In [49]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import plotly.plotly as py
from plotly.graph_objs import *
import plotly.tools as tls
from plotly import tools
import plotly.graph_objs as go
import plotly
plotly.tools.set_credentials_file(username='rrazaghi', api_key='xnbxzag0vc'
)

from sklearn.preprocessing import StandardScaler
%matplotlib notebook
```

Loading the Dataset

In [50]:

```
df_good1 = pd.read_csv('./goodfit1.csv')
df_good2 = pd.read_csv('./goodfit2.csv')
df_good3 = pd.read_csv('./goodfit3.csv')
df_poor1 = pd.read_csv('./poorfit1.csv')
df_poor2 = pd.read_csv('./poorfit2.csv')
df_poor3 = pd.read_csv('./poorfit3.csv')
```

In [51]:

```
# df_poor3.head()
```

In [52]:

```
# extracting the dataset into a matrix containing the temperature data and
an array containing the class information
X_g1 = df_good1.ix[:,1:9].values
y_g1 = df_good1.ix[:,9].values
X_g2 = df_good2.ix[:,1:9].values
y_g2 = df_good2.ix[:,9].values
X_g3 = df_good3.ix[:,1:9].values
y_g3 = df_good3.ix[:,9].values
X_p1 = df_poor1.ix[:,1:9].values
y_p1 = df_poor1.ix[:,9].values
X_p2 = df_poor2.ix[:,1:9].values
y_p2 = df_poor2.ix[:,9].values
X_p3 = df_poor3.ix[:,1:9].values
y_p3 = df_poor3.ix[:,9].values
```

Standardizing

In [53]:

```
# Here we standardize the data points before feeding them into the PCA
```

```
# Here we standardize the data points before feeding them into the PCA algorithm:
Xg1_std = StandardScaler().fit_transform(X_g1)
Xg2_std = StandardScaler().fit_transform(X_g2)
Xg3_std = StandardScaler().fit_transform(X_g3)
Xp1_std = StandardScaler().fit_transform(X_p1)
Xp2_std = StandardScaler().fit_transform(X_p2)
Xp3_std = StandardScaler().fit_transform(X_p3)
```

Eigendecomposition - Computing Eigenvectors and Eigenvalues

Covariance Matrix, Eigenvectors, and Eigenvalues:

In [54]:

```
def eigendecomp(X_std):

    cov_mat = np.cov(X_std.T)
    print('Covariance matrix \n%s' %cov_mat)

    eig_vals, eig_vecs = np.linalg.eig(cov_mat)

    print('Eigenvectors \n%s' %eig_vecs)
    print('\nEigenvalues \n%s' %eig_vals)
    return eig_vals, eig_vecs
eig_vals_g1, eig_vecs_g1 = eigendecomp(Xg1_std)
eig_vals_g2, eig_vecs_g2 = eigendecomp(Xg2_std)
eig_vals_g3, eig_vecs_g3 = eigendecomp(Xg3_std)
eig_vals_p1, eig_vecs_p1 = eigendecomp(Xp1_std)
eig_vals_p2, eig_vecs_p2 = eigendecomp(Xp2_std)
eig_vals_p3, eig_vecs_p3 = eigendecomp(Xp3_std)
```

```
Covariance matrix
[[ 1.00011461  0.92288055  0.9099399   0.46749589  0.79953475  0.88415634
   0.88860384  0.90643156]
 [ 0.92288055  1.00011461  0.93810683  0.44739292  0.88438843  0.94931778
   0.90474184  0.9154363 ]
 [ 0.9099399   0.93810683  1.00011461  0.67331851  0.86723337  0.8890719
   0.91513549  0.91903443]
 [ 0.46749589  0.44739292  0.67331851  1.00011461  0.44507371  0.35704125
   0.54316992  0.51948415]
 [ 0.79953475  0.88438843  0.86723337  0.44507371  1.00011461  0.96104961
   0.96021737  0.95669224]
 [ 0.88415634  0.94931778  0.8890719   0.35704125  0.96104961  1.00011461
   0.95745196  0.96620253]
 [ 0.88860384  0.90474184  0.91513549  0.54316992  0.96021737  0.95745196
   1.00011461  0.99595169]
 [ 0.90643156  0.9154363   0.91903443  0.51948415  0.95669224  0.96620253
   0.99595169  1.00011461]]
Eigenvectors
[[-0.35680576  0.06776584 -0.63190817 -0.5112935  -0.10590913 -0.4198527
  -0.07932012  0.11654933]
 [-0.36725946  0.13113332 -0.33765047  0.56206169  0.56039112 -0.08597871
   0.11174543 -0.28997547]
 [-0.3704844  -0.17890671 -0.24094231  0.34419068 -0.70468668  0.3707283
  -0.01455084 -0.14134624]
 [-0.21946968 -0.91766116  0.10336529  0.00620782  0.21351001 -0.12077217
```

