CH5540 Multivariate Data Analysis - Assignment 5

AE14B050

May 11, 2018

Solution 1:

(a) KPCR: We aplly the basic KPCR algorithm as follows ,

- Create a kernel on the autoscaled training data set of 70 samples to create a 70x70 Kernel.
- Use this kernel on the test data set, calculate PRESS to find the optimum number of principle components.
- Use all the found parameters to create the final regression fit. Use this in (b) to calculate the predictions as required.

(a) Predictions: The parameter used for the KPCR are

principle components = $p = 20$

Kernel width = $\sigma = 0.1406$

Temperature(deg C)	Antoine Prediction(kPa)	KPCR prediction(kPa)	Error(%)
55	63.9645	64.5222	0.8719
100	246.2437	49.2515	79.9989

This drastic difference arise because of the core kernel method itself. KPCR uses existing data to predict values for the new data by comparing how close they are. Here our data is concentrated between $10° - 70°C$, we hence get a very good estimation of the pressure at $55°C$ while a horrendous estimate at $100°C$.

Listing 1: KPCR algo

```
clc
clear all
load vpdata

%% Scaling Data with training set
train = 70;
test = size(temp,1)-train;
```

```
P = psat(1:train);
T = temp(1:train);
meanP = mean(P);
meanT = mean(T);
stdP = std(P);
stdT = std(T);
P = (P-meanP)/stdP;
T = (T-meanT)/stdT;

%% Kernel params - training data kernel
minD = zeros(train,1);
distM = zeros(train);
for i=1:train
    dist = bsxfun(@minus,T,T(i,:));
    dist = sum(dist.^2,2);
    distM(:,i)=dist;
    dist = sqrt(dist);
    dist(i)=[];
    minD(i)=min(dist);
end
sig = 5*mean(minD);%kernel width or sigma as a function of
%the mean minimum nearest point distance of each point
sig2 = 2*sig*sig; % sigma^2

%% Estimate number of principle components and width using PRESS
%Gaussian Kernel matrix
K = exp(-distM/sig2);
I = ones(size(K))/train;
[u s v] = svd(K,'econ');
PRESS = zeros(train,1);
for i=1:train
    proj = K*v(:,1:i);
    A = pinv(proj)*P; %OLS trained A

    testT = temp(train+1:end);%test data
    testT = (testT-meanT)/stdT;%scaling test data
    %% Compute Kernel, principle component projections for test data for
       cross validation
    testk = zeros(test,train);
    for j=1:test
        testk(j,:) = (T - testT(j));
    end
    testk = exp(-(testk.^2)/sig2);
    proj = testk*v(:,1:i);

    %% Predict pressure using the trained A and the principle components
    pred = (proj*A);
    %% PRESS error
    testP = psat(train+1:end);
    testP = (testP-meanT)/stdT;
    PRESS(i) = sum((pred-testP).^2);
end

%% Find Id of least PRESS value
PRESS(PRESS>1e2)=100;%correct for very large values, makes finding minimum
   faster
[val id] = min(PRESS);
p= id; %we get the optimum number of principle components
%% Predict pressures for p principle components - same as in the forloop
   above, better to compute this again than store all those above predicted
   pressure values
proj = K*v(:,1:p);
A = pinv(proj)*P; %OLS trained A

testT = temp(train+1:end);%test data
testT = (testT-meanT)/stdT;%scaling test data
%% Compute Kernel, principle component projections for test data for cross
   validation
testk = zeros(test,train);
for j=1:test
    testk(j,:) = (T - testT(j));
end
```

```
testk = exp(-(testk.^2)/sig2);
proj = testk*v(:,1:p);

%% Predict pressure using the trained A and the principle components
pred = (proj*A);

%% PRESS error
testP = psat(train+1:end);
testP = (testP-meanT)/stdT;
PRESS = sum((pred-testP).^2);

%% Predicted pressure scaled back, and RMSE
predict = pred*stdP+meanP;
RMSE = sqrt(mean((predict-psat(train+1:end)).^2));

%% Special cases for (b)
testT = 100;%set value
predictA = exp(14.0568-(2825.42/(testT+230.44)));%antoine equation
    prediction
testT = (testT-meanT)/stdT;
testk = ((T-testT).^2);
testk = exp(-testk/sig2);
proj = testk.'*v(:,1:p);
predict1 = proj*A;
predict1 = predict1*stdP + meanP;%predicted solution scaled back
error = 100*abs(predictA-predict1)/predictA;
```

Solution 2:

(a) OLS: We use OLS directly to obtain our regression fit.

(b) TLS: We use the basic last eigenvector of PCA to obtain our regression fit.

	a_1	b_1
OLS	0.4833	0.7378
TLS	0.4949	0.7523

(c) (i) p=1: Let us denote the stacking order by m.

If $m = 1$, then in the stacking vector $[y_k \quad y_{k-1} \quad u_{k-1}]$ there exists only 1 relation between all the variables, which is given by the standard equation of $p = 1$,

$$y_k = a_1 y_{k-1} + b_1 u_{k-1}$$

Hence constraints $= c = 1$.

