Ground Truth Validation with Plotting

May 8, 2020

0.0.1 Ground Truth Validation with Plotting

This notebook provides ground truth validation with plots by classifcation in PCA space.

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        from sklearn.preprocessing import StandardScaler
        import numpy as np
        from importHelpers.response import *
        from mlxtend.preprocessing import minmax_scaling
        from mpl_toolkits.mplot3d import Axes3D
        from sklearn.decomposition import PCA
        from sklearn.cluster import DBSCAN
        from sklearn import metrics
        from sklearn.preprocessing import StandardScaler
```

0.0.2 Clean

We import and normalize the data.

```
In [2]: xls = pd.ExcelFile(r'data\\191126P2_ROIAnnotationSummary_200218.xlsx')
        df = pd.read_excel(xls, 'Annotation_Summary')
        df = df[['Flash', '2P ROI', 'RBPMS', 'Syt10+', 'Syt6+', 'CAVIII', 'ChAT', 'Satb2', 'ME
        df = df.dropna(axis = 0, subset = ["2P ROI"])
        df = df[df['2P ROI'].apply(lambda x: str(x).isdigit())]
        df = df.astype({"2P ROI": int})
        for col in ['Syt10+', 'Syt6+', 'CAVIII', 'ChAT', 'Satb2', 'MEIS', 'CalR']:
            df[col] = df[col].apply(lambda x: int(not pd.isna(x)))
```

```
In [3]: l = list(df.T)
        def name_merge(x):
            p = [str(i[l[x]]) for _, i in df.loc[[l[x]]].to_dict().items()]
            return p[0] + '_wave_' + str(p[1])
        name_merge(0)

        def uniquer(x):
            return "".join([str(i[l[x]]) for _, i in df.loc[[l[x]]].to_dict().items()][2:])
```

```
d = {}
c = 0
z = []
for i in range(df.shape[0]):
    u = uniquer(i)
    if u not in d.keys():
        d[u] = c
        c += 1
    z.append(d[u])
df.insert(10, "Class", z)

s = []
for i in range(df.shape[0]):
    s.append(name_merge(i))
```

0.0.3 Combine

We combine our data into one large sheet.

```
In [4]: # FILENAME
        xlsx_filename = "data\\191126P2PhysData_withlabels.xlsx"
        excel = pd.ExcelFile(xlsx_filename)

        def renamer(sheet, ind):
            l = lambda name: str(ind) + '_' + name
            sheet = sheet.rename(index = l)
            return sheet
        i = 0
        new_sheetnames = ['Flash_40', 'Flash_52', 'Flash_56', 'Flash_58', 'Flash_60', 'Flash_6
        total = renamer(pd.read_excel(xlsx_filename,sheet_name=excel.sheet_names[i], header=0)
        for i in range(1, len(excel.sheet_names)):
            print('Working on sheet ' + str(i + 1) + ' of ' + str(len(excel.sheet_names)))
            total = total.append(renamer(pd.read_excel(xlsx_filename,sheet_name=excel.sheet_na
        print("Sheet combination complete.")
        n = total

        def getClassByName(name):
            return z[s.index(name)]
```

```
Working on sheet 2 of 8
Working on sheet 3 of 8
Working on sheet 4 of 8
Working on sheet 5 of 8
Working on sheet 6 of 8
Working on sheet 7 of 8
Working on sheet 8 of 8
Sheet combination complete.
```

In [5]:
```
n = n[[i in s for i in n.index]]
n_class = []
for name in list(n.index):
    n_class.append(getClassByName(name))
```

In [6]:
```
def transform(initial):
    # remove and subtract baseline
    # c = frameToSecDF(initial.sub(initial['baseline'], axis = 'rows').drop('baseline'
    # drop 70
    c = initial
    a = [a - b > 70 for a, b in zip(list(c.max(axis = 1)), list(c.min(axis= 0)))]
    dropped = []
    for i in range(len(a)):
        if not a[i]:
            dropped.append(list(c.T)[i])
    c = c.drop(dropped, axis = 0)
    # -1 1 scale
    last = c[c.columns[-15:]]
    last = last.mean(axis=1)
    ne = c.sub(last, axis = 0)
    n_one = ne.div(ne.abs().max(axis = 1), axis = 0)
    return n_one
```

In [7]:
```
#n = df
pca = PCA(n_components=30)
principalComponents = pca.fit_transform(n)
principalDf = pd.DataFrame(data = principalComponents)
pca_n = pd.DataFrame(data = pca.inverse_transform(principalComponents))
pca_n = pca_n.rename(index={a:b for a,b in zip(range(len(list(n.T))),list(n.T))}, colu
# comment next line for no PCA
next_n = n
```

0.0.4 Cluster

We cluster our data and check the accuracy.

In [63]:
```
db = DBSCAN(eps=3, min_samples=2).fit(principalDf)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
dlabels = db.labels_
print("DBSCAN with your params found:")
print(str(max(dlabels + 1)) + " classified labels")
print(str(list(dlabels).count(-1)) + ' unclassified points out of ' + str(len(dlabels)
```

