= \prod_t \prod_k \alpha(k)^{[x_t=k]} \prod_t \prod_k \prod_i \prod_{i'} \beta(i', i, k)^{[x_t=k][i_{t-k}=i][i_t=i']} \tag{6}$$

Then,

$$logP(I,X) = \sum_t \sum_k [x_t = k] log\alpha(k) \quad + \\ \sum_t \sum_k \sum_i \sum_{i'} [x_t = k][i_{t-k} = i][i_t = i'] log\beta(i', i, k) \tag{7}$$

Taking the expectation,

$$< logP(I,X) >_{p^{old}(X|I)} = \sum_t \sum_k p^{old}(x_t = k|I) log\alpha(k) \quad + \\ \sum_t \sum_k \sum_i \sum_{i'} p^{old}(i_t = i', i_{t-k} = i, x_t = k|I) log\beta(i', i, k) \tag{8}$$

Concentrating on the first term:

$$\frac{\partial \Big(< logP(I,X) >_{p^{old}(X|I)} + \lambda \big(1 - \sum_k \alpha(k)\big) \Big)}{\partial \alpha(k)} = 0 \tag{9}$$

$$\frac{\partial \Big(\sum_t \sum_k p^{old}(x_t = k|I) log\alpha(k) + \lambda \big(1 - \sum_k \alpha(k)\big) \Big)}{\partial \alpha(k)} = 0 \tag{10}$$

$$\frac{\sum_t p^{old}(x_t = k|I)}{\alpha(k)} - \lambda = 0 \tag{11}$$

$$\alpha(k) \propto \sum_t p^{old}(x_t = k|I) \tag{12}$$

Similarly,

$$\frac{\partial \Big(< logP(I,X) >_{p^{old}(X|I)} + \lambda \big(1 - \sum_{i'} \beta(i', i, k)\big) \Big)}{\partial \beta(i', i, k)} = 0 \tag{13}$$

$$\frac{\partial\Big(\sum_t \sum_k \sum_i \sum_{i'} p^{old}(i_t = i', i_{t-k} = i, x_t = k|I) log\beta(i', i, k) + \lambda\big(1 - \sum_{i'} \beta(i', i, k)\big)\Big)}{\partial \beta(i', i, k)} = 0 \tag{14}$$

$$\frac{\sum_t p^{old}(i_t = i', i_{t-k} = i, x_t = k|I)}{\beta(i', i, k)} - \lambda = 0 \tag{15}$$

$$\beta(i', i, k) \propto \sum_t p^{old}(i_t = i', i_{t-k} = i, x_t = k|I) \tag{16}$$