```
  -0.01764639  0.19627609]
 [-0.36277887  0.15240607  0.54139458  0.17665706 -0.27796623 -0.65908116
   0.04872352 -0.08258874]
 [-0.36948055  0.27395899  0.1234364   0.13929613  0.12187492  0.23714681
  -0.24260208  0.78983718]
 [-0.376377    0.03530754  0.26952417 -0.34915737  0.17901912  0.28224705
  -0.58235591 -0.46009775]
 [-0.37762479  0.06763484  0.19224654 -0.36281932  0.09169864  0.30662029
   0.76179394 -0.03839029]]

Eigenvalues
[  6.80756089e+00   7.93233257e-01   2.51227036e-01   1.00278578e-01
   2.30974625e-02   1.64729942e-02   2.56156089e-03   6.48512624e-03]
Covariance matrix
[[ 1.00011145  0.92160115  0.77945108  0.60263232  0.82721434  0.81047984
   0.76432875  0.76934198]
 [ 0.92160115  1.00011145  0.85748831  0.71826742  0.72444826  0.74022973
   0.70229539  0.70765767]
 [ 0.77945108  0.85748831  1.00011145  0.8918644   0.80622778  0.89133948
   0.89900263  0.89809694]
 [ 0.60263232  0.71826742  0.8918644   1.00011145  0.75491067  0.79984932
   0.79904427  0.7984104 ]
 [ 0.82721434  0.72444826  0.80622778  0.75491067  1.00011145  0.96828256
   0.92111052  0.91974796]
 [ 0.81047984  0.74022973  0.89133948  0.79984932  0.96828256  1.00011145
   0.98447588  0.97818044]
 [ 0.76432875  0.70229539  0.89900263  0.79904427  0.92111052  0.98447588
   1.00011145  0.99421719]
 [ 0.76934198  0.70765767  0.89809694  0.7984104   0.91974796  0.97818044
   0.99421719  1.00011145]]
Eigenvectors
[[ 0.33465973 -0.55743118 -0.33287131  0.07633647  0.47070208 -0.4881561
  -0.02797852 -0.0017764 ]
 [ 0.3284111  -0.6454759   0.20282485 -0.04751803 -0.2067174   0.61700129
   0.0086509  -0.09280952]
 [ 0.36400356 -0.02400763  0.37489131 -0.45440036 -0.45285183 -0.49330773
   0.10501602  0.24585811]
 [ 0.32962041  0.1981683   0.69817752  0.47259519  0.36160003 -0.0508942
  -0.04632672 -0.07609645]
 [ 0.35977163  0.13596449 -0.35438827  0.60564987 -0.39921969  0.02369258
   0.3132248   0.31879108]
 [ 0.3730465   0.21213162 -0.21584255  0.02962342 -0.27660338 -0.06151365
  -0.66417988 -0.49693326]
 [ 0.36769203  0.29777177 -0.15531507 -0.30868142  0.19881461  0.12844203
   0.6055617  -0.48373336]
 [ 0.36770105  0.28395441 -0.154148   -0.31482591  0.35279606  0.33797584
  -0.28287483  0.58530161]]

Eigenvalues
[  6.82366032e+00   5.76216477e-01   4.15855608e-01   1.31512020e-01
   3.00368021e-02   1.39851154e-02   1.84793471e-03   7.77729090e-03]
Covariance matrix
[[ 1.00011412  0.77629709  0.76063714 -0.28703689  0.47373677  0.72395466
   0.77234738  0.80292766]
 [ 0.77629709  1.00011412  0.94094437  0.09588727  0.79746933  0.9324266
   0.92318662  0.95562289]
 [ 0.76063714  0.94094437  1.00011412  0.24437218  0.8350514   0.91516597
   0.844773    0.89335154]
 [-0.28703689  0.09588727  0.24437218  1.00011412  0.58406621  0.2284173
  -0.01559173  0.01262444]
```

```
 [ 0.47373677  0.79746933  0.8350514   0.58406621  1.00011412  0.91221286
   0.7780155   0.80635004]
 [ 0.72395466  0.9324266   0.91516597  0.2284173   0.91221286  1.00011412
   0.95041312  0.96988914]
 [ 0.77234738  0.92318662  0.844773   -0.01559173  0.7780155   0.95041312
   1.00011412  0.98730746]
 [ 0.80292766  0.95562289  0.89335154  0.01262444  0.80635004  0.96988914
   0.98730746  1.00011412]]
Eigenvectors
[[-0.32235104 -0.39644566 -0.63971807  0.52241949 -0.22860528 -0.06672442
  -0.00617355 -0.00294689]
 [-0.39198896 -0.05056725 -0.03231305 -0.58762271 -0.66484957  0.16389943
   0.16168361  0.04912697]
 [-0.38474173  0.06190975 -0.44223767 -0.47309481  0.62877328 -0.12457038
   0.03018212 -0.13024418]
 [-0.06547948  0.81004769 -0.25253092  0.11316786 -0.23772777 -0.36355971
  -0.20732198  0.17687224]
 [-0.35318457  0.38397224  0.1284213   0.30938864  0.03784832  0.48846431
   0.24507761 -0.56171141]
 [-0.39836449  0.06916991  0.1996061   0.18803987  0.22710779  0.23794728
   0.23704672  0.77261353]
 [-0.38657649 -0.12657426  0.44754818  0.11729235 -0.01479152 -0.71827883
   0.26314109 -0.1873345 ]
 [-0.39654219 -0.10965179  0.27154492  0.01771848  0.02480126  0.09925033
  -0.86280329 -0.04224848]]

