%reset  
%matplotlib inline  
import matplotlib.pyplot as plt  
from matplotlib import colors  
  
#IPython is what you are using now to run the notebook  
import IPython  
print ("IPython version: %6.6s (need at least 6.1.0)" % IPython.\_\_version\_\_)  
  
# Numpy is a library for working with Arrays  
import numpy as np  
print ("Numpy version: %6.6s (need at least 1.13.1)" % np.\_\_version\_\_)  
  
# SciPy implements many different numerical algorithms  
import scipy as sp  
print ("SciPy version: %6.6s (need at least 0.19.1)" % sp.\_\_version\_\_)  
  
# Pandas makes working with data tables easier  
import pandas as pd  
print ("Pandas version: %6.6s (need at least 0.20.3)" % pd.\_\_version\_\_)  
  
# SciKit Learn implements several Machine Learning algorithms  
import sklearn  
from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA  
from sklearn.discriminant\_analysis import QuadraticDiscriminantAnalysis as QDA  
print ("Scikit-Learn version: %6.6s (need at least 0.19.0)" % sklearn.\_\_version\_\_)  
  
from itertools import combinations as cmb

Once deleted, variables cannot be recovered. Proceed (y/[n])? y  
  
  
IPython version: 7.8.0 (need at least 6.1.0)  
Numpy version: 1.17.2 (need at least 1.13.1)  
SciPy version: 1.3.1 (need at least 0.19.1)  
Pandas version: 0.25.1 (need at least 0.20.3)  
Scikit-Learn version: 0.22.1 (need at least 0.19.0)

# Set font sizes  
SMALL\_SIZE = 8  
MEDIUM\_SIZE = 12  
BIGGER\_SIZE = 14  
plt.rc('font', size=SMALL\_SIZE) # controls default text sizes  
plt.rc('axes', titlesize=SMALL\_SIZE) # fontsize of the axes title  
plt.rc('axes', labelsize=MEDIUM\_SIZE) # fontsize of the x and y labels  
plt.rc('xtick', labelsize=SMALL\_SIZE) # fontsize of the tick labels  
plt.rc('ytick', labelsize=SMALL\_SIZE) # fontsize of the tick labels  
plt.rc('legend', fontsize=SMALL\_SIZE) # legend fontsize  
plt.rc('figure', titlesize=BIGGER\_SIZE) # fontsize of the figure title

# import data  
relaxed = np.loadtxt('./Relaxed.csv',delimiter=';',usecols=range(8))  
neutral = np.loadtxt('./Neutral.csv',delimiter=';',usecols=range(8))  
closed = np.loadtxt('./Closed.csv',delimiter=';',usecols=range(8))  
openh = np.loadtxt('./Open.csv', delimiter=';',usecols=range(8))  
flexion = np.loadtxt('./Flexion.csv',delimiter=';',usecols=range(8))  
extension = np.loadtxt('./Extension.csv',delimiter=';',usecols=range(8))  
ulnar = np.loadtxt('./Ulnar.csv',delimiter=';',usecols=range(8))  
radial = np.loadtxt('./Radial.csv',delimiter=';',usecols=range(8))  
pronation = np.loadtxt('./Pronation.csv',delimiter=';',usecols=range(8))  
supination = np.loadtxt('./Supination.csv',delimiter=';',usecols=range(8))  
thumbsup = np.loadtxt('./Thumbsup.csv', delimiter=';',usecols=range(8))  
thumbsdown = np.loadtxt('./Thumbsdown.csv', delimiter=';',usecols=range(8))  
thumbflat = np.loadtxt('./ThumbFlat.csv',delimiter=';',usecols=range(8))  
indexpinch = np.loadtxt('./IndexPinch.csv', delimiter=';',usecols=range(8))  
middlepinch = np.loadtxt('./MiddlePinch.csv', delimiter=';',usecols=range(8))  
ringpinch = np.loadtxt('./RingPinch.csv', delimiter=';',usecols=range(8))  
pinkypinch = np.loadtxt('./PinkyPinch.csv', delimiter=';',usecols=range(8))  
  
flexionfist = np.loadtxt('./FlexionFist.csv',delimiter=';',usecols=range(8))  
extensionfist = np.loadtxt('./ExtensionFist.csv',delimiter=';',usecols=range(8))  
ulnarfist = np.loadtxt('./UlnarFist.csv',delimiter=';',usecols=range(8))  
radialfist = np.loadtxt('./RadialFist.csv', delimiter=';',usecols=range(8))  
pronationfist = np.loadtxt('./PronationFist.csv',delimiter=';',usecols=range(8))  
supinationfist = np.loadtxt('./SupinationFist.csv',delimiter=';',usecols=range(8))  
indexpoint = np.loadtxt('./IndexPoint.csv',delimiter=';',usecols=range(8))  
middlepoint = np.loadtxt('./MiddlePoint.csv',delimiter=';',usecols=range(8))  
ringpoint = np.loadtxt('./RingPoint.csv',delimiter=';',usecols=range(8))  
pinkypoint = np.loadtxt('./PinkyPoint.csv', delimiter=';',usecols=range(8))  
fingerspread = np.loadtxt('./FingerSpread.csv', delimiter=';',usecols=range(8))  
fingergun = np.loadtxt('./FingerGun.csv', delimiter=';',usecols=range(8))  
peace = np.loadtxt('./Peace.csv', delimiter=';',usecols=range(8))  
spiderman = np.loadtxt('./Spiderman.csv', delimiter=';',usecols=range(8))  
spock = np.loadtxt('./Spock.csv', delimiter=';',usecols=range(8))  
  
