CLASSIFIER TRAINING

This function trains a new classifier based on a list of labelled training EEG/LFP files.

INPUTS:

1. len_thr: minimum length of a seizure (in ms)

2. nleo_thr: minimum non-linear energy value of the seizure segment. Typical values are 5 (many candidate segments are selected) - 100 (very few candidate segments are selected) 10-20 (reasonable values based on our experience

3. flag: use features as for the chemical or for the genetic model (see reference paper for more information!) 'chemical' or 'genetic'

4. path: the path to the folder where the training files are saved

5. training_files: a sting contaiing the name of .mat file. The file contains a list of training file as a cell array of strings with the filenames (see training_files.mat as an example.

OUTPUTS:

1. SVMModel: SVM classifier object, as defined by Matlab (see help fitcsvm). This object should be saved as a .mat file, and later called in the detect_spikes_final function.

Example use:

[SVMModel]=train_svm_zebrafish(50,10,'chemical',cd,'training_files')

Other prerequisits:

1. For each string x in training_files must exist a x.mat, and x.txt file. x.mat contains the variabel 'labels', which is a vector of 1s and 0s, 1 meaning seizure. Each entry corresponds to a 100ms segment of LFP data training files. (see example example.mat and example.csv) In the future an artifact_label variable could also be included here, in which case lines 43-46 should be modified. x.txt is the LFP file (see File > Open for format specifications)

2. kfoldLoss function is not available in 2014b version of Matlab!

