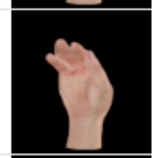
Machine Learning Final Project
Prosthetic Hand Control and Gesture Extraction

Daniel Pilgrim-Minaya

May 4, 2017

Label	Description	Instance	Label	Description	Instance
1	Index flexion		7	Little finger flexion	
2	Index extension		8	Little finger extension	
3	Middle flexion		9	Thumb adduction	
4	Middle extension		10	Thumb abduction	
5	Ring flexion		11	Thumb flexion	
6	Ring extension		12	Thumb extension	

Introduction

In prosthetic hand design, one of the first and foremost problems is the issue of human training and control. The human hand is anatomically perfect, and that is the standard to which prosthetics aspire to be. Currently in development are a class of prostheses controlled by the arm muscle's surface electrical activity. However, there is a huge issue of categorizing and grouping surface electromagnetic signals to classes of gestures.

In other studies, by recording the surface electromyogram (sEMG) of an arm with a matrix of electrodes, an image is extracted (each pixel corresponding to an electrode sample). This data can be used to train a classifier to be used for hand posture recognition, and in the past, they have done so using CNN (Convolutional Neural Networks) to pick up on hidden features. However, when developing a hand prosthetic, CNN's make for difficult versatility between subjects. Customization is necessary for each subject. For this reason, this report creates a training algorithm for a single subject's movements.

This report looks into presumably less computationally difficult machine learning algorithms for the classification of hand gestures. While CNN's have achieved a 92% accuracy in the past, this report looks at other classification algorithms to find if CNN's hold as the best classifiers for hand prosthetics.

The Data

Here is a description of the variables:

Variable Name:	Variable Description:
Gesture 1-12	Gestures were grouped from a single subject over a period of 10 trials for 10 seconds each.
Electrodes 1-128	Electrodes were placed on the skin at 128 different locations. The data does not mention at which location these electrodes were placed, but they are separate points of data. We took the average of every 5 milliseconds from the end of the first second to the start of the last second.
	Data is from http://zju-capg.org/myo/data/ (2018)

Processing Data

Much of the time was spent in this section of the project. The data was originally in 120 .mat files so they had to be converted to .csv files for use.

```
## In Matlab ##
>> matFiles = dir('*.mat');
>> numfiles = length(matFiles);
>> mydata = cell(1, numfiles);
>> for k = 1:numfiles;
load(matFiles(k).name);
csvwrite(strcat(matFiles(k).name,'.csv'),data);
end
```

After converting the .mat files to csv. We had to iterate through each .csv file to extract information about each gesture and trial for a single subject. We had to pass between different data structures for ease of iteration before we settled on the final .csv file.