Appendix B. Source Code

```
function [pri, A, C, states, obs] = generate_data(model,K,T,N,
    order)
%model The model from which data is generated.
%K      Number of sequences
%T      Length of each sequence
%N      Number of states
%past   How many steps back to consider.

states = zeros(K, T);
obs = zeros(K, T);
if model == 1
    pri = my_normalize(ones(1,N),1); % Prior p(x_1)
    A = my_normalize(circshift(eye(N), 3)+0.0001,1); % Transition
        Matrix
    C = my_normalize(eye(N)+rand(N)/100,1); % Observation matrix
    for k=1:K
        for t=1:T
            if t==1
                states(k,t) = randgen(pri);
            else
                states(k,t) = randgen(A(:, states(k,t-1)));
            end
            obs(k,t) = randgen(C(:, states(k,t)));
        end
    end
elseif model == 2
    pri = my_normalize(rand(N,1),1); % Prior p(x_1)
    A = zeros(N^order,N^order);
    for i=1:N^order
        m = my_normalize(rand(1,N),1);
        A(i,mod(i-1,N)*N+1:mod(i-1,N)*N+N) = m;
    end
```

```
    C = my_normalize(rand(N,N^order),1);
    for k=1:K
        for t=1:T
            if t<=order
                states(k,t) = randgen(pri);
            else
                states(k,t) = randgen(A(:,states(k,t-1)));
            end
            obs(k,t) = randgen(C(:, states(k,t)));
        end
    end
elseif model==3
    pri = my_normalize(ones(1,order),1); % Prior ksi
    A = zeros(N,N,order); % Transition Matrix
    for i=1:order
        A(:,:,i) =  my_normalize(eye(N)+randi(order)*0.001,1);
    end
    C = my_normalize(rand(N),1); % Just to return a matrix, see returned values at the top.
    for k=1:K
        for t=1:T
            if t<=order
                obs(k,t) = randgen(ones(1,N));
            else
                temp = zeros(N,1);
                for v=1:order
                    temp = temp + pri(v)*A(:, obs(k,t-v),v);
                end
                obs(k,t) = randgen(temp);
            end
        end
    end
    states = obs;
elseif model==4
    pri = my_normalize(rand(K),1); % Prior ksi
    A = zeros(N,N,K,K); % Transition Matrix
    for i=1:K
        for j=1:K
            A(:,:,i,j) =  my_normalize(eye(N)+randi(100)*0.0001,1);
        end
    end
    C = my_normalize(rand(N),1); % Just to return a matrix, see returned values at the top.
    for t=1:T
```

```
        if t==1
            obs(:,t) = randgen(ones(1,N));
        else
            for i=1:K
                temp = zeros(N,1);
                for j=1:K
                    temp = temp + pri(j,i)*A(:, obs(j,t-1),j,i);
                end
                obs(i,t) = randgen(temp);
            end
        end
    end
    states = obs;
end

function x = randgen(A)
r = rand();
T = cumsum(A)/sum(A);
x = find(T > r, 1, 'first');
if size(x,1)==0, x=randi(size(A,1)), end
```

```
my_sales_test = zeros(7,61,4);
my_return_test = zeros(7,61,4);
real_sales_test = zeros(7,61,4);
real_returns_test = zeros(7,61,4);

%% Parameter initialization and data generation
model = 1; % The model from which data is generated.
K = 7; % Number of sequences
T = 261; % Length of each sequence
N = 40;  % Number of states
order = 1; % How many steps back to consider.
[pri, A, C, states, obs] = generate_data(model,K,T,N,order);
solve_for = [1 0 1 1]; % The model of solution
% 1==>markov-1
% 2==>markov-2
% 3==>mm
% 4==>fm

%% Solutions
figure;
if solve_for(1)
    [lhood_m1,A_r,C_r,alpha] = hmm_markov1(obs(:,1:200),N);
    for day=1:7
        day
```

```
        [~,A,C,alpha] = hmm_markov1(obs(day,1:200),N);
