Multiclass Classification For Basic Hand Movements

Presented by

Emergency Bacon

Wenlong Zhao, Kening Zhang, Yuesheng Luo, Zijing Gu

1. Project Motivation
2. Project Overview
3. Dataset
4. Related Work
5. Methods, Results, & Analysis
6. Analysis Summary
7. Discussion
8. References

Project Motivation

Inspiration

- Human Activity Recognition with Smartphones -- Ricky Lieu, Wangting Cui

EMG based hand movement recognition dataset

- Application

Project Overview

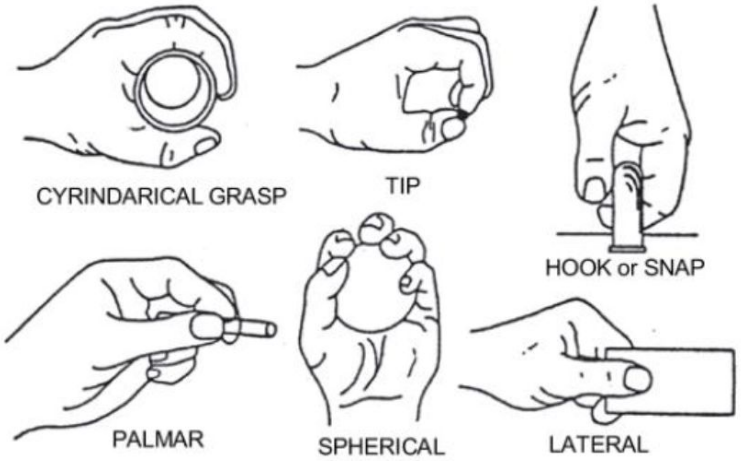
Dataset: sEMG for Basic Hand movements Data Set from UCI Machine Learning Repository

Goal: six-class classification

Approaches: boosting, random forests, feedforward NN, linear classifier

Improvement: EMD, PCA, etc.

Tools: MATLAB, Python



Dataset

One healthy subject (male, 22-year-old) conducted 6 grasps for 100 times for 3 consecutive days

3 mat files

6 x 100 = 600 data vectors per file

3 x 600 = 1800 data vectors in total

Dataset

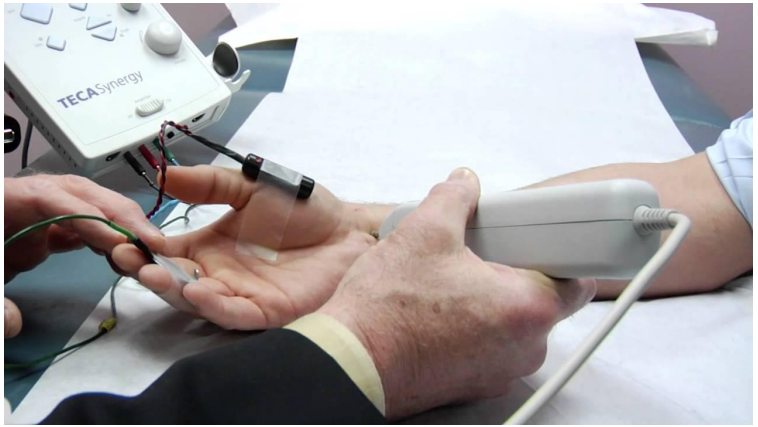
Elastic bands and the reference electrode

Measure time: 5 secs per grasp

500 Hz sampling rate

2-channel EMG system

5 x 500 x 2 = 5000 features per data vector



Related Work

Knowledge used from class:

Feedforward Neural Networks
Principal Component Analysis
...

Literature Review:

Sapsanis, C., Georgoulas, G., Tzes, A., & Lymberopoulos, D. (2013). Improving EMG based classification of basic hand movements using EMD. 2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). doi:10.1109/embc.2013.6610858

Methods, Results, & Analysis

1. Data Structure (20s)
2. MATLAB Neural Network Toolbox(40s)
3. runNN.m & loopNN.m (60s)
4. Challenge (30s)
5. Adaptive Boosting (45s)
6. Random Decision Forest (45s)
7. Another Direction of Thinking (12s)
8. Extract Empirical Mode Decomposition (60s)
9. Last Modification (5s)
10. Principal Component Analysis (13s)

1. Data Structure

concate.m

```
function[m]=concate()
    load('Database 2\male_day_1.mat')
    mCyl1=horzcat(cyl_ch1, cyl_ch2);
    mHook1=horzcat(hook_ch1,hook_ch2);
    mTip1=horzcat(tip_ch1, tip_ch2);
    mPalm1=horzcat(palm_ch1,palm_ch2);
    mSpher1=horzcat(spher_ch1,spher_ch2);
    mLat1=horzcat(lat_ch1,lat_ch2);
    m1={mCyl1,mHook1,mTip1,mPalm1,mSpher1,mLat1};
```

1) Features

5000 features representing EMG signals from 2 channels.

1800x5000 double

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0.1232	0.1997	0.2762	0.0722	0.1487	0.1232	0.0212	0.1232	0.1742	0.1232	0.3272	0.2252	0.199
2	0.1232	0.2252	0.1997	0.1742	0.1997	0.0212	0.0977	0.1232	0.1742	0.1997	0.1487	0.1742	0.225
3	0.1742	0.2252	0.1232	0.1742	0.0212	0.2252	0.0977	0.0212	0.5312	0.3782	0.0467	0.0977	-0.080
4	0.1487	0.1742	0.0212	0.1487	0.0722	0.0722	0.3527	0.2252	0.1997	-0.2848	0.4037	0.3527	0.021
5	0.2507	0.0722	0.1232	-0.3868	0.5057	0.4547	0.2252	-0.1318	-0.0043	-0.0808	0.2252	0.2252	0.123
6	0.1232	0.1997	0.1232	0.0722	0.2252	0.1487	0.6842	-0.4123	0.1232	0.5312	0.1742	0.1997	0.097
7	0.0722	0.0977	0.0212	0.1232	0.1742	0.1487	0.1997	0.2507	0.0977	0.1232	0.0977	0.0467	0.174
8	0.0212	0.1232	0.1742	0.1232	0.3017	0.1232	0.1997	0.1487	0.0977	0.0977	0.1232	0.1487	0.148
9	0.0722	-0.1318	0.2252	0.3527	0.1742	0.1232	0.1742	0.0722	0.1997	0.1232	0.1997	0.0212	0.148

