APPENDIX-A
A.1 PROGRAM FOR DOWN SAMPLE EEG DATA

function g = downsample(matrix,f);
% program to downsample the data by factor f
% PRE: matrix each row a data vector
% Post: returns a matrix A, where the rows are downsamples by factor f
% right now it just selects each fth value
  g=matrix(:,1:f:length(matrix));

A.2 PROGRAM FOR DETECTION OF K-COMPLEX WAVE IN EEG SIGNAL

% K-complex detection in EEG signals
% close all
% clear all
% Fetching data...
load lindy1a.mat
% lets fetch the training set...
load Kcomplxes1.mat
load Kcomplxes2.mat
% and process it a bit...
TrainingSet = [
  ee263Smooth(Kcomplex1([1:10:length(Kcomplex1)]));zeros(150,1)];
  ee263Smooth(Kcomplex2([1:10:length(Kcomplex2)]));zeros(200,1)];
  ee263Smooth(Kcomplex3([1:10:3010]))';
  ee263Smooth(Kcomplex4([1:10:3010]))';
  ee263Smooth(Kcomplex5([1:10:3010]))' ];
TrainingSet = TrainingSet -
(ones(size(TrainingSet,2),1)*mean(TrainingSet'));
f = Channel17;

% global variables
% number of samples in a block
global SizeOfBlock;
SizeOfBlock = 500;

global DistanceBtwBlocks;
DistanceBtwBlocks = 250;

global DistBtwSmpls;
DistBtwSmpls = 10;

global MinDiversity;
MinDiversity = 40;

% Lets downsample
f = f([1:DistBtwSmpls:length(f)]);

% Lets sliceem UP!
SlizedData = BlockSlizer(f,SizeOfBlock,DistanceBtwBlocks);
Indexies = BlockSlizer([1:length(f)],SizeOfBlock,DistanceBtwBlocks);
PostFiltered = [];
SurvivingBlocks = [];
IndexiesSurvive = [];

% Lets Filter the slized data!, and pick our stuff
% for(k = 1:size(SlizedData,1))
% CurrBlck = (ee263Smooth(SlizedData(k,:')));
% if(max(CurrBlck)-min(CurrBlck) > MinDiversity )
% PostFiltered = [PostFiltered;CurrBlck];
% SurvivingBlocks = [SurvivingBlocks;k];
IndexiesSurvive = [IndexiesSurvive;([Indexies(k,1)
Indexies(k,size(Indexies,2)) - 1).*10 + 1)];
% end
% k/size(SlizedData,1)
%end
Allws = [];
% Lets now enter the main loop...
for(k = 1:size(SlizedData,1))
    CurrBlck = SlizedData(k,:);
    % Now we use the feature extractor...
    W = RubberMask(CurrBlck,TrainingSet);
    Allws = [Allws W];
end
% FE (feature extraction) functions
% 1. LPC
% 2. GSM
% 3. rubbermask
% 4. template matching
% CLF function
% postprocessing
plot(sum(Allws)')
figure
Kul = find(sum(Allws) > 9E5);
for(i=1:length(Kul))
    [Indexies(Kul,1) (Indexies(Kul,1)+SizeOfBlock)]
end
A.3 PROGRAM TO PLOT WEIGHTS OF K-COMPLEXES

% K-complex detection in EEG signals
% Plots weights of 3 K-complexes both channels
% FE (feature extraction) functions
% LPC (linear predictive coding)

L=10; % weights
l=1000; % assumed length of complex l samples / sec

for i=1:6
    switch i
        case {1}
            complex=K1(1,1:1);
        % complex=sigolayfilt(complex,3,29)';
        % figure
        % plot(complex)
        case {2}
            complex=K1(2,1:1);
        case {3}
            complex=K2(1,1:1);
        case {4}
            complex=K2(2,1:1);
        case {5}
            complex=K3(1,1:1);
        case {6}
            complex=K3(2,1:1);
    end

d_k=complex(1:l);
X_k=toeplitz(zeros(1,L),complex);
mu = 0.0000001;
error=[];
W_k=zeros(L,1); % weight vector with 12 weights
Weights=[];
for k=1:l % iteration process
    e_k=d_k(k)-W_k'*X_k(:,k);
    error=[error e_k^2];
    W_k=W_k +2*mu*e_k*X_k(:,k);
    Weights=[Weights W_k];
end
t=linspace(1,1,1);
subplot(3,2,i)
plot(t,Weights);
W_k