Eigenvalues
[  6.12134492e+00   1.45037011e+00   2.49696009e-01   1.17703653e-01
   3.24222311e-02   1.94618341e-02   1.94210418e-03   7.97207064e-03]
Covariance matrix
[[ 1.00012412  0.89813692  0.84669064  0.69225658  0.84812707  0.83672024
   0.78147427  0.8069366 ]
 [ 0.89813692  1.00012412  0.80686275  0.83677719  0.78278761  0.71136067
   0.62956354  0.6599407 ]
 [ 0.84669064  0.80686275  1.00012412  0.83840829  0.65070368  0.62582958
   0.62238624  0.65872356]
 [ 0.69225658  0.83677719  0.83840829  1.00012412  0.50873715  0.36982983
   0.31380483  0.36004779]
 [ 0.84812707  0.78278761  0.65070368  0.50873715  1.00012412  0.96344266
   0.92760359  0.94025021]
 [ 0.83672024  0.71136067  0.62582958  0.36982983  0.96344266  1.00012412
   0.97885792  0.98272158]
 [ 0.78147427  0.62956354  0.62238624  0.31380483  0.92760359  0.97885792
   1.00012412  0.99414726]
 [ 0.8069366   0.6599407   0.65872356  0.36004779  0.94025021  0.98272158
   0.99414726  1.00012412]]
Eigenvectors
[[-0.37923197 -0.11556752  0.05432172  0.7354615   0.49070279 -0.23346205
  -0.05353013  0.02852303]
 [-0.35419283 -0.30853129  0.53157205  0.18156338 -0.65106035  0.09943707
  -0.16614456  0.0495021 ]
 [-0.33711151 -0.35417185 -0.71541302  0.05548631 -0.13262624  0.46031622
  -0.08537939 -0.09619369]
 [-0.26878352 -0.62522845  0.06295434 -0.50585878  0.22921652 -0.40594639
   0.24409004  0.00870796]
 [-0.37797139  0.19421477  0.3334354  -0.36121592  0.44855801  0.47750248
  -0.37767324 -0.07689055]
 [-0.37134045  0.31398243  0.09139421  0.02224164 -0.05826051  0.20561794
   0.82118556 -0.18615845]
 [-0.35925774  0.36445256 -0.19680481 -0.12772336 -0.21812822 -0.50376322
```

```
  -0.28950892 -0.5457931 ]
 [-0.36745011  0.3255369  -0.20108968 -0.14097658 -0.11411734 -0.18670703
  -0.04315925  0.80557413]]

Eigenvalues
[  6.26905082e+00   1.30179751e+00   2.35678220e-01   1.27944944e-01
   4.39064081e-02   1.18554702e-02   6.67293967e-03   4.08661662e-03]
Covariance matrix
[[ 1.00011325  0.93309456  0.91363508  0.56741934  0.85179347  0.90874209
   0.89305525  0.91901127]
 [ 0.93309456  1.00011325  0.89584461  0.80057121  0.85440783  0.84190988
   0.8046623   0.82622381]
 [ 0.91363508  0.89584461  1.00011325  0.5442258   0.67432183  0.73584306
   0.71683235  0.77510681]
 [ 0.56741934  0.80057121  0.5442258   1.00011325  0.66329706  0.51329223
   0.45099601  0.44361927]
 [ 0.85179347  0.85440783  0.67432183  0.66329706  1.00011325  0.95529258
   0.9441022   0.93315305]
 [ 0.90874209  0.84190988  0.73584306  0.51329223  0.95529258  1.00011325
   0.98944515  0.98798153]
 [ 0.89305525  0.8046623   0.71683235  0.45099601  0.9441022   0.98944515
   1.00011325  0.99076983]
 [ 0.91901127  0.82622381  0.77510681  0.44361927  0.93315305  0.98798153
   0.99076983  1.00011325]]
Eigenvectors
[[ 0.37435045  0.03355831  0.30788347  0.71176339 -0.42296699  0.05205466
   0.27160188 -0.04424798]
 [ 0.36877539 -0.30831891  0.14745479  0.21935361  0.26213057 -0.00125363
  -0.77761896  0.16024976]
 [ 0.33361083 -0.14016777  0.70674689 -0.54734704 -0.02917441 -0.19139058
   0.17480199 -0.04404116]
 [ 0.25738871 -0.78361221 -0.35728547 -0.03940645  0.10945052  0.1554492
   0.39273363  0.01065824]
 [ 0.36775934  0.05663681 -0.43077938 -0.29489361 -0.6668925  -0.25860588
  -0.25849626 -0.10280709]
 [ 0.37314462  0.24792112 -0.18826374  0.11072369  0.49100842 -0.37331279
   0.117108   -0.59785387]
 [ 0.36635789  0.32615214 -0.1812552  -0.02572949  0.22285683 -0.19055581
   0.23575589  0.76441723]
 [ 0.37097085  0.31475443 -0.04952994 -0.2103598   0.06368585  0.8330324
  -0.02510556 -0.13407707]]