3. The function assumes that the data is sampled at 1000Hz and it is 10 mintues long!

```
function
[SVMModel]=train_svm_zebrafish(len_thr,nleo_thr,flag,path,training_files)




load(training_files);

for i=1:numel(training_files)
    disp(num2str(i))
    %feature_vector{i}=extract_features([ training_files{i}]);
 feature_vector{i}=feature_vector{i}(:,1:6000);
     load([path '/' training_files{i} '.mat']);
```

```
    class_label{i}=labels;%(1:6000);

end

% The following lines should be modified in case artifacts labels are
% provided! In the current version the artifact_labels are just made
% equal to 0 at line 185!

 for i=1:numel(training_files)
      load([path '/' training_files{i} '.mat']);
     artifact_label{i}=zeros(size(class_label{i}));
 end


for i=1:numel(training_files)
         res=10;
        fs=1000;


        eeg=csvread([path '/' training_files{i} '.txt']);
        eeg=eeg(:,1);



        % 'From which second is the EEG stable? (detection starts from
 here): ';
        tmp=zeros(1,100);
        for j=1:100
            tmp(j)=max(abs(eeg((j-1)*fs+1:j*fs)));
            if tmp(j)<1
                break;
            end
        end
        eeg(1:(j-1)*fs)=eeg((j-1)*fs+1);

        % interpolate large artifacts

        %eeg(1)=0;
        x=1:numel(eeg);
        xi=x(find(abs(eeg)<1));
        yi=eeg(find(abs(eeg)<1));
        eegi=interp1(xi,yi,x,'linear');
        clear xi yi;

        % high pass filter again baseline fluctuation

        if strcmp(flag,'chemical')
        [b,a] = butter(6,10/(fs/2),'high');
        elseif strcmp(flag,'genetic')
        [b,a] = butter(6,1/(fs/2),'high');
        end
        eegfilt=filtfilt(b,a,eegi);
        eegfilt=zscore(eegfilt);
```

```
        eeg=eegfilt;
        clear eegi eegfilt;

        disp('preprocessing done...')


    eegs=zscore(eeg)';
    %eegs=eegs(1:10*60*fs);
    sig_kaiser = kaiserVarRT(eegs);
    cand=find(sig_kaiser>=nleo_thr);
    len=zeros(size(cand));
    fin=cand;


    % cluster the close high energy samples together
    for j=length(cand):-1:2
        if cand(j)-cand(j-1)<200  %% if they are not 200ms apart, they
 belong to the same event
            fin(j-1)=fin(j);
            len(j-1)=fin(j-1)-cand(j-1);
            len(j)=0;
        end
    end

    clear selector;
    c=find(len>len_thr);
    if isempty(c)
        feature_vector{i}=[];
        selector=[];
    else
        for j=1:numel(c)
            candfin=zeros(size(eegs));
            candfin(cand(c(j)):cand(c(j))+len(c(j)))=1;
            %candfin(cand(c(j)))=1;
            selector{j}=unique(ceil(find(candfin)/(fs/res)));
            segment=cand(c(j)):cand(c(j))+len(c(j));
            feature_vector{i}
(j,:)=extract_features_segment(eeg(segment),flag);

        end
    end



    class_orig=class_label;

    %feature_vector{i}=feature_vector{i}(selector,:);
    clear tmp1 tmp2;
    for j=1:numel(selector)
        tmp1(j)=any(class_label{i}(selector{j}));
        tmp2(j)=any(artifact_label{i}(selector{j}));
    end
    if isempty(selector)
        class_label{i}=[];
```

```
            artifact_label{i}=[];
        else
            class_label{i}=double(tmp1);
            artifact_label{i}=double(tmp2);
        end

    end


            tr=(1:numel(training_files));


            type= 'classification';

            ff=cell2mat(feature_vector(tr)'); ff(isnan(ff(:,end)),end)=0;
     ff(isnan(ff(:,end-1)),end-1)=0;%max(ff(:,end-1));

            ll=cell2mat(class_label(tr))';
            aa=cell2mat(artifact_label(tr))';

            % in the following line if some segments were marked as both
            % seizure and artifact, then the artifact label is ignored!
            neg=find(ll==0); pos=find(ll==1); art=find(aa==1&ll==0);
            %p=randperm(numel(neg));
            neg=double(unique([neg; art]));
            pos=double(pos);

            if ~isempty(pos) && ~isempty(neg)


                        c = cvpartition(numel([neg; pos]),'KFold',5);

                        grid=-3:1:3;
                        m=numel(grid);

                        fval = zeros(m,1);
                        z = zeros(m,1);
                        for j = 1:m;

                            %[searchmin fval(j)] =
     fminsearch(minfn,grid(j),opts);
                            loss(j)=kfoldLoss(fitcsvm(ff([neg;
     pos],:),ll([neg;
     pos]),'CVPartition',c,'KernelFunction','linear','BoxConstraint',exp(grid(j))));

                        end

                        z = grid(loss == min(loss)); z=z(1);
                        SVMModel = fitcsvm(ff([neg; pos],:),ll([neg;