# Combine data into navigable structure  
full\_dset = [relaxed, neutral, closed, openh,   
 flexion, extension, ulnar, radial,   
 pronation, supination, thumbsup, thumbsdown,   
 thumbflat, indexpinch, middlepinch, ringpinch,   
 pinkypinch, flexionfist, extensionfist, ulnarfist,   
 radialfist, pronationfist, supinationfist, indexpoint,   
 middlepoint, ringpoint, pinkypoint, fingerspread,   
 fingergun, peace, spiderman, spock]  
  
labels = ["relaxed","neutral","closed","open",  
 "flexion","extension","ulnar","radial",  
 "pronation","supination","thumbs up","thumbs down",  
 "thumb flat","index pinch","middle pinch","ring pinch",  
 "pinky pinch","flexion fist","extension fist","ulnar fist",  
 "radial fist","pronation fist","supination fist","index point",  
 "middle point","ring point","pinky point","finger spread",  
 "finger gun","peace","spiderman","spock"]  
  
n\_classes = len(full\_dset)

# Calculates RMS's of 250ms chunks (for all channels, single class)  
def rms(data):  
 output = np.zeros((8,120))  
 for i in range(8):  
 for j in range(120):  
 low = j \* 50  
 high = low + 50  
 output[i,j] = np.sqrt(np.mean(data[low:high,i]\*\*2))  
 return output

# Perform RMS calculation  
rms\_t = np.empty((8,0))  
rms\_v = np.empty((8,0))  
for i,dset in enumerate(full\_dset):  
 rmss = rms(dset)  
 rms\_t = np.append(rms\_t,rmss[:,:int(len(rmss[0])\*0.25)],axis=1)  
 rms\_v = np.append(rms\_v,rmss[:,int(len(rmss[0])\*0.25):],axis=1)

def lda\_format(rms\_set):  
 # Reformat data for LDA classifier  
 classes = np.empty((1,0))  
 n\_samples = int(len(rms\_set[0])/n\_classes)  
 for i in range(n\_classes):  
 classes = np.append(classes,np.full((1,n\_samples),int(i)),axis=1)  
 zipped = np.concatenate((rms\_set,classes),axis=0)   
 zipped = np.swapaxes(zipped,0,1)  
 np.random.shuffle(zipped)  
 data = zipped[:,:-1]  
 classes = np.ravel(zipped[:,-1:])  
 return(data,classes)

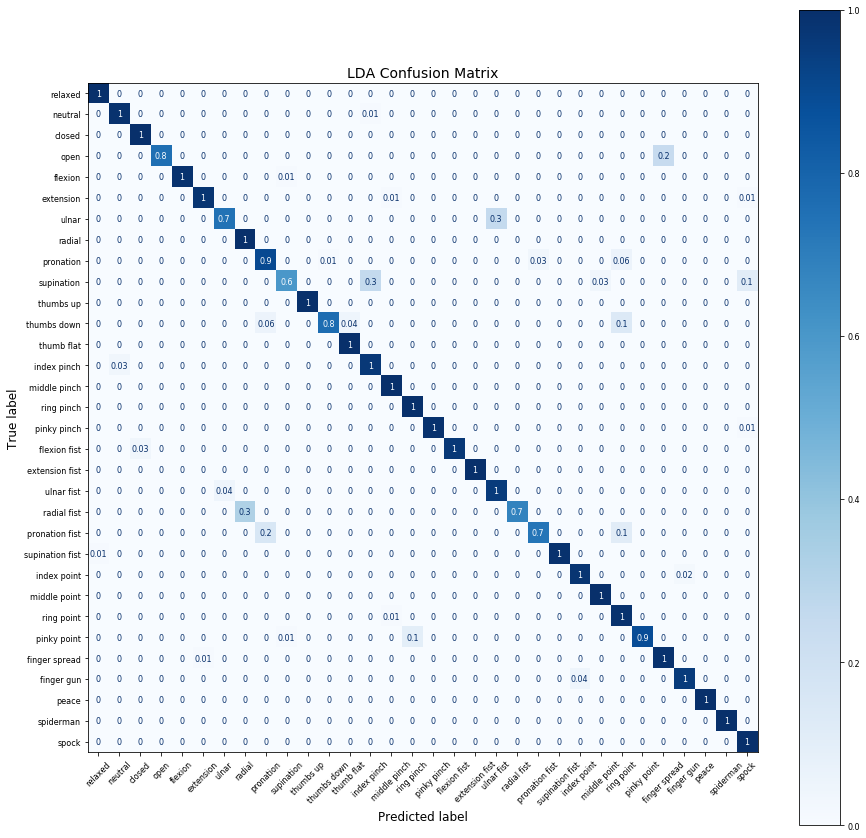
# Format Data  
data\_t,classes\_t = lda\_format(rms\_t)  
data\_v,classes\_v = lda\_format(rms\_v)  
  