Eigenvalues
[  6.65096696e+00   8.11914375e-01   4.64714580e-01   3.82376503e-02
   1.72020276e-02   1.92062865e-03   8.45646484e-03   7.49331503e-03]
Covariance matrix
[[ 1.00011613  0.91436206  0.48894953  0.58422568  0.81594894  0.79875644
   0.78135576  0.78879904]
 [ 0.91436206  1.00011613  0.68999284  0.78502914  0.81302501  0.73648583
   0.71675548  0.75973269]
 [ 0.48894953  0.68999284  1.00011613  0.57207324  0.46111312  0.41070205
   0.3836344   0.47444195]
 [ 0.58422568  0.78502914  0.57207324  1.00011613  0.43135659  0.26516284
   0.23288969  0.30257657]
 [ 0.81594894  0.81302501  0.46111312  0.43135659  1.00011613  0.96868664
   0.96246321  0.97322544]
 [ 0.79875644  0.73648583  0.41070205  0.26516284  0.96868664  1.00011613
   0.99336137  0.99193972]
 [ 0.78135576  0.71675548  0.3836344   0.23288969  0.96246321  0.99336137
   1.00011613  0.99149897]
```

```
 [ 0.78879904  0.75973269  0.47444195  0.30257657  0.97322544  0.99193972
   0.99149897  1.00011613]]
Eigenvectors
[[ 0.37359887  0.08203477  0.34174446  0.74777344  0.38483942 -0.09669846
  -0.12859495 -0.06124664]
 [ 0.37987726  0.29239673  0.14852391  0.16439611 -0.82918022 -0.0213772
   0.18152835 -0.0127523 ]
 [ 0.2534204   0.45213707 -0.82861471  0.12035873  0.14469695 -0.06095756
  -0.03109089  0.06813169]
 [ 0.2315996   0.64274585  0.39036174 -0.48610243  0.2907857   0.23552603
  -0.03930642  0.05465987]
 [ 0.39531521 -0.17136657  0.04849086 -0.35552366  0.0590262  -0.81995768
  -0.09719232 -0.01870595]
 [ 0.3838053  -0.29824489 -0.0468243  -0.07094837  0.21533523  0.19189161
   0.79908861  0.18650211]
 [ 0.37857005 -0.32711265 -0.04537384 -0.07957745 -0.09462515  0.3221676
  -0.50763753  0.60897368]
 [ 0.38905137 -0.25476475 -0.1247561  -0.15894553 -0.00761438  0.34359736
  -0.20570453 -0.76320031]]

Eigenvalues
[  5.92081297e+00   1.35772516e+00   4.93940997e-01   1.85510282e-01
   1.96648430e-02   1.52346090e-02   5.63620230e-03   2.40398421e-03]
```

Selecting Principal Components

The typical goal of a PCA is to reduce the dimensionality of the original feature space by projecting it onto a smaller subspace, where the eigenvectors will form the axes. However, the eigenvectors only define the directions of the new axis, since they have all the same unit length 1, which can confirmed by the following two lines of code:

In [55]:

```
for ev in eig_vecs_g1:
    np.testing.assert_array_almost_equal(1.0, np.linalg.norm(ev))
print('Everything ok!')
```

```
Everything ok!
```

In [56]:

```
def eigensort(eig_vals, eig_vecs):
    # Make a list of (eigenvalue, eigenvector) tuples
    eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig
_vals))]

    # Sort the (eigenvalue, eigenvector) tuples from high to low
    eig_pairs.sort()
    eig_pairs.reverse()

    # Visually confirm that the list is correctly sorted by decreasing eige
nvalues
    print('Eigenvalues in descending order:')
    for i in eig_pairs:
        print(i[0])
    return eig_pairs
eig_pairs_g1 = eigensort(eig_vals_g1, eig_vecs_g1)
```

```
eig_pairs_g2 = eigensort(eig_vals_g2, eig_vecs_g2)
eig_pairs_g3 = eigensort(eig_vals_g3, eig_vecs_g3)
eig_pairs_p1 = eigensort(eig_vals_p1, eig_vecs_p1)
eig_pairs_p2 = eigensort(eig_vals_p2, eig_vecs_p2)
eig_pairs_p3 = eigensort(eig_vals_p3, eig_vecs_p3)
```