This process took especially long because for the first machine learning trial, we used the complete data set, which was 72MB of data. In addition to having a large computational runtime, the complete data set actually was less effective at predictions for our first decision tree than taking the average of every five seconds of the complete set. The belief is this is because having the complete data set exposed the algorithm to lots of noise. By finding the average of 5 milliseconds in the middle of the trial (from the first second to the last second), we lowered our computational runtime, cut out noise, and had easier predictions.

```
#Balance between sample size and processing ability, also taking the middle 8 seconds for sample
gestureTrialNode = array(0, dim=c(12,10,128,100))
gestureNodeSamples = array(0, dim=c(12,128,1000))

for (i in 1:12){
  #for each trial
  for (j in 1:10){
    #read in data from person-gesture-trial.mat.csv
    gestureNum= ifelse(nchar(i) ==1, paste("00",i, sep=""), paste("0", i, sep=""))
    trialNum = ifelse(nchar(j) ==1, paste("00",j, sep=""), paste("0", j, sep=""))
    filename = paste("001-", gestureNum, "-", trialNum, ".mat.csv", sep="")
```

```
        #print(paste("reading",filename))
        hand_data = read.csv(filename)
        #for each node, sync data to nodesTrialsMatrix
        x = 1
        for (node in hand_data){
            #what to do with data of each node of each trial of each gesture
            #print(mean(node[100:900]))
            for (y in 1:100){
                gestureTrialNode[i,j,x,y] <- mean(node[((5*(y-1))+251):(5*(y)+25)])
            }
            x=x+1
        }    }}
##
for (x in 1:12){
    k=1
for (i in 1:10){
    for (j in 1:100){
        for (y in 1:128){
            gestureNodeSamples[x,y,k] <- gestureTrialNode[x,i,y,j]
        }  k=k+1}}}
A = matrix(0,nrow=12001,ncol=129)

k=2
for (x in 1:12){
    for (y in 1:1000){
        A[k,1] = paste("Gesture",x, sep="")
        k=k+1 }  }
for (x in 1:129){
    A[1,1] = "start"
    A[1,x] = paste("Node",x-1, sep="")
}

l = 2
for (i in 1:12){
    for (k in 1:1000){
        for (j in 1:128){
            A[l,j+1] <- gestureNodeSamples[i,j,k]
        }
        l = l+1 }}

A = A[2:12001,1:129]
hand_data = as.data.frame(A)
write.csv(hand_data, file = "hand_data1.csv")
hand_data$V2 <- as.numeric(as.character(hand_data$V2))
hand_data$V3 <- as.numeric(as.character(hand_data$V3))
.....ommited
hand_data$V127 <- as.numeric(as.character(hand_data$V127))
hand_data$V128 <- as.numeric(as.character(hand_data$V128))

str(hand_data)
'data.frame':        12000 obs. of  129 variables:
 $ V1  : Factor w/ 12 levels "Gesture1","Gesture10",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ V2  : num  -0.006494 -0.003207 -0.001007 0.000536 -0.004628 ...
 $ V3  : num  -0.00167 -0.00132 0.00272 0.00492 0.00238 ...
.....ommited
 $ V97 : num  0.006064 0.005659 -0.002489 0.004113 -0.000366 ...
 $ V98 : num  0.01032 0.00313 0.00271 -0.00245 0.00108 ...
 $ V99 : num  0.009631 -0.004172 0.005131 -0.000675 -0.002119 ...
  [list output truncated]
```

Modeling the Data:

This report tries 3 different classification methods in Machine Learning to classify hand gestures. We focused on the Recursive Partitioning Decision Tree, Random Forests, and Support Vector Machines for gesture classifications. The accuracy of Convolutional Neural Networks as 92% is the reference baseline.