2) Labels of Two Versions

feature | label | labelPy

1800x1 double

	1	2	3
1489	4		
1490	4		
1491	4		
1492	4		
1493	4		
1494	4		
1495	4		
1496	4		
1497	4		
1498	4		
1499	4		
1500	4		
1501	5		
1502	5		
1503	5		
1504	5		
1505	5		
1506	5		
1507	5		

feature | label | labelPy

1800x6 double

	1	2	3	4	5	6
589	0	1	0	0	0	0
590	0	1	0	0	0	0
591	0	1	0	0	0	0
592	0	1	0	0	0	0
593	0	1	0	0	0	0
594	0	1	0	0	0	0
595	0	1	0	0	0	0
596	0	1	0	0	0	0
597	0	1	0	0	0	0
598	0	1	0	0	0	0
599	0	1	0	0	0	0
600	0	1	0	0	0	0
601	0	0	1	0	0	0
602	0	0	1	0	0	0
603	0	0	1	0	0	0
604	0	0	1	0	0	0
605	0	0	1	0	0	0
606	0	0	1	0	0	0
607	0	0	1	0	0	0

Each gesture is associated with a number (label), corresponding to feature input matrix.

Cyrindarical: 0
Tip: 1
Hook: 2
Palmar: 3
Spherical: 4
Lateral: 5

2. Neural Network Toolbox

3. RunNN.m & LoopNN.m

```
function [trPercentAcc,vaPercentAcc,tstPercentAcc] = runNN(trainingFunc,hiddenLayer,performFunc,feature,label)
% Solve a Pattern Recognition Problem with a Neural Network

x = feature';
t = label';

trainFcn = trainingFunc;

% Create a Pattern Recognition Network
hiddenLayerSize = hiddenLayer;
net = feedforwardnet(hiddenLayerSize, trainFcn);

% Choose Input and Output Pre/Post-Processing Functions
net.input.processFcns = {'removeconstantrows','mapminmax'};
net.output.processFcns = {'removeconstantrows','mapminmax'};

% Setup Division of Data for Training, Validation, Testing
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 60/100;
net.divideParam.valRatio = 20/100;
net.divideParam.testRatio = 20/100;

```

```
% Choose a Performance Function
net.performFcn = performFunc;

% Choose Plot Functions
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotconfusion', 'plotroc'};

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y);
tind = vec2ind(t);
yind = vec2ind(y);
y_int= ind2vec(yind);
percentErrors = sum(tind ~= yind)/numel(tind);

```

```
% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
trainPredictions = y_int .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
valPredictions = y_int .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
testPredictions = y_int .* tr.testMask{1};
tindTr=zeros(1,size(t,2)); % Initial as 0s
tindVa=zeros(1,size(t,2)); % Initial as 0s
tindTst=zeros(1,size(t,2)); % Initial as 0s
yindTr=ones(1,size(t,2))*9; % Initial as 9s
yindVa=ones(1,size(t,2))*9; % Initial as 9s
yindTst=ones(1,size(t,2))*9; % Initial as 9s
nTr=0;nVa=0;nTst=0;
```

```
for r = 1:size(t,1)
    for c = 1:size(t,2)
        if trainTargets(r,c)==1
            tindTr(c)=r; nTr=nTr+1;
        end
        if trainPredictions(r,c)==1
            yindTr(c)=r;
        end
        if valTargets(r,c)==1
            tindVa(c)=r; nVa=nVa+1;
        end
        if valPredictions(r,c)==1
            yindVa(c)=r;
        end
        if testTargets(r,c)==1
            tindTst(c)=r; nTst=nTst+1;
        end
        if testPredictions(r,c)==1
            yindTst(c)=r;
        end
    end
end
trPercentAcc = sum(tindTr == yindTr)/nTr;
vaPercentAcc = sum(tindVa == yindVa)/nVa;
tstPercentAcc = sum(tindTst == yindTst)/nTst;
```

```
function outputForm = loopNN(nLoop, feature, label)

%array of training functions
trainFuncArray=[string('trainscg')];

%array of hidden layer numbers
hiddenLayerArray = [10,40,160,640,2560];
%hiddenLayerArray = [10,20,40,60,80];

%array of performance functions
performFuncArray=[string('mse')];

%results in a string array
outputForm = strings(length(trainFuncArray)*length(hiddenLayerArray)*length(performFuncArray)*6+1,7);
outputForm= [string('TrainingFcn'), string('HiddenLayers'), string('PerformanceFcn'), string('NumOfLoops'), ...
    string('trainingAcc'), string('validationAcc'), string('testingAcc')];
row = 1;

```

NNResult

31x7 string

	1	2	3	4	5	6	7
1	TrainingFcn	Neurons	PerformanceFcn	NumOfLoops	trainingAcc	validationAcc	testingAcc
2	trainscg	10	mse	1	0.21019	0.15833	0.12222
3	trainscg	10	mse	2	0.2	0.18611	0.17778
4	trainscg	10	mse	3	0.22593	0.15278	0.17222
5	trainscg	10	mse	4	0.175	0.16389	0.19167
6	trainscg	10	mse	5	0.33519	0.14722	0.16944
7	trainscg	10	mse	avg	0.22926	0.16167	0.16667
8	trainscg	40	mse	1	0.41481	0.25	0.22222
9	trainscg	40	mse	2	0.30556	0.16667	0.175
10	trainscg	40	mse	3	0.27593	0.16944	0.16667
11	trainscg	40	mse	4	0.34722	0.19722	0.16111
12	trainscg	40	mse	5	0.34444	0.16944	0.17222
13	trainscg	40	mse	avg	0.33759	0.19056	0.17944
14	trainscg	160	mse	1	0.62315	0.21944	0.26667
15	trainscg	160	mse	2	0.59907	0.20556	0.225
16	trainscg	160	mse	3	0.47778	0.16389	0.18056
17	trainscg	160	mse	4	0.55463	0.19167	0.22778
18	trainscg	160	mse	5	0.49074	0.21111	0.23333
19	trainscg	160	mse	avg	0.54907	0.19833	0.22667
20	trainscg	640	mse	1	0.83056	0.25556	0.24444
21	trainscg	640	mse	2	0.97593	0.20556	0.21667
22	trainscg	640	mse	3	0.85278	0.23056	0.25278
23	trainscg	640	mse	4	0.90926	0.22222	0.20278