```
Eigenvalues in descending order:
6.80756089031
0.793233257385
0.251227036391
0.100278577595
0.0230974624518
0.0164729941817
0.0064851262427
0.00256156088798
Eigenvalues in descending order:
6.82366031578
0.57621647684
0.415855607999
0.131512019892
0.0300368020564
0.0139851154078
0.00777729089578
0.00184793471348
Eigenvalues in descending order:
6.12134491692
1.4503701102
0.249696009173
0.11770365302
0.0324222311005
0.0194618341273
0.00797207064392
0.00194210418315
Eigenvalues in descending order:
6.26905081885
1.30179750754
0.235678220125
0.127944944356
0.0439064080655
0.0118554701743
0.0066729396745
0.00408661662431
Eigenvalues in descending order:
6.65096696129
0.8119143746
0.464714579893
0.0382376503121
0.0172020276411
0.00845646484439
0.00749331502672
0.00192062865252
Eigenvalues in descending order:
5.92081297072
1.35772515649
0.493940996562
0.185510282048
0.0196648429544
0.0152346089591
0.00563620230104
0.00240398421062
```

Explained Variance

After sorting the eigenpairs, the next question is "how many principal components are we going to choose for our new feature subspace?" A useful measure is the so-called "explained variance," which can be calculated from the eigenvalues. The explained variance tells us how much information (variance) can be attributed to each of the principal components.

In [57]:

```
def explainedVar(eig_vals):
    tot = sum(eig_vals_g1)
    var_exp = [(i / tot)*100 for i in sorted(eig_vals_g1, reverse=True)]
    return var_exp
varexp_g1 = explainedVar(eig_vals_g1); varexp_g2 = explainedVar(eig_vals_g2
);varexp_g3 = explainedVar(eig_vals_g3);
varexp_p1 = explainedVar(eig_vals_p1); varexp_p2 = explainedVar(eig_vals_p2
);varexp_p3 = explainedVar(eig_vals_p3);
# cum_var_exp = np.cumsum(var_exp)

trace1 = go.Bar(
        x=['PC %s' %i for i in range(1,9)],
        y=varexp_g1,
        showlegend=True)
trace2 = go.Bar(
        x=['PC %s' %i for i in range(1,9)],
        y=varexp_g2,
        showlegend=False)
trace3 = go.Bar(
        x=['PC %s' %i for i in range(1,9)],
        y=varexp_g3,
        showlegend=False)
trace4 = go.Bar(
        x=['PC %s' %i for i in range(1,9)],
        y=varexp_p1,
        showlegend=False)
trace5 = go.Bar(
        x=['PC %s' %i for i in range(1,9)],
        y=varexp_p2,
        showlegend=False)
trace6 = Bar(
        x=['PC %s' %i for i in range(1,9)],
        y=varexp_p3,
        showlegend=False)

# trace2 = Scatter(
#          x=['PC %s' %i for i in range(1,9)],
#          y=cum_var_exp,
#          name='cumulative explained variance')

# data = Data([trace1, trace2])

# layout=Layout(
#          yaxis=YAxis(title='Explained variance in percent'),
#          title='Explained variance by different principal components for t
he good fit trial 1')
```

```
fig = tools.make_subplots(rows=3, cols=2,
                          subplot_titles=('good fit 1','poor fit 1','good fi
t 2', 'poor fit 2','good fit 3', 'poor fit 3'),
                          specs=[[{}, {}], [{}, {}], [{}, {}]],
                          horizontal_spacing = 0.1, vertical_spacing = 0.15)
fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 2, 1)
fig.append_trace(trace3, 3, 1)
fig.append_trace(trace4, 1, 2)
fig.append_trace(trace5, 2, 2)
fig.append_trace(trace6, 3, 2)

fig['layout']['yaxis1'].update(title='EV (%)')
fig['layout']['yaxis2'].update(title='EV (%)')
fig['layout']['yaxis3'].update(title='EV (%)')
fig['layout']['yaxis4'].update(title='EV (%)')
fig['layout']['yaxis5'].update(title='EV (%)')
fig['layout']['yaxis6'].update(title='EV (%)')
fig['layout'].update(title='Explained variance by different principal compo
nents')
py.iplot(fig)
```

```
This is the format of your plot grid:
[ (1,1) x1,y1 ]  [ (1,2) x2,y2 ]
[ (2,1) x3,y3 ]  [ (2,2) x4,y4 ]
[ (3,1) x5,y5 ]  [ (3,2) x6,y6 ]
```

Out[57]:

As we can see above, choosing two principal componnets with the highest eigenvalue will ensure that we save more than 95 percent of the data while reducing the dimension from nine to two.

Projection Matrix and Projection Onto the New Feature Space

In [58]:

```
Y1 = Xg1_std.dot(np.hstack((eig_pairs_g1[0][1].reshape(8,1), eig_pairs_g1[1
][1].reshape(8,1))))
Y2 = Xg2_std.dot(np.hstack((eig_pairs_g2[0][1].reshape(8,1), eig_pairs_g2[1
][1].reshape(8,1))))
Y3 = Xg3_std.dot(np.hstack((eig_pairs_g3[0][1].reshape(8,1), eig_pairs_g3[1
][1].reshape(8,1))))
Y4 = Xp1_std.dot(np.hstack((eig_pairs_p1[0][1].reshape(8,1), eig_pairs_p1[1
][1].reshape(8,1))))
Y5 = Xp2_std.dot(np.hstack((eig_pairs_p2[0][1].reshape(8,1), eig_pairs_p2[1
][1].reshape(8,1))))
Y6 = Xp3_std.dot(np.hstack((eig_pairs_p3[0][1].reshape(8,1), eig_pairs_p3[1
][1].reshape(8,1))))
# print('Matrix W:\n', matrix_w)
```

Here we will visualize the PCA results first for all the activities and then for the top three activities that are suspicious to play a key role in possible classification (walking on treadmill for two intervals and sit rest with px simulator doffed):

In [59]:

```
fig = tools.make_subplots(rows=3, cols=2,
                          subplot_titles=('good fit 1','poor fit 1','good fi
t 2', 'poor fit 2','good fit 3', 'poor fit 3'),
                          specs=[[{}, {}], [{}, {}], [{}, {}]],
                          horizontal_spacing = 0.1, vertical_spacing = 0.15)
i = 0
clr = ['black','orange', 'blue','red','cyan','magenta','yellow','deeppink',
'firebrick']
for name in ('don px components ', 'walk on treadmill (2)', 'walk on treadm
ill (1)', 'sit rest with px simulator doffed', 'sit rest with px simulator
donned (3)', 'sit rest bare limb', 'doff px components', 'sit rest with px
simulator donned (2)',
             'sit rest with px simulator donned (1)'):

    trace1 = go.Scatter(
        x=Y1[y_g1==name,0],
        y=Y1[y_g1==name,1],
        mode='markers',
        name=name,
        marker=dict(
            size=3,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace1, 1, 1)

    trace2 = go.Scatter(
```

```
        x=Y2[y_g2==name,0],
        y=Y2[y_g2==name,1],
        mode='markers',
        name=name,
        marker=dict(
            size=3,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace2, 2, 1)

    trace3 = go.Scatter(
        x=Y3[y_g3==name,0],
        y=Y3[y_g3==name,1],
        mode='markers',
        name=name,
        marker=dict(
            size=3,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace3, 3, 1)

    trace4 = go.Scatter(
        x=Y4[y_p1==name,0],
        y=Y4[y_p1==name,1],
        mode='markers',
        name=name,
        marker=dict(
            size=3,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace4, 1, 2)

    trace5 = go.Scatter(
        x=Y5[y_p2==name,0],
        y=Y5[y_p2==name,1],
        mode='markers',
        name=name,
        marker=dict(
            size=3,
            color = clr[i]))
    fig.append_trace(trace5, 2, 2)

    trace6 = go.Scatter(
        x=Y6[y_p3==name,0],
        y=Y6[y_p3==name,1],
        mode='markers',
        name=name,
        marker=dict(
            size=3,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace6, 3, 2)
    i += 1


fig['layout'].update(title='PC1 VS PC2 for all the activities')

py.iplot(fig)
```

This is the format of your plot grid:

```
[ (1,1) x1,y1 ]  [ (1,2) x2,y2 ]
[ (2,1) x3,y3 ]  [ (2,2) x4,y4 ]
[ (3,1) x5,y5 ]  [ (3,2) x6,y6 ]

The draw time for this plot will be slow for clients without much RAM.
```

```
/Users/rohamrazaghi/anaconda/lib/python2.7/site-
packages/plotly/plotly/plotly.py:1443: UserWarning:

Estimated Draw Time Slow
```

Out[59]:

In [ ]:

```
fig = tools.make_subplots(rows=3, cols=2,
                          subplot_titles=('good fit 1','poor fit 1','good fi
t 2', 'poor fit 2','good fit 3', 'poor fit 3'),
                          specs=[[{}, {}], [{}, {}], [{}, {}]],
                          horizontal_spacing = 0.1, vertical_spacing = 0.15)
i = 0
clr = ['black','orange', 'blue','red','cyan','magenta','yellow','deeppink',
'firebrick']

for name in ('walk on treadmill (2)', 'walk on treadmill (1)', 'sit rest wi
th px simulator doffed'):

    trace1 = go.Scatter(
```

```python
        x=Y1[y_g1==name,0],
        y=Y1[y_g1==name,1],
        mode='markers',
        name=name,
        marker=dict(
            size=1,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace1, 1, 1)

    trace2 = go.Scatter(
        x=Y2[y_g2==name,0],
        y=Y2[y_g2==name,1],
        mode='markers',
        name=name,
        marker=dict(
            size=1,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace2, 2, 1)

    trace3 = go.Scatter(
        x=Y3[y_g3==name,0],
        y=Y3[y_g3==name,1],
        mode='markers',
        name=name,
        marker=dict(
            size=1,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace3, 3, 1)

    trace4 = go.Scatter(
        x=Y4[y_p1==name,0],
        y=Y4[y_p1==name,1],
        mode='markers',
        name=name,
        marker=dict(
            size=1,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace4, 1, 2)

    trace5 = go.Scatter(
        x=Y5[y_p2==name,0],
        y=Y5[y_p2==name,1],
        mode='markers',
        name=name,
        marker=dict(
            size=1,
            color = clr[i]))
    fig.append_trace(trace5, 2, 2)

    trace6 = go.Scatter(
        x=Y6[y_p3==name,0],
        y=Y6[y_p3==name,1],
        mode='markers',
        name=name,
        marker=dict(
            size=1,
```

```
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace6, 3, 2)
    i += 1

fig['layout'].update(title='PC1 VS PC2 for walking on treadmill and sit res
t with px simulator doffed ')

py.iplot(fig)
```

In [ ]:

```
fig = tools.make_subplots(rows=3, cols=2,
                          subplot_titles=('good fit 1','poor fit 1','good fi
t 2', 'poor fit 2','good fit 3', 'poor fit 3'),
                          specs=[[{}, {}], [{}, {}], [{}, {}]],
                          horizontal_spacing = 0.1, vertical_spacing = 0.15)
i = 0
clr = ['black','orange', 'blue','red','cyan','magenta','yellow','deeppink',
'firebrick']

for name in ('walk on treadmill (2)', 'walk on treadmill (1)', 'sit rest wi
th px simulator doffed'):

    trace1 = go.Scatter(
        x=np.mean(Y1[y_g1==name,0]),
        y=np.mean(Y1[y_g1==name,1]),
        mode='markers',
        name=name,
        marker=dict(
            size=10,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace1, 1, 1)

    trace2 = go.Scatter(
        x=np.mean(Y2[y_g2==name,0]),
        y=np.mean(Y2[y_g2==name,1]),
        mode='markers',
        name=name,
        marker=dict(
            size=10,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace2, 2, 1)

    trace3 = go.Scatter(
        x=np.mean(Y3[y_g3==name,0]),
        y=np.mean(Y3[y_g3==name,1]),
        mode='markers',
        name=name,
        marker=dict(
            size=10,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace3, 3, 1)

    trace4 = go.Scatter(
        x=np.mean(Y4[y_p1==name,0]),
```

```
        y=np.mean(Y4[y_p1==name,1]),
        mode='markers',
        name=name,
        marker=dict(
            size=10,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace4, 1, 2)

    trace5 = go.Scatter(
        x=np.mean(Y5[y_p2==name,0]),
        y=np.mean(Y5[y_p2==name,1]),
        mode='markers',
        name=name,
        marker=dict(
            size=10,
            color = clr[i]))
    fig.append_trace(trace5, 2, 2)

    trace6 = go.Scatter(
        x=np.mean(Y6[y_p3==name,0]),
        y=np.mean(Y6[y_p3==name,1]),
        mode='markers',
        name=name,
        marker=dict(
            size=10,
            color = clr[i],
            opacity=0.8))
    fig.append_trace(trace6, 3, 2)
    i += 1

fig['layout'].update(title='Calculated means on PC1 VS PC2 for walking on t
readmill and sit rest with px simulator doffed ')