A.4 PROGRAM FOR K-COMPLEX DETECTION TRAINING ALGORITHM
% K-complex detection in EEG signals
% Training Algorithm
clear all;
close all;
% preprocessing
T=1; % assumed length of complex in sec (use only integer)
ds=10; % factor for downsampling
l=T*1000/ds; % length of cut
t=linspace(1,1,1); % vector for plots
% FE (feature extraction) function: LPC
L=25; % number of weights
a=size(K);
for i=1:10 % a(1) % run loop for all k-complexes
    complex=K(i,1:1,1);
    % complex=offset(complex);
    figure(i)
    subplot(2,2,1)
    plot(complex); title('original');
    d_k=complex(1:1);
    X_k=toeplitz(zeros(1,L),complex);
    mu = 0.0000001;
    error=[];
    y_k=[];
    W_k=zeros(L,1); % weight vector with 12 weights
    Weights=[];
    for k=1:L % iteration process
        y=W_k'*X_k(:,k);
        y_k=[y_k y]; % predicted output
        e_k=d_k(k)-y_k(k);
        error=[error e_k^2];
        W_k=W_k+2*mu*e_k*X_k(:,k);
        Weights=[Weights W_k];
    end
    subplot(2,2,3); plot(y_k); title('predicted signal');
    subplot(2,2,2); plot(error);title('error');
A.5 PROGRAM TO DETECT NON K-COMPLEX AND
DOWNSAMPLING OF EEG DATA FILES.

function [a,b,c] = lindyla(T,ds);
% function cuts all k-complexes from lindyla
% T is length of cut in sec; ds is factor for downsampling
% returns 3 dimensinal matrices
% a=k-complex matrix
% b=non-k-comlex matrix after stimulus
% c=non-complex

load lindyla;
L=T/ds; % length of k-complex cut (signal is downsampled)
l=T*1000/ds; % length of cut

% generate Matrix with two rows
% 1st row Channel 17, 2nd row Channel 18
lindy=[Channel17'; Channel18'];
lindy=downsample(lindy,ds);

% filter lindy with Savitzky-Golay Filter
lindy=sgolayfilt(lindy',3,9)';

% cut out all k-complexes each 1 samples = 1 sec long
st=round(2168/ds); % number 1
K1=lindy(:,st:st+1);
st=round(66561/ds); % number 4
K4=lindy(:,st:st+1);
st=round(157245/ds);  % number 8
K8=lindy(:,st:st+1);

st=round(174409/ds);  % number 9
K9=lindy(:,st:st+1);

st=round(200572/ds);  % number 10
K10=lindy(:,st:st+1);

st=round(229941/ds);  % number 11
K11=lindy(:,st:st+1);

st=round(255167/ds);  % number 12
K12=lindy(:,st:st+1);

st=round(280635/ds);  % number 13
K13=lindy(:,st:st+1);

st=round(306719/ds);  % number 14
K14=lindy(:,st:st+1);

st=round(335092/ds);  % number 15
K15=lindy(:,st:st+1);

st=round(357186/ds);  % number 16
K16=lindy(:,st:st+1);

st=round(379855/ds);  % number 17
K17=lindy(:,st:st+1);

st=round(407592/ds);  % number 18
K18=lindy(:,st:st+1);

st=round(431229/ds);  % number 19
K19=lindy(:,st:st+1);

st=round(451899/ds);  % number 20
K20=lindy(:,st:st+1);

st=round(480650/ds);  % number 21
K21 = lindy(:, st:st+1);
st = round(499961/ds);  % number 22
K22 = lindy(:, st:st+1);
st = round(518625/ds);  % number 23
K23 = lindy(:, st:st+1);
st = round(542343/ds);  % number 24
K24 = lindy(:, st:st+1);
st = round(560444/ds);  % number 25
K25 = lindy(:, st:st+1);
st = round(577600/ds);  % number 26
K26 = lindy(:, st:st+1);
st = round(595974/ds);  % number 27
K27 = lindy(:, st:st+1);