24	trainscg	640	mse	5	0.86019	0.23889	0.18889
25	trainscg	640	mse	avg	0.88574	0.23056	0.22111
26	trainscg	2560	mse	1	0.84815	0.24444	0.23333
27	trainscg	2560	mse	2	0.97315	0.27222	0.21944
28	trainscg	2560	mse	3	0.99815	0.23611	0.25556
29	trainscg	2560	mse	4	0.66389	0.24444	0.21389
30	trainscg	2560	mse	5	0.67037	0.26389	0.24444
31	trainscg	2560	mse	avg	0.83074	0.25222	0.23333

4. Challenge

How shall we face failure?

Overfitting or underfitting?

24	trainscg	640	mse	5	0.86019	0.23889	0.18889
25	trainscg	640	mse	avg	0.88574	0.23056	0.22111
26	trainscg	2560	mse	1	0.84815	0.24444	0.23333
27	trainscg	2560	mse	2	0.97315	0.27222	0.21944
28	trainscg	2560	mse	3	0.99815	0.23611	0.25556
29	trainscg	2560	mse	4	0.66389	0.24444	0.21389
30	trainscg	2560	mse	5	0.67037	0.26389	0.24444
31	trainscg	2560	mse	avg	0.83074	0.25222	0.23333

5. Adaptive Boosting

Adaptive Boosting

```
# boosting (not work well, but should improve the result. Work on parameters)
bdt_real = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=4, criterion='entropy', max_features='sqrt'),
    n_estimators=2000,
    learning_rate=1.0)

bdt_real.fit(X_train, Y_train)
```

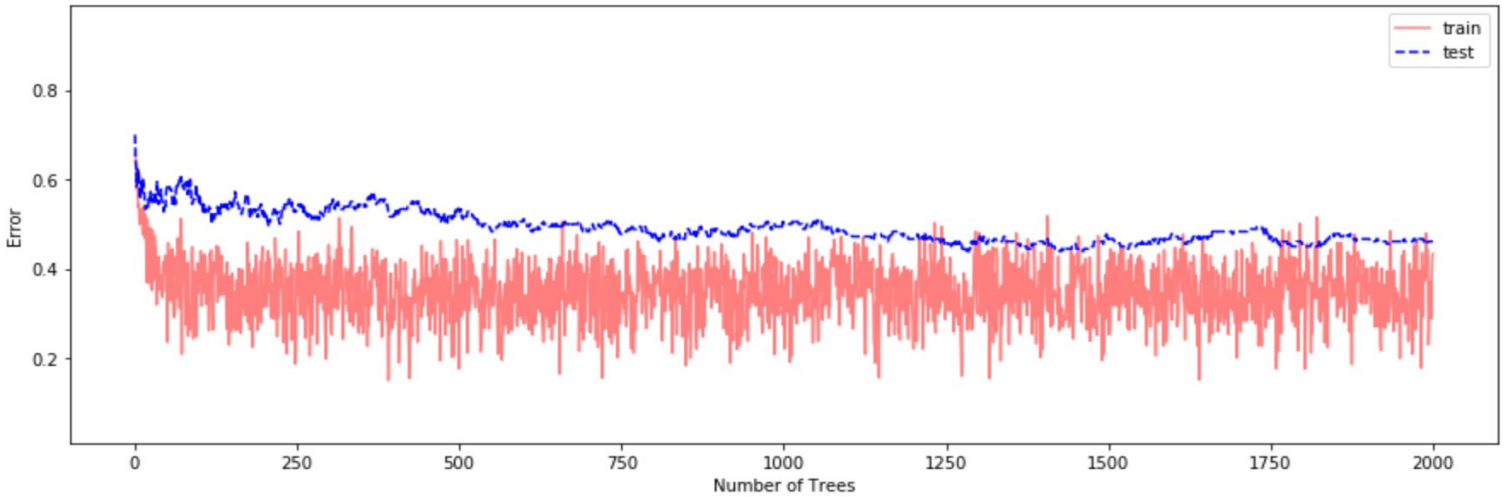
```
# test error
real_test_errors = []
for real_test_predict in bdt_real.staged_predict(X_test):
    real_test_errors.append(
        1. - accuracy_score(real_test_predict, Y_test))

n_trees_real = len(bdt_real)
```

```
# train error
real_estimator_errors = bdt_real.estimator_errors_[:n_trees_real]
```

Adaptive Boosting

```
plt.show()
```




```
confusion_matrix(Y_test, real_test_predict)
```

```
array([[18,  1,  0,  0,  0, 14],
       [ 0, 19,  3, 13,  0,  2],
       [ 0,  0, 13, 13,  0,  5],
       [ 0,  0,  3, 15,  0,  4],
       [ 2,  0,  0,  0, 17, 14],
       [ 0,  0,  2,  7,  0, 15]])
```

6. Random Forest

Random Forest

```
# random forest
rf = RandomForestClassifier(max_depth=20, criterion='entropy', n_estimators=5000, n_jobs=2,
                            max_features='auto', verbose=1, oob_score=True, min_samples_split=5)
rf.fit(X_train, Y_train)
```

```
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    2.9s
[Parallel(n_jobs=2)]: Done 196 tasks      | elapsed:   12.0s
[Parallel(n_jobs=2)]: Done 446 tasks      | elapsed:   27.0s
[Parallel(n_jobs=2)]: Done 796 tasks      | elapsed:   48.7s
[Parallel(n_jobs=2)]: Done 1246 tasks      | elapsed:  1.2min
[Parallel(n_jobs=2)]: Done 1796 tasks      | elapsed:  1.8min
[Parallel(n_jobs=2)]: Done 2446 tasks      | elapsed:  2.5min
[Parallel(n_jobs=2)]: Done 3196 tasks      | elapsed:  3.3min
[Parallel(n_jobs=2)]: Done 4046 tasks      | elapsed:  4.1min
[Parallel(n_jobs=2)]: Done 4996 tasks      | elapsed:  5.1min
[Parallel(n_jobs=2)]: Done 5000 out of 5000 | elapsed:  5.1min finished
```