        alpha_pred = squeeze(alpha(:,end,:));
        predictions = zeros(1,61);

        for t = 201:261
            [alpha_pred, pred] = hmm1_next_sale_prediction(A,C,alpha_pred);
            predictions(:,t-200) = pred;

            %updating current state since we already make the observation.
            for i=1:size(alpha_pred,2)
                alpha_pred(:,i) = C(obs(i,t), :)'.*alpha_pred(:,i);
            end

            alpha_pred = my_normalize(alpha_pred,1);
        end
        my_sales_test(day,:,1) = predictions;
        real_sales_test(day,:,1) = obs(day,201:261);
        my_return_test(day,:,1) = my_sales_test(day,:,1)-real_sales_test(day,:,1);
    end

    subplot(4,1,1)
    plot(1:size(lhood_m1,2),lhood_m1)
    title('Markov-1')
    ylabel('likelihood');xlabel('iteration count');
end
if solve_for(2)
    [lhood_m2,A_r,C_r,alpha] = hmm_markov_full(obs,N,order);
    subplot(4,1,2)
    plot(1:size(lhood_m2,2),lhood_m2)
    title('Markov-2')
    ylabel('likelihood');xlabel('iteration count');
end
if solve_for(3)
    order_markov_mm=3;
    [lhood_m3, A, ksi] = hmm_mm(obs(:,1:200),N,order_markov_mm);
    preds = zeros(7,61);
    for t = 201:261
        [~, pred] = mm_next_sale_prediction(A,ksi,obs(:,t-order_markov_mm:t-1));
        preds(:,t-200) = pred;
    end
```

```
    my_sales_test(:,:,3) = preds;
    real_sales_test(:,:,3) = obs(:,201:261);
    my_return_test(:,:,3) = my_sales_test(:,:,3)-real_sales_test
        (:,:,3);

    subplot(4,1,3)
    plot(1:size(lhood_m3,2),lhood_m3)
    title('Mixed Memory Markov')
    ylabel('likelihood');xlabel('iteration count');

end
if solve_for(4)
    [lhood_m4, A, ksi] = hmm_fm(obs(:,1:200),N);
    preds = zeros(7,61);
    for t = 201:261
        [~, pred] = fm_next_sale_prediction(A,ksi,obs(:,t-1));
        preds(:,t-200) = pred;
    end
    my_sales_test(:,:,4) = preds;
    real_sales_test(:,:,4) = obs(:,201:261);
    my_return_test(:,:,4) = my_sales_test(:,:,4)-real_sales_test
        (:,:,4);

    subplot(4,1,4)
    plot(1:size(lhood_m4,2),lhood_m4)
    title('Factorial Markov')
    ylabel('likelihood');xlabel('iteration count');
end
```

```
outlet_count = 1;
ids = zeros(1,outlet_count);
my_sales = zeros(7,61,outlet_count,4);
my_return = zeros(7,61,outlet_count,4);
real_sales = zeros(7,61,outlet_count,4);
real_returns = zeros(7,61,outlet_count,4);

%% Main loop
for outlet_num = 1:outlet_count
    fprintf('Outlet Number: %d\n', outlet_num);
    sale = Sevk - Iade;
    [~,day_c, outlet_c] = size(sale);
    training_c = 200; % this many weeks are used for training.
    solve_for = [0 0 1 0]; % The model of solution

    outlets_full_info = zeros(1,2840);
```

```
    for i=1:2840
        obs = sale(:,:,i);
        outlets_full_info(i) = sum(sum(double(obs==0)));
    end
    outlets_full_info = find(outlets_full_info==0);

    outlet_id = outlets_full_info(randi(size(outlets_full_info,2)));
    ids(outlet_num) = outlet_id;
    obs = sale(:,1:training_c,outlet_id);
    % obs = double(obs==0) + obs; %make all zeros one.
    whole_sale = sale(:,:,outlet_id);
    N = max(whole_sale(:))

    %% Solutions
    figure;
    if solve_for(1)
        for day=1:7
            day
            [lhood_m1,A,C,alpha] = hmm_markov1(obs(day,:),N);
            alpha_pred = squeeze(alpha(:,end,:));
            predictions = zeros(1,261-training_c);

            for t = training_c+1:261
                [alpha_pred, pred] = hmm1_next_sale_prediction(A,C,alpha_pred);