This inreases simultaneously for each increase in m . For stacking m,

$$y_k = a_1 y_{k-1} + b_1 u_{k-1}$$

$$y_{k-1} = a_1 y_{k-2} + b_1 u_{k-2}$$

till we reach,

$$y_{k-m+1} = a_1 y_{k-m} + b_1 u_{k-m}$$

The number of contraints between all the variables will hence be be $c = m$

(ii) Any true order: Extending the above for any true order $= p$, we can rewrite the stacked vector relations as,

$$y_k = a_1 y_{k-1} + a_2 y_{k-2} + ... + a_p y_{k-p} + b_1 u_{k-1} + b_2 u_{k-2} + ... + b_p u_{k-p}$$

$$y_{k-1} = a_1 y_{k-2} + a_2 y_{k-3} + ... + a_p y_{k-p} + b_1 u_{k-2} + b_2 u_{k-3} + ... + b_p u_{k-p-1}$$

till we reach,

$$y_{k-m+p} = a_1 y_{k+p-m-1} + a_2 y_{k+p-m-2} + ... + a_p y_{k-m} + b_1 u_{k+p-m-1} + b_2 u_{k+p-m-2} + ... + b_p u_{k-m}$$

The number of contraints between all the variables will hence be,

$$c = m - p + 1$$

(d) PCA on m=10: We obtain the following eigenvalues for this during the svd.

1-10	3.2165	2.7518	2.3259	1.9089	1.4061	1.3325	1.2379	1.2125	1.1496	1.1252
11-20	0.7892	0.0233	0.0227	0.0220	0.0207	0.0187	0.0182	0.0182	0.0156	0.0108
21	0.0105									

It is very easy to observe the last 10 eigen values are insignificant which mean there exists 10 constraint equations in the given stacking vector. From (c) we have the relation, $constraints = m - p + 1$.

We input the $m = 10$, we found $c = constraints = 10$. Using the realtion we get $p = 1$. Which is the true stacking order. Using $m = p = 1$. We do the PCA again to obtain the regression fit.

	a_1	b_1
m=1	0.4949	0.7523

This is nothing but the TLS fit.

(e) IPCA with error variance estimation: The IPCA method consists of two steps. In the first step, we perform an iterative PCA on the matrix of lagged measurements for some L and determine the process order η from the identified number of constraints. This step of determining exact process order constitutes a key novel contribution of this work. An important by-product of the first step is the estimate of noise covariance matrix. Subsequently, in the second step we reconfigure the data matrix as per the determined order and scale it with $\Sigma_e^{-1/2}$. A PCA of this scaled matrix is used to obtain the desired regression model. Estimation of delay is implicit in the model identification step.

Listing 2: OLS TLS

```
clc
clear all
load arx

U = umeas.';
Y = ymeas.';
YU = [Y , U];
YU = YU(1:end-1,:);%1st order
%% OLS
A = pinv(YU)*Y(2:end);
pred = YU*A;
RMSE = sqrt(sum((pred - Y(2:end)).^2)/size(YU,1));
%% TLS
YYU = create_stack(Y,U,1);%[Y(2:end),YU];
[u s v] = svd(cov(YYU));
rel = v(:,end);
```

```
At = -rel(2:end)/rel(1);
predt = YU*At;
RMSEt = sqrt(sum((predt - Y(2:end)).^2)/size(YU,1));
```

Listing 3: PCA for m=10

```
clc
clear all
load arx

U = umeas.';
Y = ymeas.';
%% Stacking order 10
YU10 = create_stack(Y,U,10);
[u s v] = svd(cov(YU10));
meanA = mean(v(:,end-8:end)');
rel10 = v(:,end);
A10 = -rel10(2:end)/rel10(1);
pred10 = YU10(:,2:end)*A10;
RMSE10 = sqrt(sum((pred10 - Y(11:end)).^2)/size(YU10,1));

%% Stacking order 1 >> estimated from s in previous step (significant s = 11
    => constraints = 10 => p = 10+1-(m=10)=1)
% YU5 = create_stack(Y,U,1);
% [u s v] = svd(cov(YU5));
% rel5 = v(:,end);
% A5 = -rel5(2:end)/rel5(1);
% pred5 = YU5(:,2:end)*A5;
% RMSE5 = sqrt(sum((pred5 - Y(2:end)).^2)/size(YU5,1));
```

Solution 3:

Applying sparse PCA we obtain a constraint matrix F. The columnwise accuracies of correctly predicted non-zero values are,

1-10	0.0000	1.8868	0.0000	0.0000	0.0000	10.9091	22.3404	0.0000	0.0000	0.0000
11-20	0.0000	8.3333	19.5652	12.1212	0.0000	17.5676	0.0000	0.0000	0.0000	0.0000
21-30	0.0000	0.0000	0.0000	0.0000	16.2162	0.0000	17.6471	7.1429	30.0971	22.4138
31-33	18.8235	0.0000	0.0000							

Acccuracies in %.