Random Forest

```
# train accuracy
accuracy_score(rf.predict(X_train), Y_train)
```

```
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 196 tasks      | elapsed:    0.1s
[Parallel(n_jobs=2)]: Done 446 tasks      | elapsed:    0.2s
[Parallel(n_jobs=2)]: Done 796 tasks      | elapsed:    0.4s
[Parallel(n_jobs=2)]: Done 1246 tasks      | elapsed:    0.6s
[Parallel(n_jobs=2)]: Done 1796 tasks      | elapsed:    0.9s
[Parallel(n_jobs=2)]: Done 2446 tasks      | elapsed:    1.2s
[Parallel(n_jobs=2)]: Done 3196 tasks      | elapsed:    1.5s
[Parallel(n_jobs=2)]: Done 4046 tasks      | elapsed:    1.9s
[Parallel(n_jobs=2)]: Done 4996 tasks      | elapsed:    2.3s
[Parallel(n_jobs=2)]: Done 5000 out of 5000 | elapsed:    2.3s finished
```

```
1.0
```

```
confusion_matrix(Y_test, Y_pred)
```

```
array([[31,  0,  0,  0,  2,  0],
       [ 2, 30,  2,  2,  0,  1],
       [ 0,  0, 20,  5,  0,  6],
       [ 0,  2,  7, 11,  0,  2],
       [ 0,  0,  0,  0, 33,  0],
       [ 0,  0,  5,  4,  3, 12]])
```

Random Forest

```
# test accuracy
Y_pred = rf.predict(X_test)
accuracy_score(Y_pred, Y_test)
```

```
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 196 tasks      | elapsed:    0.1s
[Parallel(n_jobs=2)]: Done 446 tasks      | elapsed:    0.1s
[Parallel(n_jobs=2)]: Done 796 tasks      | elapsed:    0.3s
[Parallel(n_jobs=2)]: Done 1246 tasks      | elapsed:    0.4s
[Parallel(n_jobs=2)]: Done 1796 tasks      | elapsed:    0.5s
[Parallel(n_jobs=2)]: Done 2446 tasks      | elapsed:    0.7s
[Parallel(n_jobs=2)]: Done 3196 tasks      | elapsed:    0.9s
[Parallel(n_jobs=2)]: Done 4046 tasks      | elapsed:    1.2s
[Parallel(n_jobs=2)]: Done 4996 tasks      | elapsed:    1.4s
[Parallel(n_jobs=2)]: Done 5000 out of 5000 | elapsed:    1.4s finished
```

```
0.76111111111111107
```

7. Another Direction of Thinking

8. Empirical Mode Decomposition

1) EMG->3 IMF Functions + residual

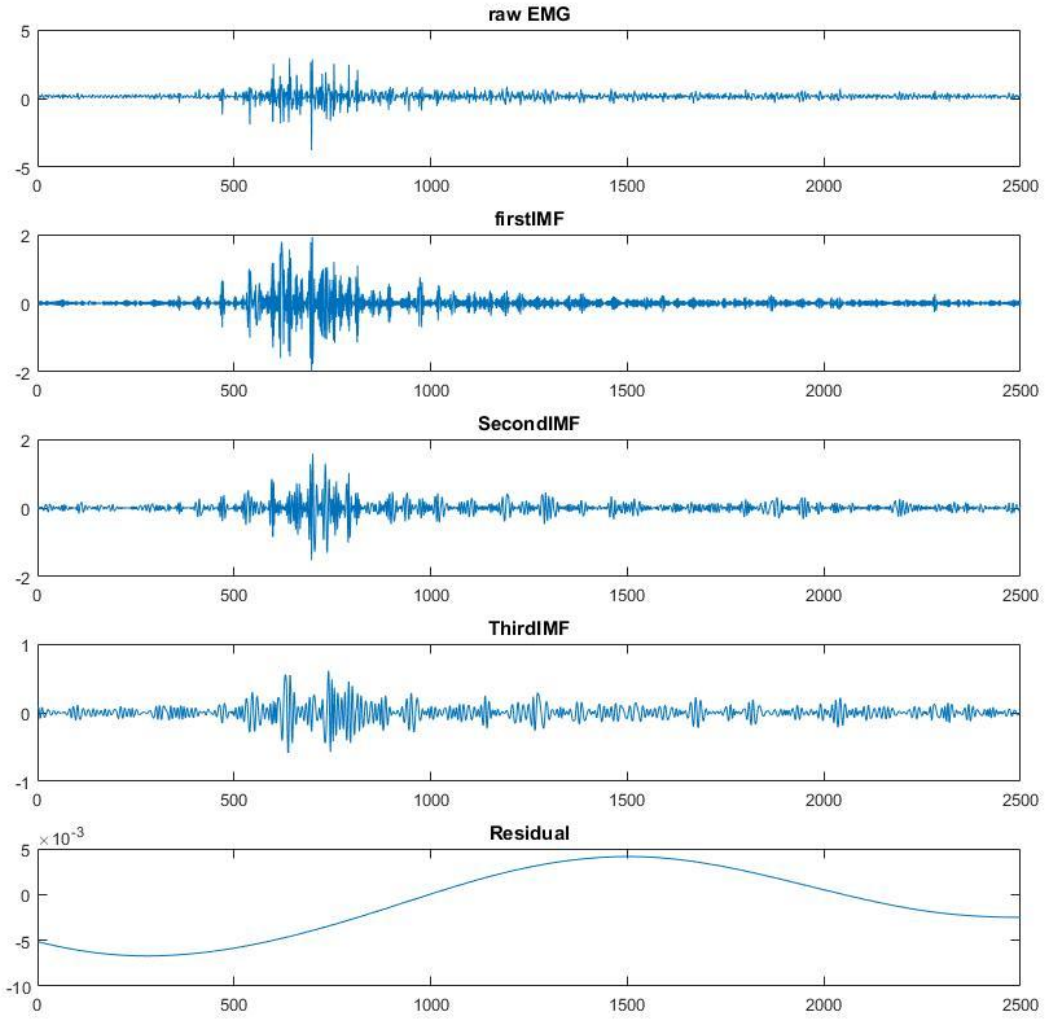
EMD acts as an adaptive non-linear filter, decomposing the signal into a number of IMFs, where an IMF represents a simple oscillatory function satisfying two conditions:

1. The number of zero crossings and the number of local extrema are either equal or differ by one.
2. The local average (defined by the average of local maximum and local minimum envelops) is equal to zero.