                predictions(:,t-training_c) = pred;

                %updating current state since we already make the observation.
                for i=1:size(alpha_pred,2)
                    alpha_pred(:,i) = C(whole_sale(i,t), :)'.*alpha_pred(:,i);
                end

                alpha_pred = my_normalize(alpha_pred,1);
            end

            my_sales(day,:,outlet_num,1) = predictions;
            real_sales(day,:,outlet_num,1) = sale(day,201:261,outlet_id);
            real_returns(day,:,outlet_num,1) = Iade(day,201:261,outlet_id);
            my_return(day,:,outlet_num,1) = my_sales(day,:,outlet_num,1)-real_sales(day,:,outlet_num,1);
```

```
        end

        diff = my_sales(:,:,outlet_num,1)-real_sales(:,:,outlet_num,1);
        sum(sum(diff.*(double(diff>0))))
        sum(sum(real_returns(:,:,outlet_num,1)))
        sum(sum(diff.*(double(diff<0))))
        sum(sum(my_sales(:,:,outlet_num,1)))

        subplot(4,1,1)
        plot(1:size(lhood_m1,2),lhood_m1)
        title('Markov-1')

    end
    if solve_for(2)
        order_markov_full=2;
        for day=1:7
            day
            [lhood_m2,A,C,alpha] = hmm_markov_full(obs,N,order_markov_full);
            alpha_pred = squeeze(alpha(:,end,:));
            predictions = zeros(1,261-training_c);

            for t = training_c+1:261
                [alpha_pred, pred] = hmm1_next_sale_prediction(A,C,alpha_pred);
                predictions(:,t-training_c) = pred;

                %updating current state since we already make the observation.
                for i=1:size(alpha_pred,2)
                    alpha_pred(:,i) = C(whole_sale(i,t), :)'.*alpha_pred(:,i);
                end

                alpha_pred = my_normalize(alpha_pred,1);
            end

            my_sales(day,:,outlet_num,2) = predictions;
            real_sales(day,:,outlet_num,2) = sale(day,201:261,outlet_id);
            real_returns(day,:,outlet_num,2) = Iade(day,201:261,outlet_id);
            my_return(day,:,outlet_num,2) = my_sales(day,:,outlet_num,2)-real_sales(day,:,outlet_num,2);
```

```

        end

        diff = my_sales(:,:,outlet_num,2)-real_sales(:,:,outlet_num,2);
        sum(sum(diff.*(double(diff>0))));
        sum(sum(real_returns(:,:,outlet_num)));
        sum(sum(diff.*(double(diff<0))));
        sum(sum(my_sales(:,:,outlet_num)));


        subplot(4,1,2)
        plot(1:size(lhood_m2,2),lhood_m2)
        title('Markov-2')

    end
    if solve_for(3)
        order_markov_mm=5;
        [lhood_m3, A, ksi] = hmm_mm(obs(:,1:end),N,order_markov_mm);
        preds = zeros(7,261-training_c);
        for t = training_c+1:261
            [~, pred] = mm_next_sale_prediction(A,ksi,sale(:,t-order_markov_mm:t-1,outlet_id));
            preds(:,t-training_c) = pred;
        end
        my_sales(:,:,outlet_num,3) = preds;
        real_sales(:,:,outlet_num,3) = sale(:,201:261,outlet_id);
        real_returns(:,:,outlet_num,3) = Iade(:,201:261,outlet_id);
        my_return(:,:,outlet_num,3) = my_sales(:,:,outlet_num,3)-real_sales(:,:,outlet_num,3);

        diff = my_sales(:,:,outlet_num,3)-real_sales(:,:,outlet_num,3);
        sum(sum(diff.*(double(diff>0))));
        sum(sum(real_returns(:,:,outlet_num,3)));
        sum(sum(diff.*(double(diff<0))));
        sum(sum(my_sales(:,:,outlet_num,3)));

        subplot(4,1,3)
        plot(1:size(lhood_m3,2),lhood_m3)
        title('Mixed Memory')

    end
```

```
    if solve_for(4)
        [lhood_m4, A, ksi] = hmm_fm(obs(:,1:end),N);