# Perform LDA fit  
lda = LDA()  
lda.fit(data\_t, classes\_t)  
  
# Perform QDA fit  
qda = QDA()  
qda.fit(data\_t, classes\_t)

QuadraticDiscriminantAnalysis(priors=None, reg\_param=0.0,  
 store\_covariance=False, tol=0.0001)

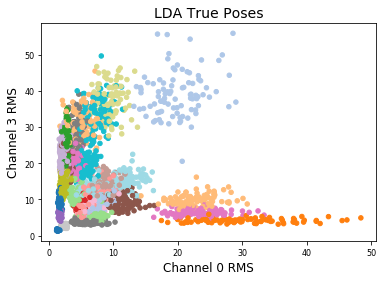
# Function for evaluating LDA success  
def lda\_eval(lda, v\_data, v\_classes):  
 pred = lda.predict(v\_data)  
 num\_correct = 0  
 misses = []  
 for i,p in enumerate(pred):  
 if v\_classes[i] == p:  
 num\_correct += 1  
 else:  
 misses.append((v\_classes[i],p))  
 pos\_rate = num\_correct/len(v\_classes)  
 return (pred,pos\_rate,misses)  
   
pred,rate,misses = lda\_eval(lda, data\_v, classes\_v)  
print('True Positive Rate: '+str(rate))

True Positive Rate: 0.93125

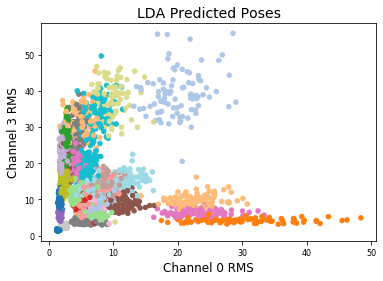
fig, a = plt.subplots(1, 1)  
fig.set\_figheight(15)  
fig.set\_figwidth(15)  
plt.title("LDA Confusion Matrix",fontdict={'fontsize': 14})  
sklearn.metrics.plot\_confusion\_matrix(lda,data\_v,classes\_v,  
 xticks\_rotation=45,  
 values\_format='0.1g',  
 cmap=plt.cm.Blues,  
 normalize='true',  
 ax=a,  
 display\_labels=labels)  
  
plt.savefig('32-Pose LDA Confusion Matrix.pdf',bbox\_inches='tight')



plt.title("LDA True Poses",fontdict={'fontsize': 14})  
plt.xlabel("Channel 0 RMS",fontdict={'fontsize': 12})  
plt.ylabel("Channel 3 RMS",fontdict={'fontsize': 12})  
plt.scatter(data\_v[:,0], data\_v[:,3], c=classes\_v, s=20, cmap='tab20')  
plt.savefig("32-Pose LDA True Classes Scatter.pdf")



plt.title("LDA Predicted Poses",fontdict={'fontsize': 14})  
plt.xlabel("Channel 0 RMS",fontdict={'fontsize': 12})  
plt.ylabel("Channel 3 RMS",fontdict={'fontsize': 12})  
plt.scatter(data\_v[:,0], data\_v[:,3], c=pred, s=20, cmap='tab20')  
plt.savefig("32-Pose LDA Predicted Classes Scatter.pdf")



# Function for evaluating QDA success  
def qda\_eval(qda, v\_data, v\_classes):  
 pred = qda.predict(v\_data)  
 num\_correct = 0  
 misses = []  
 for i,p in enumerate(pred):  
 if v\_classes[i] == p:  
 num\_correct += 1  
 else:  
 misses.append((v\_classes[i],p))  
 pos\_rate = num\_correct/len(v\_classes)  
 return (pred,pos\_rate,misses)  
   
pred,rate,misses = qda\_eval(qda, data\_v, classes\_v)  
print('True Positive Rate: '+str(rate))

True Positive Rate: 0.9708333333333333

fig, a = plt.subplots(1, 1)  
fig.set\_figheight(15)  
fig.set\_figwidth(15)  
plt.title("QDA Confusion Matrix",fontdict={'fontsize': 30})  
sklearn.metrics.plot\_confusion\_matrix(qda,data\_v,classes\_v,  
 xticks\_rotation=45,  
 values\_format='0.1g',  
 cmap=plt.cm.Blues,  
 normalize='true',  
 ax=a,  
 display\_labels=labels)

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x1a215cbf10>