Reference:

Improving EMG based Classification of basic hand movements using EMD

Christos Sapsanis, George Georgoulas, Anthony Tzes and Dimitrios Lymberopoulos



Our EMG,IMF Function with Channel 1
EMD TOOLBOX
2500 for channel 1

```
IMFs=emd(feature(1,1:2500));
first=IMFs(1,:);
second=IMFs(2,:);
third=IMFs(3,:);
residual=IMFs(11,:);

subplot(5,1,1);
plot(feature(1,1:2500));
title('raw EMG');

subplot(5,1,2);
plot(first);
title('firstIMF');

subplot(5,1,3);
plot(second);
title('SecondIMF');

subplot(5,1,4);
plot(third);
title('ThirdIMF');

subplot(5,1,5);
plot(residual);
title('Residual');
```

2) Extract features from EMG function

1) Integrated Electromyogram (IEMG):

This feature is an average value of the absolute values of EMG, defined as $IEMG = \frac{1}{N}\sum_{k=1}^{N}|x_k|$,

where x_k is the k^{th} sample data out of N samples of EMG raw data.

2) Zero Crossing (ZC):

ZC counts the number of times that the signal crosses zero. A threshold needs to be introduced to reduce the noise induced at zero crossing. Given two contiguous EMG signals x_k and x_{k+1}, the ZC can he calculated as:

$$ZC = \sum f(x), \text{ where}$$

$$f(x) = \begin{cases} 1, & \text{if, } (x_k > 0 \; AND \; x_{k+1} < 0) \\ & \text{OR } (x_k < 0 \; AND \; x_{k+1} > 0 \\ 0, & \text{otherwise} \end{cases}$$

for $k = 1, 2, 3, \ldots N\text{-}1$

3) Variance (VAR):

VAR is a measure of the power density of the EMG signal given by:

$$VAR = \frac{1}{N-1}\sum_{k=1}^{N}(x_k - \mu)^2$$

where μ is the average.

4) Slope Sign Changes (SSC):

SSC counts the number of times the slope of the signal changes sign. Given three contiguous EMG signals x_{k-1}, x_k and x_{k+1} the number of slope sign changes can be calculated by $SSC = \sum f(x)$ where

$$f(x) = \begin{cases} 1, & \text{if, } (x_k < x_{k+1} \; AND \; x_k < x_{k-1}) \\ & \text{OR } (x_k > x_{k+1} \; AND \; x_k > x_{k-1}) \\ 0, & \text{otherwise} \end{cases}$$

for $k = 1, 2, 3, \ldots (N\text{-}1)$

5) Waveform Length (WL)

WL is a cumulative variation of the EMG that can indicate the degree of variation about the EMG signal. It is given by $WL = \sum_{k=1}^{N-1}(|x_{k+1} - x_k|)$

6) Willison Amplitude (WAMP)

WAMP is the number of counts for each change of the EMG signal amplitude that exceeds a defined threshold. It can indicate the muscle contraction level as given by

$$WAMP = \sum_{k=1}^{N-1} f(|x_{k+1} - x_k|)$$

$$f(x) = \begin{cases} 1, & \text{if } x > \text{threshold} \\ 0, & \text{otherwise} \end{cases}$$

7) Kurtosis:

The kurtosis of a distribution is defined as $k = \frac{E(x-\mu)^4}{\sigma^4}$

8) Skewness

The skewness of a distribution is defined as $s = \frac{E(x-\mu)^3}{\sigma^3}$

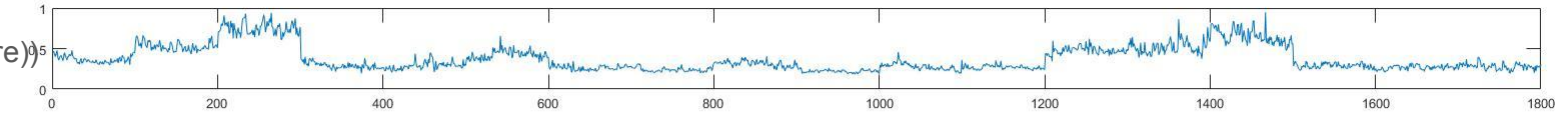
Reference:

Improving EMG based Classification of basic hand movements using EMD

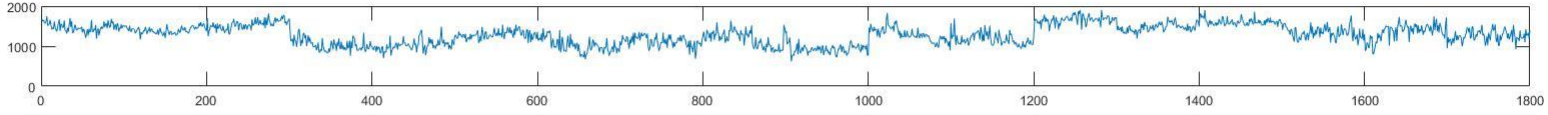
Christos Sapsanis, George Georgoulas, Anthony Tzes and Dimitrios Lymberopoulos

8 Feature Extraction Functions For 6*300 Trials

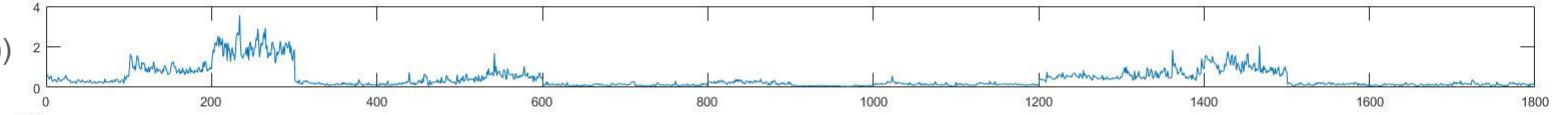
>>plot(IEMG(feature))