% generate matrix with all k-complexes
Ka = [K1(1,:); K4(1,:); K8(1,:); K9(1,:); K10(1,:); K11(1,:); K12(1,:); K13(1,:); K14(1,:); K15(1,:); K16(1,:); K17(1,:); K18(1,:); K19(1,:); K20(1,:); K21(1,:); K22(1,:); K23(1,:); K24(1,:); K25(1,:); K26(1,:); K27(1,:)];
 Kb = [K1(2,:); K4(2,:); K8(2,:); K9(2,:); K10(2,:); K11(2,:); K12(2,:); K13(2,:); K14(2,:); K15(2,:); K16(2,:); K17(2,:); K18(2,:); K19(2,:); K20(2,:); K21(2,:); K22(2,:); K23(2,:); K24(2,:); K25(2,:); K26(2,:); K27(2,:)];
K = cat(3, Ka, Kb);  % 3 dimensional array

% cut out all no k-complexes after stimulus each L samples = 1 sec long
st = round(21661/ds);  % number 2
N2 = lindy(:, st:st+1);
st = round(38807/ds);  % number 3
N3 = lindy(:, st:st+1);
st = round(84389/ds);  % number 5
\texttt{N5=lindy(:,st:st+1);}
\texttt{st=round(110995/ds); \hspace{1cm} \% number 6}
\texttt{N6=lindy(:,st:st+1);}
\texttt{st=round(137667/ds); \hspace{1cm} \% number 7}
\texttt{N7=lindy(:,st:st+1);}
\texttt{Na=[N2(1,:);N3(1,:);N5(1,:);N6(1,:);N7(1,:)];}
\texttt{Nb=[N2(2,:);N3(2,:);N5(2,:);N6(2,:);N7(2,:)];}
\texttt{N=cat(3,Na,Nb); \% 3 dimensional array}
\texttt{\% cut out all some no k-complexes without stimulus each 1 samples = 1 sec long}
\texttt{\% cut D samples before stimulus}
\texttt{D=400;}
\texttt{st=round(407592/ds)-D; \hspace{1cm} \% number 18}
\texttt{S18=lindy(:,st:st+1);}
\texttt{st=round(431229/ds)-D; \hspace{1cm} \% number 19}
\texttt{S19=lindy(:,st:st+1);}
\texttt{st=round(451899/ds)-D; \hspace{1cm} \% number 20}
\texttt{S20=lindy(:,st:st+1);}
\texttt{st=round(480650/ds)-D; \hspace{1cm} \% number 21}
\texttt{S21=lindy(:,st:st+1);}
\texttt{st=round(499961/ds)-D; \hspace{1cm} \% number 22}
\texttt{S22=lindy(:,st:st+1);}
\texttt{st=round(518625/ds)-D; \hspace{1cm} \% number 23}
\texttt{S23=lindy(:,st:st+1);}
\texttt{st=round(542343/ds)-D; \hspace{1cm} \% number 24}
\texttt{Sa=[S18(1,:);S19(1,:);S20(1,:);S21(1,:);S22(1,:);S23(1,:)];}
\texttt{Sb=[S18(2,:);S19(2,:);S20(2,:);S21(2,:);S22(2,:);S23(2,:)];}
S=cat(3,Sa,Sb); % 3 dimensional array
a=K;
b=N;
c=S;

A.6 DETECTION AND WEIGHTS OF NON K-COMPLEXES
% K-complex detection in EEG signals
% Plots weights of 3 Non-K-Complexes both channels
% FE (feature extraction) functions
% LPC (linear predictive coding)
L=10; % weights
l=1000; % assumed length of complex l samples / sec
for i=1:6
    switch i
    case {1}
        complex=N1(1,1:l);
        % complex=sgolayfilt(complex,3,29);
        % figure
        % plot(complex)
    case {2}
        complex=N1(2,1:l);
    case {3}
        complex=N2(1,1:l);
    case {4}
        complex=N2(2,1:l);
    case {5}
        complex=N3(1,1:l);
case {6}
    complex=N3(2,1:1);
end
d_k=complex(1:1);
X_k=toeplitz(zeros(1,L),complex);
mu = 0.0000001;
error=[];
W_k=zeros(L,1); % weight vector with 12 weights
Weights=[];
for k=1:1 % iteration process
    e_k=d_k(k)-W_k'*X_k(:,k);
    error=[error e_k^2];
    W_k=W_k +2*mu*e_k*X_k(:,k);
    Weights=[Weights W_k];
end
t=linspace(1,1,1);
subplot(3,2,i)
plot(t,Weights);
W_k
end