        preds = zeros(7,261-training_c);
        for t = training_c+1:261
            [~, pred] = fm_next_sale_prediction(A,ksi,sale(:,t-1,outlet_id));
            preds(:,t-training_c) = pred;
        end

        my_sales(:,:,outlet_num,4) = preds;
        real_sales(:,:,outlet_num,4) = sale(:,201:261,outlet_id);
        real_returns(:,:,outlet_num,4) = Iade(:,201:261,outlet_id);
        my_return(:,:,outlet_num,4) = my_sales(:,:,outlet_num,4)-real_sales(:,:,outlet_num,4);

        diff = my_sales(:,:,outlet_num,4)-real_sales(:,:,outlet_num,4);
        sum(sum(diff.*(double(diff>0))));
        sum(sum(real_returns(:,:,outlet_num,4)));
        sum(sum(diff.*(double(diff<0))));
        sum(sum(my_sales(:,:,outlet_num,4)));


        subplot(4,1,4)
        plot(1:size(lhood_m4,2),lhood_m4)
        title('Factorial Markov')

    end

    %used for checking if likelihood decreases or not.
    for time = 1:500
        if size(lhood_m1,2)<time && solve_for(1)==1 && lhood_m1(time)>lhood_m1(time+1)
            1
            time
        end
        if size(lhood_m2,2)<time && solve_for(2)==1 && lhood_m2(time)>lhood_m2(time+1)
            2
            time
        end
        if size(lhood_m3,2)<time && solve_for(3)==1 && lhood_m3(time)>lhood_m3(time+1)
            3
```

```
                time
            end
            if size(lhood_m4,2)<time && solve_for(4)==1 && lhood_m4(
                time)>lhood_m4(time+1)
                4
                time
            end
        end

end

%% prints results
diff = my_sales-real_sales;
for i = [1 3 4]
    fprintf('Return by method %d is %d\n',i, sum(sum(sum(diff
        (:,:,:,i).*(double(diff(:,:,:,i)>0)),1),2),3))
    fprintf('Missed newspaper by method %d is %d\n',i, sum(sum(
        sum(diff(:,:,:,i).*(double(diff(:,:,:,i)<0)),1),2),3))
end
fprintf('Real return is this time span is %d\n', sum(sum(sum(
    real_returns,1),2),3));
```

```
function [output] = my_normalize(input,dir)
if dir==1 % Normalizes each column
    output = bsxfun(@rdivide,input,sum(input));
else % Normalizes each row
    output = input./repmat(sum(input,2)',size(input,1),1)';
end
end
```

```
function [likelihood,A,C,alpha] = hmm_markov1(obs,state_count)

%% HMM variables
[seq_count, seq_length] = size(obs);
max_iter_count = 500;
convergence_unit = 1e-2;

%obs = double(obs==0) + obs;

priors = my_normalize(rand(state_count,1),1);

% A is the transition matrix. A(i,j) denotes the probability of
    going from state i to state j
A = my_normalize(rand(state_count,state_count),2);
% C is the observation matrix. C(i,j) denotes the probabillity of
    seeing i at state j.
```

```
C = my_normalize(rand(state_count,state_count),1);


A=zeros(state_count,state_count);
for k=1:seq_count
    for t=2:seq_length
        A(obs(k,t-1),obs(k,t)) = A(obs(k,t-1),obs(k,t)) + 1;
    end
end
A = A + 1; %to get rid of zero probabilities
A = my_normalize(A,2);



%% EM Algorithm
likelihood = [];
for iter_number=1:max_iter_count
    %% E-step:
    iter_number
    alpha = zeros(state_count, seq_length,seq_count);
    beta = zeros(state_count, seq_length,seq_count);
    gamma = zeros(state_count, seq_length,seq_count);
    c_f = zeros(seq_count, seq_length);
    c_b = zeros(seq_count, seq_length);
    for k=1:seq_count
        [alpha(:,:,k), beta(:,:,k), gamma(:,:,k), c_f(k,:) ,c_b(k
            ,:)] = forward_backward(priors,A',C,obs(k,:));
    end
    likelihood = [likelihood -1*sum(sum(log(c_f)))];

    % ksi calculation