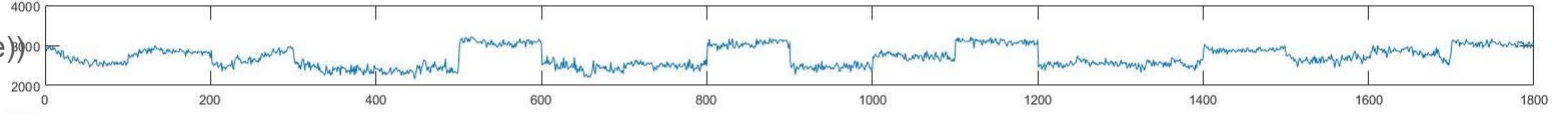
>> plot(ZC(feature))



>> plot(Var(feature))



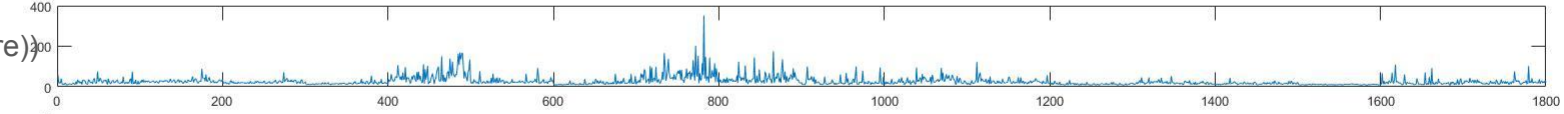
>> plot(SSC(feature))



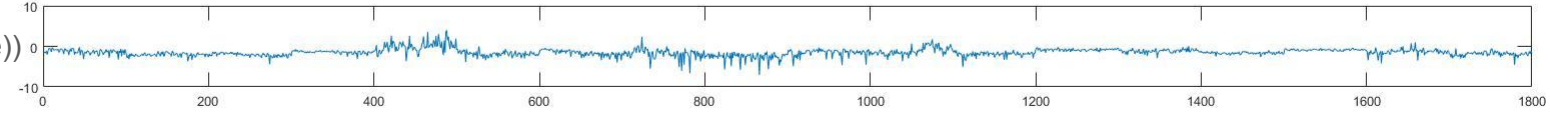
>> plot(WL(feature))

>> plot(WAMP(feature))

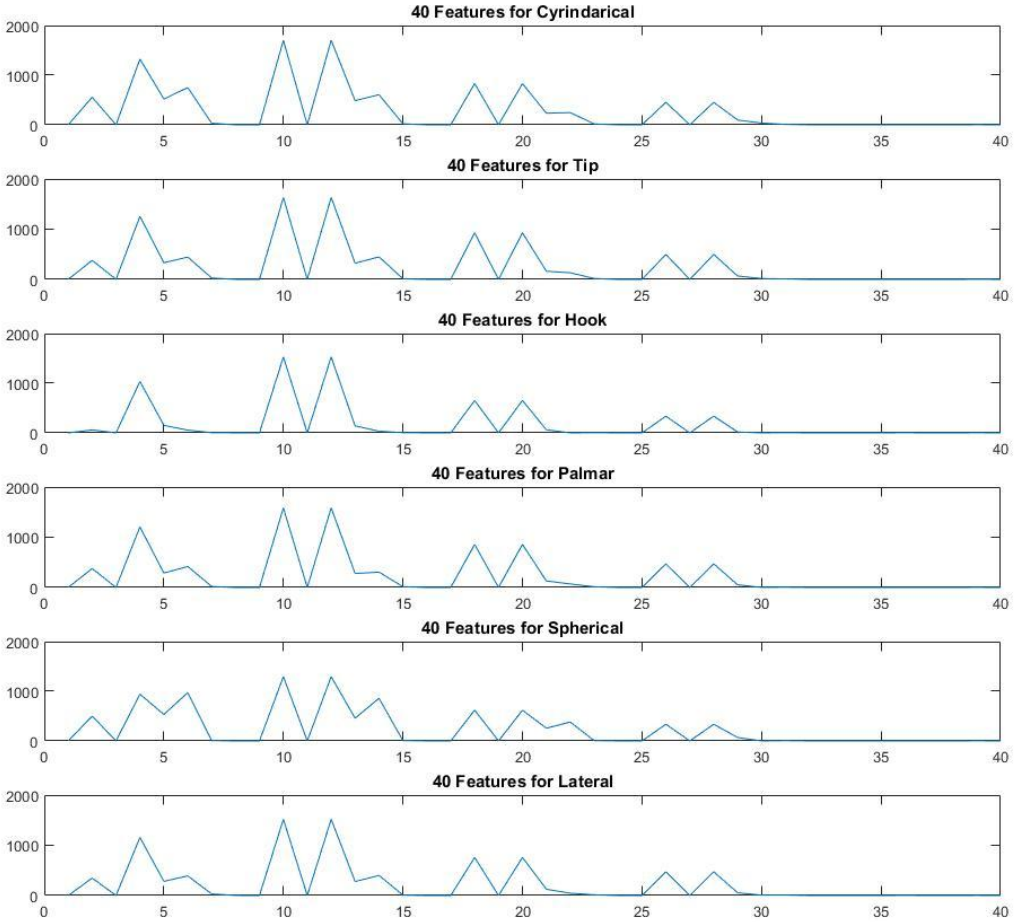
>> plot(KUR(feature))



>> plot(SKE(feature))



40 Features in total




```
subplot(6,1,1);
plot(feature1(1,:));
title('40 Features for Cyrindarical');

subplot(6,1,2);
plot(feature1(301,:));
title('40 Features for Tip');

subplot(6,1,3);
plot(feature1(601,:));
title('40 Features for Hook');

subplot(6,1,4);
plot(feature1(901,:));
title('40 Features for Palmar');

subplot(6,1,5);
plot(feature1(1201,:));
title('40 Features for Spherical');

subplot(6,1,6);
plot(feature1(1501,:));
title('40 Features for Lateral');
```