     pos]),'KernelFunction','linear','BoxConstraint',exp(z));



            end
```

```
function sig_kaiser = kaiserVarRT(sig)
%This function calculates the energy of the signal with the teager-
kaiser
%energy operator. This energy is related to the square of the
 immediate
%amplitude and frequency. The operator enhances spike activity
 facilitating
%the segmentation.

[ySize, xSize] = size(sig);
sig_kaiser = zeros(ySize, xSize);

for c=1:xSize
    sig_kaiser(4:end,c) = (sig(3:end-1,c).*sig(2:end-2,c))-
(sig(4:end,c).*sig(1:end-3,c));
end
sig_kaiser=abs(sig_kaiser);

return;

function scales = make_scales(startFreq,endFreq,freqRes,wl,fs)
% scales = make_scales(startFreq,endFreq,freqRes,wl)
% Function that determines the scales needed as input for a CWT with a
% certain wavelet. The start frequency (in Hz) is the first input
 argument,
% the end frequency (in Hz) the second, the frequency resolution at
 which
% the transformation should be calculated, the third. The final
 argument is
% the name of the wavelet used for the CWT. A sampling of 250Hz of the
 data
% is assumed.
%

centerFreq = centfrq(wl);
counter=1;
for i=startFreq:freqRes:endFreq
    scales(counter) = ((centerFreq*fs)/(i));
    counter=counter+1;
end


function featvec=extract_features_segment(eeg,flag)
% flag: specify if feature extraction according to the genetic or
 chemical model should be run (see reference paper for details!)

%
 fs=1000;
%
%
if strcmp(flag,'chemical')
```

```
% compute wavelet coefficients
%
 scales = make_scales(10,100,1,'morl',fs);
 freqs=10:1:100;
 f10=find(freqs==10);
 f20=find(freqs==20);
 f50=find(freqs==50);

%
    wc=cwt(eeg,scales,'morl').^2;

for i=1:length(scales)
    wc(i,:)=wc(i,:)./std(wc(i,:));
end

% mean wavelet coef. in 1-50Hz band
wc20=mean(wc(f10+1:f20,:),1);
wc50=mean(wc(f20+1:f50,:),1);
wc100=mean(wc(f50+1:end,:),1);


    i=1;
% feature 1: time domain
     zamp(i)=max(abs(eeg));
     rms(i)=sqrt(mean(eeg.^2));

% feature 2: the aveage wavelet coef. in 1-50Hz band exceeds the 20x
 standard deviation
   %  mwc10(i)=mean(wc10);
     mwc20(i)=mean(wc20);
     mwc50(i)=mean(wc50);
     mwc100(i)=mean(wc100);
     mwc(:,i)=mean(wc,2);


% criterion 3: zero crossings, mins, maxes

    [zeronum(i),
 maxnum(i),minnum(i),m_zero_intvl(i),std_zero_intvl(i)]=countzerominmax(zscore(eeg
    zeronum=zeronum/length(eeg);
    maxnum=maxnum/length(eeg);
    minnum=minnum/length(eeg);

%GEN
%featvec=[zamp; rms; mwc10; mwc20; mwc50; mwc100; mwc; wcr10; wcr20;
 zeronum; maxnum; minnum; m_zero_intvl; std_zero_intvl]';
%CHEM
featvec=[zamp; rms; mwc; mwc20; mwc50; mwc100; zeronum; maxnum;
 minnum; m_zero_intvl; std_zero_intvl]';

elseif strcmp(flag,'genetic')

    % compute wavelet coefficients
%
```

```
 scales = make_scales(2,100,1,'bior1.5',fs);

    wc=cwt(eeg,scales,'bior1.5').^2;

for i=1:length(scales)
    wc(i,:)=zscore(wc(i,:));
end

% mean wavelet coef. in 1-50Hz band
wc10=mean(wc(1:10,:),1);
wc20=mean(wc(1:20,:),1);
wc50=mean(wc(1:50,:),1);
wc100=mean(wc(1:end,:),1);

    i=1;
% feature 1: time domain
     zamp(i)=max(abs(eeg));
     rms(i)=sqrt(mean(eeg.^2));

% feature 2: the aveage wavelet coef. in 1-50Hz band exceeds the 20x
 standard deviation
     mwc10(i)=mean(wc10);
     mwc20(i)=mean(wc20);
     mwc50(i)=mean(wc50);
     mwc100(i)=mean(wc100);
     mwc(:,i)=mean(wc,2);

% criterion 3: high and low frequency ratio
     wcr10(i)=mean(abs(wc100))/mean(abs(wc10));
     wcr20(i)=mean(abs(wc100))/mean(abs(wc20));

% criterion 4: zero crossings, mins, maxes

    [zeronum(i),
 maxnum(i),minnum(i),m_zero_intvl(i),std_zero_intvl(i)]=countzerominmax(zscore(eeg
    zeronum=zeronum/length(eeg);
    maxnum=maxnum/length(eeg);
    minnum=minnum/length(eeg);



featvec=[zamp; rms; mwc10; mwc20; mwc50; mwc100; mwc; wcr10; wcr20;
 zeronum; maxnum; minnum; m_zero_intvl; std_zero_intvl]';

end


function [zeronum,
 maxnum,minnum,m_zero_intvl,std_zero_intvl]=countzerominmax(x)
% This function calualates the number zero crossing, minima, maxima,
 and
% the mean and standard deviation of zero-to-zero intervals
zeronum=0;
maxnum=0;
```

```
minnum=0;
zero_idx=[];
for i=2:length(x)-1
    if sign(x(i-1))~=sign(x(i))
        zeronum=zeronum+1;
        zero_idx=[zero_idx i];
    end

    zero_intvl=zero_idx(2:end)-zero_idx(1:end-1);

    m_zero_intvl=mean(zero_intvl);
    std_zero_intvl=std(zero_intvl);



    if x(i-1)<x(i) && x(i)>x(i+1)
        maxnum=maxnum+1;
    end

    if x(i-1)>x(i) && x(i)<x(i+1)
        minnum=minnum+1;
    end

end
```

```
Not enough input arguments.

Error in train_svm_zebrafish (line 57)
load(training_files);
```

Published with MATLAB® R2016a