    ksi = zeros(state_count, state_count,seq_length-1,seq_count);
    for k=1:seq_count
        for t=1:seq_length-1
            run_sum = 0;
            for i=1:state_count
                for j=1:state_count
                    ksi(i, j, t, k) = alpha(i, t, k)*A(i, j)*C(
                        obs(k,t+1), j)*beta(j,t+1,k);
                    run_sum = run_sum + ksi(i, j,t,k);
                end
            end
            ksi(:, :, t, k) = ksi(:, :, t, k)/run_sum;
        end
    end
```

```
    %% M-step:
    %maximizing priors
    priors = my_normalize(sum(gamma(:,1,:),3),1);
    %maximizing A
    g = repmat(sum(sum(gamma(:,1:seq_length-1,:),3),2)',
        state_count,1)';
    A = my_normalize((sum(sum(ksi,4),3))./g,2);
    %maximizing C
    for j=1:state_count
        for t=1:seq_length
            for k=1:seq_count
                C(obs(k,t),j) = C(obs(k,t),j) + gamma(j, t, k);
            end
        end
        C(:,j) = my_normalize(C(:,j),1);
    end

    if iter_number>1 && likelihood(iter_number)-likelihood(
        iter_number-1)<convergence_unit
        break;
    end
end
end
```

```
function [alpha, beta_postdict, gamma, c_f, c_b] =
    forward_backward(priors,A,C,obs)
% A(i,j) denotes the probability of going from state j to state i

%% initialize variables
K = size(obs,2);
N = size(A,1);
alpha = zeros(N, K);
alpha_predict = zeros(N, K);
beta = zeros(N, K);
beta_postdict = zeros(N, K);

c_f = zeros(K,1);
c_b = zeros(K,1);


%% forward direction
for k=1:K,
    if k==1,
        alpha_predict(:,k) = priors;
```

```
    else
        alpha_predict(:,k) = A*alpha(:, k-1);
    end;
    alpha(:, k) = C(obs(k), :)'.*alpha_predict(:,k);
    c_f(k) = 1/sum(alpha(:,k));
    alpha(:,k) = alpha(:,k)*c_f(k);
end;
%% backward direction
for k=K:-1:1,
    if k==K,
        beta_postdict(:,k) = ones(N,1);
    else
        beta_postdict(:,k) = A'*beta(:, k+1);
    end;
    beta(:, k) = C(obs(k), :)'.*beta_postdict(:,k);
    beta(:,k) = beta(:,k)*(1/sum(beta(:,k)));
    c_b(k) = 1/sum(beta_postdict(:,k));
    beta_postdict(:,k) = beta_postdict(:,k)*c_b(k);
end;

gamma = alpha.*beta_postdict;
gamma = my_normalize(gamma,1);
```

```
function [likelihood,A,C,alpha] = hmm_markov_full(obs,N,past)

%% HMM variables
[seq_count, seq_length] = size(obs);
max_iter_count = 500;
convergence_unit = 1e-2;

% Prior p(x_1)
priors = my_normalize(rand(N^past,1),1);
% A is the transition matrix. A(i,j) denotes the probability of going from state i to state j
A = zeros(N^past,N^past);
for i=1:N^past
    m = my_normalize(rand(1,N),1);
    A(i,mod(i-1,N)*N+1:mod(i-1,N)*N+N) = m;
end
% C is the observation matrix. C(i,j) denotes the probabillity of seeing i at state j.
C = my_normalize(rand(N,N^past),1);

%% EM Algorithm
likelihood = [];
```

```
for iter_number=1:max_iter_count
    %% E-step:
    iter_number
    alpha = zeros(N^past, seq_length,seq_count);
    beta = zeros(N^past, seq_length,seq_count);
    gamma = zeros(N^past, seq_length,seq_count);
    c_f = zeros(seq_count, seq_length);
    c_b = zeros(seq_count, seq_length);
    for k=1:seq_count
        [alpha(:,:,k), beta(:,:,k), gamma(:,:,k), c_f(k,:) ,c_b(k
            ,:)] = forward_backward(priors,A',C,obs(k,:));
    end
    likelihood = [likelihood -1*sum(sum(log(c_f)))];

    % ksi calculation
    ksi = zeros(N^past, N^past,seq_length-1,seq_count);
    for k=1:seq_count
        for t=1:seq_length-1
            run_sum = 0;
            for i=1:N^past
                for j=1:N^past