A.7 TO ELIMINATE DC OFFSET FROM COMPLEX
function g = offset(complex);
% eliminates dc-offset from complex
% eliminate dc-offset
mean1=mean(complex(1,:));
%mean2=mean(complex(2,:));
A.8 PROGRAM FOR TRAINING SET FOR KNOWN ARTIFACTS

% Function to do rubber mask fitting,
% Usage
% W = RubberMask(CurrBlck,TrainingSet);
% Training Set which is a matrix with known artifacts
% The ith row of TrainingSet contains the ith artifact
% The TrainingSet needs of course to have its artifacts
% all of the same length, but you can pad up with zeros to meet that
% Post: w(i) contains the xcor of CurrBlck to the ith artifact in
% Training set...
% returns zero if TrSet is empty...

function f = RubberMask(Block,TrSet);

w = [];

% due to matlabs xcorr student version a minor preprocessing is Needed
RevBlock = Block(length(Block):-1:1) - mean(Block);

for(k = [1:size(TrSet,1)])
    Xcorrel = conv(RevBlock,TrSet(k,:));
    w = [w;max(Xcorrel)];
end

if(size(TrSet,1) == 0)
f = 0;
else
    f = w;
end
A.9 PROGRAM FOR SCALING TEST K-COMPLEX

% K-complex detection in EEG signals
% Scaling test of theoretical k-complex

clear all;
close all;

% generation of k-complex
l=100; % assumed length of complex 1 samples / sec
k=linspace(1,100,100);
complex=zeros(1,l);
complex(21:30)=0.8*k(1:10);
complex(31:50)=complex(30)-0.6*k(1:20);
complex(51:70)=complex(50)+0.2*k(1:20);

% generate scaled complex
complex2=zeros(1,l);
complex2(21:30)=k(1:10);
complex2(31:50)=complex2(30)-0.8*k(1:20);
complex2(51:70)=complex2(50)+0.3*k(1:20);

plot(k,complex,k,complex2)

% FE (feature extraction) functions
% LPC (linear predictive coding)

L=10; % weights
d_k=complex;
X_k=toeplitz(zeros(1,L),complex);
mu = 0.0005; % select mu 0.005 instabil
error=[];
W_k=zeros(L,1); % weight vector with 12 weights
Weights=[];
for k=1:1 % iteration process
    e_k = d_k(k) - W_k'*X_k(:,k);
    error = [error e_k^2];
    W_k = W_k + 2*mu*e_k*X_k(:,k);
    Weights = [Weights W_k];
end

t = linspace(1,1,1);
figure
subplot(2,1,1)
plot(t,Weights);
title('Weights')
 subplot(2,1,2)
d_k = complex2;
X_k = toeplitz(zeros(1,L),complex);
u = 0.0005; % select mu 0.005 instabil
error = []; % weight vector with 12 weights
Weights = [];
for k=1:1 % iteration process
    e_k = d_k(k) - W_k'*X_k(:,k);
    error = [error e_k^2];
    W_k = W_k + 2*mu*e_k*X_k(:,k);
    Weights = [Weights W_k];
end
plot(t,Weights)
figure
plot(t,error(1,:))
A.10 PROGRAM TO DETECT K-COMPLEX SAVITZKY-GOLAY TEST

% K-complex detection in EEG signals
% Savitzky-Golay Filter test

clear all;
close all;
load Kcomplex1;
a=sgolayfilt(Kcomplex1,3,9);
b=sgolayfilt(Kcomplex1,3,15);
c=sgolayfilt(Kcomplex1,3,29);

subplot(2,2,1)
plot(Kcomplex1)
title('original')

subplot(2,2,2)
plot(a)
title('Savitzky-Golay Filter 1')

subplot(2,2,3)
plot(b)
title('Savitzky-Golay Filter 2')

subplot(2,2,4)
plot(c)
title('Savitzky-Golay Filter 3')
A.11 PROGRAM FOR SHIFTING TEST OF K-COMPLEX