Got 1800*40

1) Linear Discriminant Classifier

```
load fisheriris
MdlLinear = fitcdiscr(featureLast,label);
accuracy=0;
for i=1:300
    predicted(i) = predict(MdlLinear,test(i));
    if predicted(i) == testLabel(i)
        accuracy=accuracy+1;
    end
end
accuracy=accuracy/300
```

```
accuracy =

    0.6500
```

2) Feedforward Neural Network

runNN.m

loopNN.m

	1	2	3	4	5	6	7
1	TrainingFcn	Neurons	PerformanceFcn	NumOfLoops	trainingAcc	validationAcc	testingAcc
2	trainscg	10	mse	1	0.87778	0.83056	0.79167
3	trainscg	10	mse	2	0.79074	0.76667	0.76667
4	trainscg	10	mse	3	0.87963	0.85833	0.83333
5	trainscg	10	mse	4	0.875	0.82778	0.81667
6	trainscg	10	mse	5	0.90556	0.875	0.83056
7	trainscg	10	mse	avg	0.86574	0.83167	0.80778
8	trainscg	20	mse	1	0.85741	0.83333	0.78333
9	trainscg	20	mse	2	0.82593	0.83056	0.82222
10	trainscg	20	mse	3	0.89444	0.83611	0.83333
11	trainscg	20	mse	4	0.85093	0.80833	0.81111
12	trainscg	20	mse	5	0.91111	0.85556	0.85278
13	trainscg	20	mse	avg	0.86796	0.83278	0.82056
14	trainscg	40	mse	1	0.87685	0.80556	0.83611
15	trainscg	40	mse	2	0.90648	0.85833	0.86667
16	trainscg	40	mse	3	0.95278	0.90278	0.88333
17	trainscg	40	mse	4	0.92778	0.85556	0.875
18	trainscg	40	mse	5	0.81574	0.775	0.78611
19	trainscg	40	mse	avg	0.89593	0.83944	0.84944
20	trainscg	60	mse	1	0.95093	0.88889	0.85833
21	trainscg	60	mse	2	0.94444	0.89167	0.88611
22	trainscg	60	mse	3	0.89907	0.85278	0.83333
23	trainscg	60	mse	4	0.92037	0.90278	0.825

24	trainscg	60	mse	5	0.94167	0.89444	0.89167
25	trainscg	60	mse	avg	0.9313	0.88611	0.85889
26	trainscg	80	mse	1	0.96204	0.91389	0.91389
27	trainscg	80	mse	2	0.95185	0.91389	0.9
28	trainscg	80	mse	3	0.85926	0.83889	0.80556
29	trainscg	80	mse	4	0.97685	0.92222	0.9
30	trainscg	80	mse	5	0.92037	0.92778	0.88611
31	trainscg	80	mse	avg	0.93407	0.90333	0.88111

3) Random Forest

Random Forest with Ch1 and Ch2

In [36]:
```
# random forest
rf = RandomForestClassifier(max_depth=20, criterion='entropy', n_estimators=5000, n_jobs=-1,
                            max_features='auto', verbose=1, oob_score=True, min_samples_split=5)
```

In [17]:
```
# test accuracy
Y_pred = rf.predict(X_test)
print accuracy_score(Y_pred, Y_test)

# train accuracy
print accuracy_score(rf.predict(X_train), Y_train)

confusion_matrix(Y_test, Y_pred)
```

```
[Parallel(n_jobs=40)]: Done 120 tasks      | elapsed:    0.1s
[Parallel(n_jobs=40)]: Done 370 tasks      | elapsed:    0.2s
[Parallel(n_jobs=40)]: Done 720 tasks      | elapsed:    0.3s
[Parallel(n_jobs=40)]: Done 1170 tasks      | elapsed:    0.4s
[Parallel(n_jobs=40)]: Done 1720 tasks      | elapsed:    0.7s
[Parallel(n_jobs=40)]: Done 2370 tasks      | elapsed:    0.9s
[Parallel(n_jobs=40)]: Done 3120 tasks      | elapsed:    1.1s
[Parallel(n_jobs=40)]: Done 3970 tasks      | elapsed:    1.3s
[Parallel(n_jobs=40)]: Done 4920 tasks      | elapsed:    1.6s
[Parallel(n_jobs=40)]: Done 5000 out of 5000 | elapsed:    1.6s finished
```

```
0.919444444444
```

```
[Parallel(n_jobs=40)]: Done 120 tasks      | elapsed:    0.1s
[Parallel(n_jobs=40)]: Done 370 tasks      | elapsed:    0.2s
[Parallel(n_jobs=40)]: Done 720 tasks      | elapsed:    0.3s
[Parallel(n_jobs=40)]: Done 1170 tasks      | elapsed:    0.5s
[Parallel(n_jobs=40)]: Done 1720 tasks      | elapsed:    0.7s
[Parallel(n_jobs=40)]: Done 2370 tasks      | elapsed:    0.9s
[Parallel(n_jobs=40)]: Done 3120 tasks      | elapsed:    1.2s
[Parallel(n_jobs=40)]: Done 3970 tasks      | elapsed:    1.5s
```

```
1.0
```

```
[Parallel(n_jobs=40)]: Done 4920 tasks      | elapsed:    1.9s
[Parallel(n_jobs=40)]: Done 5000 out of 5000 | elapsed:    1.9s finished
```

Out[17]:
```
array([[66,  0,  0,  0,  0,  0],
       [ 2, 51,  2,  0,  0,  0],
       [ 1,  1, 61,  5,  0,  4],
       [ 0,  1,  1, 49,  0,  7],
       [ 0,  0,  0,  0, 51,  0],
       [ 0,  2,  0,  3,  0, 53]])
```

91.9%!!!

9. More to Modify

Lack of principal features?

Annoying Feature:
Feature of SSC Does not seem to give us information helpful for classification.