                    ksi(i, j, t, k) = alpha(i, t, k)*A(i, j)*C(
                        obs(k,t+1), j)*beta(j,t+1,k);
                    run_sum = run_sum + ksi(i, j,t,k);
                end
            end
            ksi(:, :, t, k) = ksi(:, :, t, k)/run_sum;
        end
    end

    %% M-step:
    %maximizing priors
    priors = my_normalize(sum(sum(gamma(:,1,:),2),3),1);
    %maximizing A
    g = repmat(sum(sum(gamma(:,1:seq_length-1,:),3),2)', N^past
        ,1)';
    A = my_normalize((sum(sum(ksi,4),3))./g,2);

    %maximizing C
    for j=1:N^past
        for t=1:seq_length
            for k=1:seq_count
                C(obs(k,t),j) = C(obs(k,t),j) + gamma(j, t, k);
            end
        end
```

```
            C(:,j) = my_normalize(C(:,j),1);
        end

        if iter_number>1 && likelihood(iter_number)-likelihood(iter_number-1)<convergence_unit
            break;
        end
    end
end
```

```
function [likelihood, A, ksi] = hmm_mm(obs,state_count,order)

%% HMM variables
[seq_count, seq_length] = size(obs);
max_iter_count = 500;
convergence_unit = 1e-3;

% ksi is the weight matrix
ksi = my_normalize(rand(order,1),1);
%ksi = ones(order,1)/order;
A = zeros(state_count,state_count,order);

%{
% A is the transition matrix. A(i,j) denotes the probability of going from state i to state j
for i=1:order
    A(:,:,i) = my_normalize(rand(state_count,state_count),2);
end
%}

for v=1:order
    for k=1:seq_count
        for t=order+1:seq_length
            A(obs(k,t-v),obs(k,t),v) = A(obs(k,t-v),obs(k,t),v) + 1;
        end
    end
end
A = A + 1;
for i=1:order
    A(:,:,i) = my_normalize(A(:,:,i),2);
end

%% EM Algorithm
likelihood = [];
```

```
for iter_number=1:max_iter_count

    iter_number
    likelihood = [likelihood calculate_likelihood(obs,A,ksi)];

    %% ksi updates:
    ksi_update = zeros(order,seq_length-order,seq_count);
    for v=1:order
        for k=1:seq_count
            for t=order+1:seq_length
                ksi_update(v,t,k) = ksi(v)*A(obs(k,t-v),obs(k,t),v);
            end
        end
    end
    ksi = sum(sum(ksi_update,2),3)/sum(sum(sum(ksi_update,1),2),3);

    %% Transition Matrix Update
    for i=1:state_count
        for j=1:state_count
            for v=1:order
                run_sum1 = 0;
                run_sum2 = 0;
                for k=1:seq_count
                    for t=order+1:seq_length
                        run_sum1 = run_sum1 + ksi(v)*(obs(k,t-v)==i)*(obs(k,t)==j);
                        run_sum2 = run_sum2 + ksi(v)*(obs(k,t-v)==i);
                    end
                end
                if run_sum2>0
                    A(i,j,v) = run_sum1/run_sum2;
                end
            end
        end
    end
    for v=1:order
        A(:,:,v) = my_normalize(A(:,:,v),2);
    end

    if iter_number>1 && likelihood(iter_number)-likelihood(iter_number-1)<convergence_unit
        break;
```

```
    end
end
end

%% likelihood calculation
function l_hood = calculate_likelihood(obs,A,ksi)
order = size(A,3);
[seq_count, seq_length] = size(obs);
l_hood = 0;
for k=1:seq_count
    for t=order+1:seq_length
        temp = 0;
        for v=1:order
            temp = temp + ksi(v)*A(obs(k,t-v),obs(k,t),v);
        end
        l_hood = l_hood + temp;
    end
end
end
```

```
function [likelihood,A,ksi] = hmm_fm(obs,state_count)

%% HMM variables
[seq_count, seq_length] = size(obs);
max_iter_count = 500;
convergence_unit = 1e-2;

% ksi is the weight matrix