Recursive Partitioning Decision Tree

With the standard Recursive Partitioning Decision tree, the goal was to show that the data could be used to make predictions and to demonstrate that the data was formatted correctly. Our first classification tree was extremely inaccurate with 9% accuracy because the entire data set was used, which was a case of not preparing the data correctly. With this decision tree, our accuracy was boosted to 18%, however this was still extremely inaccurate; for example, there was no classification branch for Gesture 9 at all.

```
### CLASSIFICATION TREE ###

require(rpart)
require(rpart.plot)
hand_data = read.csv("hand_data1.csv")
hand_data$X <- NULL
hand_data_tree = hand_data

#randomize entires
set.seed(99)
g <- runif(nrow(hand_data_tree))
hand_data_tree_shuffled <- hand_data_tree[order(g),]
> tree1 = rpart(V1 ~ ., data=hand_data_tree_shuffled[1:8000,], method="class")
> p1 <- predict(tree1, hand_data_tree_shuffled[8001:12000,], type="class")
> tree1
n= 100

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 100 89 Gesture1 (0.11 0.08 0.06 0.07 0.07 0.1 0.04 0.07 0.11 0.09 0.09 0.11)
    2) V53>=0.00338949 7  1 Gesture1 (0.86 0.14 0 0 0 0 0 0 0 0 0 0) *
    3) V53< 0.00338949 93 82 Gesture6 (0.054 0.075 0.065 0.075 0.075 0.11 0.043 0.075 0.12 0.097 0.097 0.12)
      6) V92< 0.002148788 83 73 Gesture3 (0.06 0.072 0.072 0.072 0.084 0.12 0.048 0.012 0.12 0.11 0.11 0.12)
       12) V106>=-0.002160144 16  8 Gesture3 (0 0 0.062 0 0.062 0.5 0 0.062 0.19 0.062 0.062 0) *
       13) V106< -0.002160144 67 57 Gesture9 (0.075 0.09 0.075 0.09 0.09 0.03 0.06 0 0.1 0.12 0.12 0.15)
         26) V94>=0.0004587256 8  4 Gesture10 (0 0.5 0 0 0.12 0 0 0 0.38 0 0 0) *
         27) V94< 0.0004587256 59 49 Gesture9 (0.085 0.034 0.085 0.1 0.085 0.034 0.068 0 0.068 0.14 0.14 0.17)
           54) V108>=0.0005565613 49 41 Gesture7 (0.082 0.02 0.1 0.1 0.1 0.041 0.061 0 0.082 0.16 0.16 0.082)
            108) V60< -0.001342616 9  5 Gesture2 (0.22 0 0.33 0 0.44 0 0 0 0 0 0 0) *
            109) V60>=-0.001342616 40 32 Gesture7 (0.05 0.025 0.05 0.12 0.025 0.05 0.075 0 0.1 0.2 0.2 0.1)
              218) V125< -0.0008264796 7  3 Gesture12 (0 0 0.29 0.57 0 0 0 0 0.14 0 0 0) *
              219) V125>=-0.0008264796 33 25 Gesture7 (0.061 0.03 0 0.03 0.03 0.061 0.091 0 0.091 0.24 0.24 0.12)
                438) V120< -0.000294041 9  3 Gesture7 (0.11 0.11 0 0 0 0 0 0 0 0.67 0.11 0) *
                439) V120>=-0.000294041 24 17 Gesture8 (0.042 0 0 0.042 0.042 0.083 0.12 0 0.12 0.083 0.29 0.17)
                  878) V18< -0.0002808394 7  4 Gesture6 (0 0 0 0 0 0.14 0.14 0 0.43 0.29 0 0) *
                  879) V18>=-0.0002808394 17 10 Gesture8 (0.059 0 0 0.059 0.059 0.059 0.12 0 0 0 0.41 0.24) *
           55) V108< 0.0005565613 10  4 Gesture9 (0.1 0.1 0 0.1 0 0 0.1 0 0 0 0 0.6) *
      7) V92>=0.002148788 10  4 Gesture5 (0 0.1 0 0.1 0 0 0 0.6 0.1 0 0 0.1) *

> table(hand_data_tree_shuffled[8001:12000,1], predicted =p1)
```

```
table(hand_data_tree_shuffled[8001:12000,1], predicted =p1)
          predicted
           Gesture1 Gesture10 Gesture11 Gesture12 Gesture2 Gesture3 Gesture4 Gesture5 Gesture6 Gesture7 Gesture8 Gesture9
Gesture1         65        25        82         1       92       45        0        4        4       21        5        0
Gesture10        10       153        69        13       14       10        0       11       15       44        3        0
Gesture11        44        24       172        17       18       26        2        1        4       35        6        0
Gesture12        17        66       122        51       15        9        0        0        8       21       10        0
Gesture2         37        14        43         0       92       23        2        2       13       79       24        0
Gesture3         61         4        77         3       50       98        0        0        8       18        8        0
Gesture4         46        47        61         1       37       57       29        9        0       27        6        0
Gesture5          9         6        20         0       10        3        0      281        2        3        3        0
Gesture6         37        14        81         0       37       52        0        1       73       28        6        0
Gesture7         24         6        73         0        6       26        0        1       32      167        6        0
Gesture8         13         3        74         1       17       29        0       14        1      120       54        0
Gesture9         43         7       176         0        8       43        1        2       11       39        7        0
```