In [40]:
```
# test accuracy
Y_pred = rf.predict(X_test)
print accuracy_score(Y_pred, Y_test)

# train accuracy
print accuracy_score(rf.predict(X_train), Y_train)

confusion_matrix(Y_test, Y_pred)
```

```
[Parallel(n_jobs=40)]: Done 120 tasks      | elapsed:    0.2s
[Parallel(n_jobs=40)]: Done 370 tasks      | elapsed:    0.3s
[Parallel(n_jobs=40)]: Done 720 tasks      | elapsed:    0.4s
[Parallel(n_jobs=40)]: Done 1170 tasks      | elapsed:    0.6s
[Parallel(n_jobs=40)]: Done 1720 tasks      | elapsed:    0.7s
[Parallel(n_jobs=40)]: Done 2370 tasks      | elapsed:    0.9s
[Parallel(n_jobs=40)]: Done 3120 tasks      | elapsed:    1.1s
[Parallel(n_jobs=40)]: Done 3970 tasks      | elapsed:    1.4s
[Parallel(n_jobs=40)]: Done 4920 tasks      | elapsed:    1.6s
[Parallel(n_jobs=40)]: Done 5000 out of 5000 | elapsed:    1.6s finished
[Parallel(n_jobs=40)]: Done 120 tasks      | elapsed:    0.1s
```

```
0.905555555556
```

Use the rest of 7 features to reperform the test

10. Principal Component Analysis

```
# PCA
from sklearn.decomposition import PCA
pca = PCA(n_components=50)
X_r = pca.fit(X).transform(X)

# Percentage of variance explained for each components
print('explained variance ratio: %s'
      % str(sum(pca.explained_variance_ratio_)))

use_pca = True

if use_pca:
    X_train, X_test, Y_train, Y_test = train_test_split(X_r, Y, test_size=0.20)
else:
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20)
```

```
explained variance ratio: 0.99999999937
```

In [32]:

```
# test accuracy
Y_pred = rf.predict(X_test)
print accuracy_score(Y_pred, Y_test)

# train accuracy
print accuracy_score(rf.predict(X_train), Y_train)

confusion_matrix(Y_test, Y_pred)
```

```
[Parallel(n_jobs=40)]: Done 120 tasks      | elapsed:    0.1s
[Parallel(n_jobs=40)]: Done 370 tasks      | elapsed:    0.2s
[Parallel(n_jobs=40)]: Done 720 tasks      | elapsed:    0.3s
[Parallel(n_jobs=40)]: Done 1170 tasks      | elapsed:    0.5s
[Parallel(n_jobs=40)]: Done 1720 tasks      | elapsed:    0.6s
[Parallel(n_jobs=40)]: Done 2370 tasks      | elapsed:    0.8s
[Parallel(n_jobs=40)]: Done 3120 tasks      | elapsed:    1.1s
[Parallel(n_jobs=40)]: Done 3970 tasks      | elapsed:    1.4s
[Parallel(n_jobs=40)]: Done 4920 tasks      | elapsed:    1.8s
[Parallel(n_jobs=40)]: Done 5000 out of 5000 | elapsed:    1.8s finished
[Parallel(n_jobs=40)]: Done 120 tasks      | elapsed:    0.1s
```

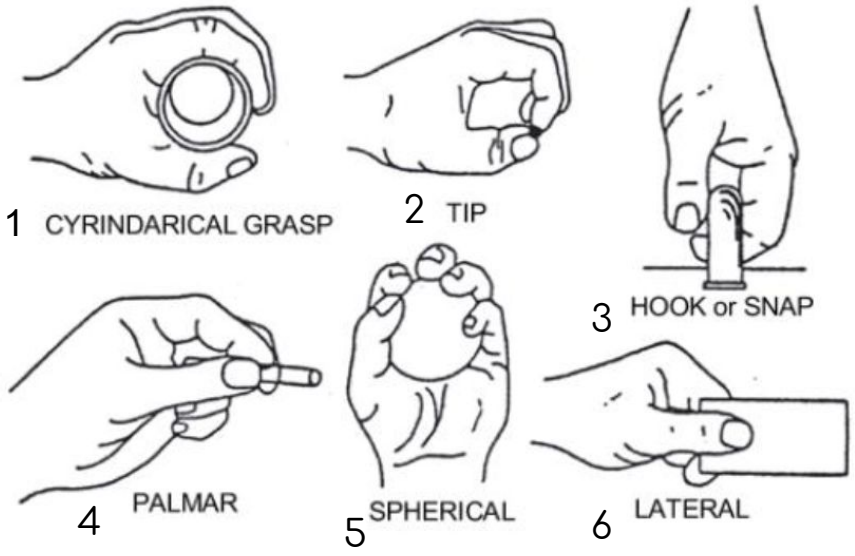
```
0.936111111111
```

Analysis Summary

Analysis

	Test Rate of Accuracy before EMD	After EMD
NN MATLAB	~ 23%	~ 84%
NN python	~ 30%	
AdaBoost	~ 60%	
Random Forest	~ 75%	~ 90%
Random Forest + PCA	~ 43%	~ 92%
Linear Discriminant Analysis		~ 65%

```
Out[17]: array([[66,  0,  0,  0,  0,  0],
                [ 2, 51,  2,  0,  0,  0],
                [ 1,  1, 61,  5,  0,  4],
                [ 0,  1,  1, 49,  0,  7],
                [ 0,  0,  0,  0, 51,  0],
                [ 0,  2,  0,  3,  0, 53]])
```



Discussion

What did we learn?

- Hands-on experience with running different machine learning algorithms
  - MATLAB NN Toolbox and EMD toolbox
  - AdaBoost, random forest
- Learned data processing and feature extraction steps
  - EMD, IMF
  - Zero Crossing, SSC
- Improved coding skills in MATLAB and python

Potential Improvements

- Choose several features from all eight popular features
- Try new feature function
- Cross Validation

Extensions

- Gesture recognition can be extended to more EMG functions

References

1. (n.d.). Retrieved December 07, 2017, from http://archive.ics.uci.edu/ml/datasets/sEMG for Basic Hand movements
2. Sapsanis, C., Georgoulas, G., Tzes, A., & Lymberopoulos, D. (2013). Improving EMG based classification of basic hand movements using EMD. 2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). doi:10.1109/embc.2013.6610858
3. http://perso.enslyon.fr/patrick.flandrin/emd.html

Special thanks to:

Professor Virginia de Sa
Sai Gullapally, Shuai Tang
Alex Li*

* UCSD Math PhD student specialized in machine learning