% K-complex detection in EEG signals
% Shifting test of theoretical k-complex

clear all;
close all;
% global variables
% generation of k-complex
l=100;  % assumed length of complex 1 samples / sec
k=linspace(1,100,100);
complex=zeros(1,l);
complex(21:30)=0.8*k(1:10);
complex(31:50)=complex(30)-0.6*k(1:20);
complex(51:70)=complex(50)+0.2*k(1:20);
% generate shifted complex
complex2=[zeros(1,10) complex];
complex2=complex2(1:100);
plot(k,complex,k,complex2)
% FE (feature extraction) functions
% LPC (linear predictive coding)
L=10;  % weights
d_k=complex;
X_k=toeplitz(zeros(1,L),complex);
mu = 0.0005;  % select mu 0.005 instabil
error=[];
W_k=zeros(L,1);  % weight vector with 12 weights
Weights=[];
for k=1:l  % iteration process
\[ e_k = d_k(k) - W_k'X_k(:,k); \]
\[ \text{error} = [\text{error} e_k^2]; \]
\[ W_k = W_k + 2\mu e_k X_k(:,k); \]
\[ \text{Weights} = [\text{Weights} W_k]; \]

end

t = linspace(1,1,l);
figure
subplot(2,1,1)
plot(t,Weights);
title('Weights')
subplot(2,1,2)
d_k = complex2;
X_k = toeplitz(zeros(1,L),complex2);
mu = 0.0005; % select mu 0.005 instable
error = []; % weight vector with 12 weights
Weights = [];
for k = 1:l % iteration process
    \[ e_k = d_k(k) - W_k'X_k(:,k); \]
    \[ \text{error} = [\text{error} e_k^2]; \]
    \[ W_k = W_k + 2\mu e_k X_k(:,k); \]
    \[ \text{Weights} = [\text{Weights} W_k]; \]
end
plot(t,Weights)
figure
plot(t,error(1,:))
title('Error')


% CLF function
% postprocessing

A.12 PROGRAM TO TRAIN CLASSIFIER

function [f,minima]=TrainClassifier(Good,Bad)
% usage:
% [p,minima]=TrainClassifier(Good,Bad)
% function to train a classifier on the form
% x'p, where x is the input and p is an upsummer
% found by LP.
% work well.

Aineq = [];
for(i = 1:size(Good,1))
for(j = 1:size(Bad,1))
    Aineq = [Aineq; 1 -(Good(i,:) - Bad(j,:))];
end
end
Aineq = [Aineq;zeros(size(Good,2),1) -eye(size(Good,2))];
bineq = zeros(size(Aineq,1),1);
Aeq = [0 ones(1,size(Good,2))];
beq = [1];
c = [-1;zeros(size(Good,2),1)];
sol = linprog(c,Aineq,bineq,Aeq,beq);
minima = sol(1);
f = sol(2:length(sol))';
A.13 DETECTION OF K-COMPLEX FOR TRAINING ALGORITHMS TO ADJUST WEIGHTS

% K-complex detection in EEG signals
% Training Algorithm
% feeds 3 different k-complexes and adjusts weights
% FE (feature extraction) functions
% LPC (linear predictive coding)

L=10; % weights
l=1000; % assumed length of complex l samples / sec
mu = 0.0000001;
error=[];
W_k=zeros(L, 1); % weight vector with 12 weights
Weights=[];
for i=1:3
    switch i
        case {1}
            complex=K1(1,1:l);
            %complex=sgolayfilt(complex,3,29)';
            %figure
            %plot(complex)
        case {2}
            complex=K2(1,1:l);
        case {3}
            complex=K3(1,1:l);
    end
\[ d_k = \text{complex}(1:l); \]
\[ X_k = \text{toeplitz(zeros(1,L)),complex}; \]
\[ \text{for } k=1:l \quad \text{% iteration process} \]
  \[ e_k = d_k(k) - W_k'X_k(:,k); \]
  \[ \text{error} = [\text{error } e_k^2]; \]
  \[ W_k = W_k + 2*\mu e_k X_k(:,k); \]
  \[ \text{Weights} = [\text{Weights } W_k]; \]
\[ \text{end} \]
\[ \text{end} \]
\[ t = \text{linspace(1,3*1,3*1)}; \]
\[ \text{%subplot(3,2,i)} \]
\[ \text{plot(t,Weights)}; \]
\[ \text{end} \]