_

Random Forests

Using the randomForest package in R, we achieved mediocre success with making a solid decision tree for gesture prediction. Given our OOB accuracy was 60%, this shows that this classification algorithm shows strength, however, it is still not up to prosthetics production standards. This tree, although a low bar, had one plotted category for each gesture, which was an improvement as well.

```
### Random Forest ##########################################################################
> hand_data_train = hand_data_shuffled[1:8000,]
> hand_data_test = hand_data_shuffled[8001:12000,]
> set.seed(99)
> rf <- randomForest(V1~.,data=hand_data_train)
> rf <- randomForest::randomForest(V1~.,data=hand_data_train)
> attributes(rf)
$names
 [1] "call"            "type"            "predicted"       "err.rate"
 [5] "confusion"       "votes"           "oob.times"       "classes"
 [9] "importance"      "importanceSD"    "localImportance" "proximity"
[13] "ntree"           "mtry"            "forest"          "y"
[17] "test"            "inbag"           "terms"

$class
[1] "randomForest.formula" "randomForest"

> print(rf)

Call:
 randomForest(formula = V1 ~ ., data = hand_data_train)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 11

        OOB estimate of  error rate: 30.29%
Confusion matrix:
```

```
Call:
 randomForest(formula = V1 ~ ., data = hand_data_train)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 11

        OOB estimate of  error rate: 30.29%
Confusion matrix:
          Gesture1 Gesture10 Gesture11 Gesture12 Gesture2 Gesture3 Gesture4 Gesture5 Gesture6 Gesture7 Gesture8 Gesture9 class.error
Gesture1       383        10        10        14       57      109       18       19       18        5        9       18  0.42835821
Gesture10        3       547        22        15        8        2        8       27       21       13        1        8  0.18962963
Gesture11       12        23       406        73       22       16       35        4       18       23        4       49  0.40729927
Gesture12       13        28        79       451       32        7       17        3       12       16        8       15  0.33773862
Gesture2        69        27         9         7      354       45       42        1       51       43       18        7  0.47399703
Gesture3        87         0         6         5       23      468        6        5       14        6        9       18  0.27666151
Gesture4        29        30        41         8       30       17      451       23        9        5        4       23  0.32686567
Gesture5         1         0         3         0        0        5        6      619        1        0        0        2  0.02825746
Gesture6        16        20        16         1       18       34       15        4      485       19        6       35  0.27503737
Gesture7         1        13        20         2       14        5        7        0       61      495       12       29  0.24886191
Gesture8         7         5         7         1        7       22       13        5        4       23      593       13  0.15285714
Gesture9        13         5        64         5       21       22       37        7       47       72       16      325  0.48738170
```

Support Vector Machine

Using the ggplot2 library, there was a lot of success with Support Vector Machines, which should be expected as Support Vector Machines are very similar to the perceptron and thus CNNs. With Support Vector Machines, the data was able to get an 82% accuracy, which is very good considering there was no tuning necessary. This is the best algorithm trialed so far.

```
> library(ggplot2)
> library(e1071)
> mymodel = svm(V1~., data=hand_data_train)
> summary(mymodel)
Call:
svm(formula = V1 ~ ., data = hand_data_train)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.0078125

Number of Support Vectors:  7210
 ( 661 523 614 639 657 658 586 604 654 603 614 397 )

Number of Classes:  12
Levels:
 Gesture1 Gesture10 Gesture11 Gesture12 Gesture2 Gesture3 Gesture4 Gesture5 Gesture6 Gesture7 Gesture8
Gesture9

> p2 <- predict(mymodel, hand_data_test
+ )
> tab1 <- table(Predicted=p2,Actual=hand_data$V1)
Error in table(Predicted = p2, Actual = hand_data$V1) :
  all arguments must have the same length
> tab1 <- table(Predicted=p2,Actual=hand_data_test$V1)
```

	Actual											
Predicted	Gesture1	Gesture10	Gesture11	Gesture12	Gesture2	Gesture3	Gesture4	Gesture5	Gesture6	Gesture7	Gesture8	Gesture9
Gesture1	256	1	0	2	32	21	8	4	4	0	0	1
Gesture10	5	268	6	8	4	0	7	1	7	6	0	5
Gesture11	1	13	257	44	7	1	15	4	3	4	3	8
Gesture12	2	12	32	236	1	0	2	2	0	2	0	5
Gesture2	28	1	3	4	250	16	12	10	6	2	1	2
Gesture3	20	1	2	2	11	297	4	6	10	0	2	3
Gesture4	7	2	5	7	9	1	268	16	1	1	6	3
Gesture5	0	3	0	0	0	0	2	310	0	0	0	0
Gesture6	4	16	1	2	6	4	2	4	274	18	2	9
Gesture7	1	5	3	2	4	4	5	1	6	294	6	12
Gesture8	0	0	1	5	3	6	3	1	7	12	277	9
Gesture9	6	3	5	7	0	3	2	4	13	2	3	309

```
> 1-sum(diag(tab1))/sum(tab1)
[1] 0.176
```

Conclusion

Neural Nets, KNN, and Random Forests were all fairly close in performance, and all better than Linear Regression. Our best OOB performance was from our Support Vector Machine with a 82% prediction accuracy, which still isn't as good as a Convolutional Nueral Network at 92% accuracy.