% ksi(i,j)==>how i'th seq one step before affects current j'th seq
ksi = my_normalize(rand(seq_count),1);

% A is the transition matrix. A(i,j,v,u) denotes how v'th sequence
% being i one step ago affects u'th sequence being j.
A = zeros(state_count,state_count,seq_count,seq_count);
for i=1:seq_count
    for j=1:seq_count
        A(:,:,i,j) = my_normalize(rand(state_count),2);
    end
end

%% EM Algorithm
likelihood = [];
for iter_number=1:max_iter_count
```

```
    iter_number
    likelihood = [likelihood calculate_likelihood(obs,A,ksi)];

    %% ksi updates:
    ksi_update = zeros(seq_count);
    for u=1:seq_count %affected
        temp_ksi = zeros(1,seq_count);
        for v=1:seq_count %affecting
            temp = 0;
            for t=1:seq_length-1
                temp = temp + ksi(v,u)*A(obs(v,t),obs(u,t+1),v,u);
            end
            temp_ksi(v) = temp;
        end
        ksi_update(:,u) = temp_ksi;
    end
    ksi = my_normalize(ksi_update,1);

    %% Transition Matrix Update
    for i=1:state_count
        for j=1:state_count
            for u=1:seq_count %affected
                for v=1:seq_count %affecting
                    run_sum1 = 0;
                    run_sum2 = 0;
                    for t=1:seq_length-1
                        run_sum1 = run_sum1 + ksi(v,u)*(obs(v,t)==i)*(obs(u,t+1)==j);
                        run_sum2 = run_sum2 + ksi(v,u)*(obs(v,t)==i);
                    end
                    if run_sum2>0
                        A(i,j,v,u) = run_sum1/run_sum2;
                    end
                end
            end
        end
    end
    for i=1:seq_count
        for j=1:seq_count
            A(:,:,i,j) = my_normalize(A(:,:,i,j),1);
        end
    end
```

```

    if iter_number>1 && likelihood(iter_number)-likelihood(iter_number-1)<convergence_unit
        break;
    end
end
end

%% likelihood calculation
function l_hood = calculate_likelihood(obs,A,ksi)
[seq_count, seq_length] = size(obs);
l_hood = 0;
for t=1:seq_length-1
    temp = 0;
    for u=1:seq_count %current series
        for v=1:seq_count %a step before
            temp = temp + ksi(v,u)*A(obs(v,t),obs(u,t+1),v,u); %v affecting u
        end
    end
    l_hood = l_hood + temp;
end
end
```

```
function [alpha_pred, pred] = hmm1_next_sale_prediction(A,C,alpha_g)
seq_count = size(alpha_g,2);
alpha = zeros(size(A,1),seq_count);
alpha_pred = zeros(size(A,1),seq_count);
pred = zeros(seq_count,1);
for i=1:seq_count
    alpha_pred(:,i) = A'*alpha_g(:,i);
    alpha(:,i) = C*alpha_pred(:,i);
    [~,id] = max(alpha(:,i));
    pred(i) = id;
end
end
```

```
function [pred_dis, pred] = mm_next_sale_prediction(A,ksi,obs)
order = size(A,3);
seq_count = size(obs,1);
pred_dis = zeros(seq_count,size(A,1));
pred = zeros(seq_count,1);
for i=1:seq_count
    pr = zeros(1,size(A,1));
    for v=1:order
```

```
            pr = pr + ksi(v)*A(obs(i,end-v+1),:,v);
        end
        [~,id] = max(pr);
        pred(i) = id;
        pred_dis(i,:) = pr;
    end
end
```

```
function [pred_dis, pred] = fm_next_sale_prediction(A,ksi,obs)
seq_count = size(obs,1);
pred = zeros(seq_count,1);
pred_dis = zeros(seq_count,size(A,1));
for i=1:seq_count
    pr = zeros(1,size(A,1));
    for j=1:seq_count
        pr = pr + ksi(j,i)*A(obs(j),:,j,i);
    end
    [~,id] = max(pr);
    pred(i) = id;
    pred_dis(i,:) = pr;
end
end
```

References

Lawrence K. Saul and Michael I. Jordan. Mixed memory markov models: decomposing complex stochastic processes as mixtures of simpler ones, 1998.