Listing 4: SPCA for 3

```
clc
clear all
load yeastdata

X = microarraydata.';
card = sum(Astruct).';
num_comp = size(card,1);

F = sparsePCA(X, card, num_comp,10);

Fcomp = F;
Fcomp(Fcomp~=0)=1;
Fcomp = Fcomp&Astruct;
Fs = sum(Fcomp);
ac = 100 - (100*(card-Fs.')./card);
```

Solution 4:

From the given mixtures we use ICA to compute the independent components. These are then correlated to the true audio signals and the correlation factors are given below.

Est↓\True→	Kennedy Speech	No Ways
Kennedy Speech	-0.9753	-0.0026
No Ways	-0.0241	-0.9907

From the correlations it is eveident we obtain the true signals very accurately. Let us visualize this for brevity.

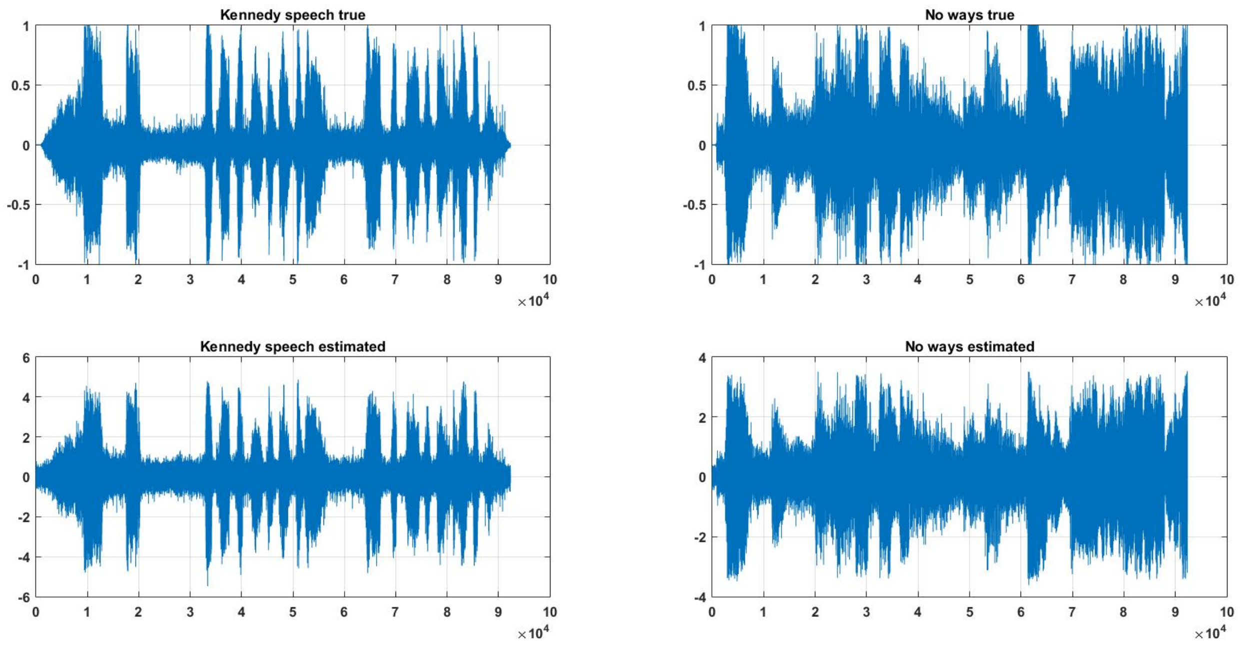

Figure 1: Estimated and True Signal comparision

Listing 5: ICA for 4

```
clc

clear all

load audiomixture

[ken,Fken] = audioread('kennedy_ask_not.wav');

[no,Fno] = audioread('no_ways_tired.wav');


%% Find the independent components

sno = size(no);

sken = size(ken);

A = zeros(5,2); %corrected for random state

for i=1:5

    for j=1:2

        A(i,j) = ((-1)^(j+1))*i/10;

    end

end

[IndS, MixM, SepM] = fastica(Zmeas,'numOfIC',2,'approach','symm','
   stabilization','on','initGuess',A);


%% Find the id at which ken is correlated maximum to the estimated signal

core = zeros(100,1);
```

```
for i =1:1000
    dum = ken(i:min(i+sno-1,sken));
    core(i) = corr(IndS(1,1:min(sno,size(dum))).',dum);
end
[val id] = max(abs(core));

%% Change data accordingly and find the correlation coefficient
ken = ken(id:end);
IndS = IndS(:,1:sken-id+1);
no = no(1:sken-id+1);
core2 = corr([ken,no],IndS.');
[M,maxid]=max(abs(core2));

%% Plot figures of the the signals
figure
subplot(2,2,1)
plot(ken)
title('Kennedy speech true')
subplot(2,2,3)
plot(IndS(maxid(1),:))
title('Kennedy speech estimated')
subplot(2,2,2)
plot(no)
title('No ways true')
subplot(2,2,4)
plot(IndS(maxid(2),:))
title('No ways estimated')
```