py.iplot(fig)
```

Fast Fourier transform

A fast Fourier transform (FFT) algorithm computes the discrete Fourier transform (DFT) of a sequence, or its inverse. Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors.[1] As a result, it manages to reduce the complexity of computing the DFT.

Below, we visualize the data first over the activities with the highest noise level (taking into account that the dataset is averaged over different location in order to ease the process of visualization), then we compare the FFT output among different trials for possible classification.

In [41]:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.fftpack

y1 = np.mean(df_good1[df_good1['Class']== 'walk on treadmill (1)'].ix[:,1:9
```

```
].values, axis = 1)
y2 = np.mean(df_good2[df_good2['Class']== 'walk on treadmill (1)'].ix[:,1:9
].values, axis = 1)
y3 = np.mean(df_good3[df_good3['Class']== 'walk on treadmill (1)'].ix[:,1:9
].values, axis = 1)
y4 = np.mean(df_poor1[df_poor1['Class']== 'walk on treadmill (1)'].ix[:,1:9
].values, axis = 1)
y5 = np.mean(df_poor2[df_poor2['Class']== 'walk on treadmill (1)'].ix[:,1:9
].values, axis = 1)
y6 = np.mean(df_poor3[df_poor3['Class']== 'walk on treadmill (1)'].ix[:,1:9
].values, axis = 1)

x1 = df_good1[df_good1['Class']== 'walk on treadmill (1)'].ix[:,0].values
x2 = df_good2[df_good2['Class']== 'walk on treadmill (1)'].ix[:,0].values
x3 = df_good3[df_good3['Class']== 'walk on treadmill (1)'].ix[:,0].values
x4 = df_poor1[df_poor1['Class']== 'walk on treadmill (1)'].ix[:,0].values
x5 = df_poor2[df_poor2['Class']== 'walk on treadmill (1)'].ix[:,0].values
x6 = df_poor3[df_poor3['Class']== 'walk on treadmill (1)'].ix[:,0].values


def fft(y):
    Fs = 150.0;  # sampling rate
    Ts = 1.0/Fs; # sampling interval
    #t = np.arange(0,1,Ts) # time vector


    n = len(y) # length of the signal
    k = np.arange(n)
    T = n/Fs
    frq = k/T # two sides frequency range
    frq = frq[range(n/2)] # one side frequency range

    Y = np.fft.fft(y)/n # fft computing and normalization
    Y = Y[range(n/2)]
    return Y, frq
y1f, frq1 = fft(y1)
y2f, frq2 = fft(y2)
y3f, frq3 = fft(y3)
y4f, frq4 = fft(y4)
y5f, frq5 = fft(y5)
y6f, frq6 = fft(y6)
trace1 = go.Scatter(
        x=x1,
        y=y1)
trace2 = go.Scatter(
        x=x2,
        y=y2)
trace3 = go.Scatter(
        x=x3,
        y=y3)
trace4 = go.Scatter(
        x=x4,
        y=y4)
trace5 = go.Scatter(
        x=x5,
        y=y5)
trace6 = go.Scatter(
        x=x6,
        y=y6)
fig = tools.make_subplots(rows=3, cols=2,
```

```
                           subplot_titles=('good fit 1','poor fit 1','good fi
t 2', 'poor fit 2','good fit 3', 'poor fit 3'),
                           specs=[[{}, {}], [{}, {}], [{}, {}]],
                           horizontal_spacing = 0.1, vertical_spacing = 0.15)
fig.append_trace(trace1, 1, 1)
fig.append_trace(trace2, 2, 1)
fig.append_trace(trace3, 3, 1)
fig.append_trace(trace4, 1, 2)
fig.append_trace(trace5, 2, 2)
fig.append_trace(trace6, 3, 2)
# fig, ax = plt.subplots(2, 1)
# ax[0].plot(x1,y1)
# ax[0].set_xlabel('Time')
# ax[0].set_ylabel('Amplitude')
# ax[1].plot(frq,abs(Y),'r') # plotting the spectrum
# ax[1].set_xlabel('Freq (Hz)')
# ax[1].set_ylabel('|Y(freq)|')
fig['layout'].update(title='Original averaged signal over all 8 locations i
n the first walking on treadmill activity')


py.iplot(fig)
```

```
This is the format of your plot grid:
[ (1,1) x1,y1 ]  [ (1,2) x2,y2 ]
[ (2,1) x3,y3 ]  [ (2,2) x4,y4 ]
[ (3,1) x5,y5 ]  [ (3,2) x6,y6 ]
```

Out[41]:

In [48]:

```
%matplotlib inline
fig = plt.figure()

ax1 = fig.add_subplot(321)
ax1.plot(frq1, abs(y1f), 'r-')
ax1.set_title('good fit 1')
ax2 = fig.add_subplot(322)
ax2.plot(frq4, abs(y4f), 'k-')
ax2.set_title('poor fit 1')
ax3 = fig.add_subplot(323)
ax3.plot(frq2, abs(y2f), 'b-')
ax3.set_title('good fit 2')
ax4 = fig.add_subplot(324)
ax4.plot(frq5, abs(y5f), 'g-')
ax4.set_title('poor fit 2')
ax5 = fig.add_subplot(325)
ax5.plot(frq3, abs(y3f), 'g-')
ax5.set_title('good fit 3')
ax6 = fig.add_subplot(326)
ax6.plot(frq6, abs(y5f), 'g-')
ax6.set_title('poor fit 3')
plt.tight_layout()
fig = plt.gcf()

plotly_fig = tls.mpl_to_plotly( fig )
plotly_fig['layout']['title'] = 'FFT for all trials on first walking period
(Amplitude VS Freq)'
plotly_fig['layout']['margin'].update({'t':50})
py.iplot(plotly_fig)
```

Out[48]:

Above analysis was also performed on the second walking period and both revealed no significant difference among the datasets. Note:(It would be a good idea to run this individually on different locations without calculating the average).

In [ ]:

```

```