```
DBSCAN with your params found:
17 classified labels
263 unclassified points out of 603
```

```
In [64]: def accuracy(dlabels, n_class):
             correct = 0
             total = 0
             for i in range(len(n_class)):
                 for j in range(i + 1, len(n_class)):
                     if (dlabels[i] == -1):
                         continue
                     if (dlabels[i] == dlabels[j]):
                         if (n_class[i] == n_class[j]):
                             correct += 1
                         total += 1
             print(correct / total)
             return correct, total
```

```
In [65]: c, t = accuracy(dlabels, n_class)
```

```
0.5484463526912181
```

```
In [66]: c/t
```

```
Out[66]: 0.5484463526912181
```

```
In [67]: principalDf
```

```
Out[67]:              0          1         2         3         4         5         6  \
         0     5.296242  11.438733 -1.941788  0.672688  1.512897  1.593108 -1.556878
         1    -8.961606   0.821415  0.872051 -0.068536 -0.148502  0.373586  0.223538
         2    46.047550   9.843774  8.505736  0.669403  1.150362  3.110441  0.650291
         3     6.629867  -8.005853  2.974874 -4.160257  0.386959 -0.751278  1.256460
         4     7.262711  -3.773677 -0.234283 -2.012756 -0.606126  0.253001 -0.493958
         ..         ...        ...       ...       ...       ...       ...       ...
         598  56.478581  -1.069781  5.394306  5.014743  0.146647 -3.288913 -0.864049
         599   0.900329   2.081248 -2.569101 -1.646001 -5.225555  4.446586  1.032229
         600  86.069387   4.911019 -8.700686 -3.982611  5.676211 -5.117172  9.869552
         601  -2.663390   3.564411  1.047435  0.087165 -1.445347  0.126223  0.728677
         602 -22.709063   0.041991 -2.249121 -0.765698  0.067335  0.423351  0.004722

                     7         8         9  ...        20        21        22  \
         0    0.166458  1.623784 -2.099839  ... -0.848804  0.449910 -0.306620
         1   -0.102673 -0.009905  0.051821  ... -0.114216  0.095280 -0.138507
         2   -1.593223  2.232610 -0.454005  ... -0.616975  0.074850 -0.633611
         3    0.203335  1.030219  0.110859  ... -0.526312 -0.158662 -0.115096
         4    0.456840 -0.285309  0.398290  ... -0.164122 -0.023964  0.209616
         ..        ...       ...       ...  ...       ...       ...       ...
         598 -0.948919 -0.655205 -1.388754  ...  0.540207 -1.289464 -0.589563
         599  0.894688  0.777157  0.931601  ...  1.838356  0.061765  0.421710
         600  2.054305  5.325421 -0.927324  ...  1.085215  2.096575  0.242512
         601  0.323571  1.117575 -0.380801  ...  0.191265  0.346384 -0.562886
```

```
602  0.580567  0.473852  0.150237  ...  0.414118 -0.123561 -0.513917

           23        24        25        26        27        28        29
0    0.707912  0.665966 -0.810913  0.364751 -0.011992  0.190733  0.932099
1    0.082974 -0.149544 -0.114316  0.228797 -0.055865 -0.095102  0.129866
2    0.549459  0.463773 -0.364180  0.199313 -0.221965 -0.192388  0.421371
3   -0.033200 -0.236026 -0.334343 -0.215082  0.008013 -0.108442  0.356863
4    0.300078 -0.384097  0.226188 -0.013780 -0.011374 -0.005103  0.084176
..        ...       ...       ...       ...       ...       ...       ...
598 -0.823798  0.434301  0.533213  0.762063 -0.105738 -0.066573  0.208553
599  0.331686 -0.341397 -0.087614 -0.037755  0.092054  0.860092 -0.357990
600 -0.209664  0.434316 -0.833985  0.624363 -0.282178 -0.286373  0.224959
601 -0.088753  0.297879  0.266931 -0.554825  0.324423 -0.227923  0.341419
602 -0.270670  0.714482  0.260941 -0.155110  0.357669 -0.707748  0.162137

[603 rows x 30 columns]
```

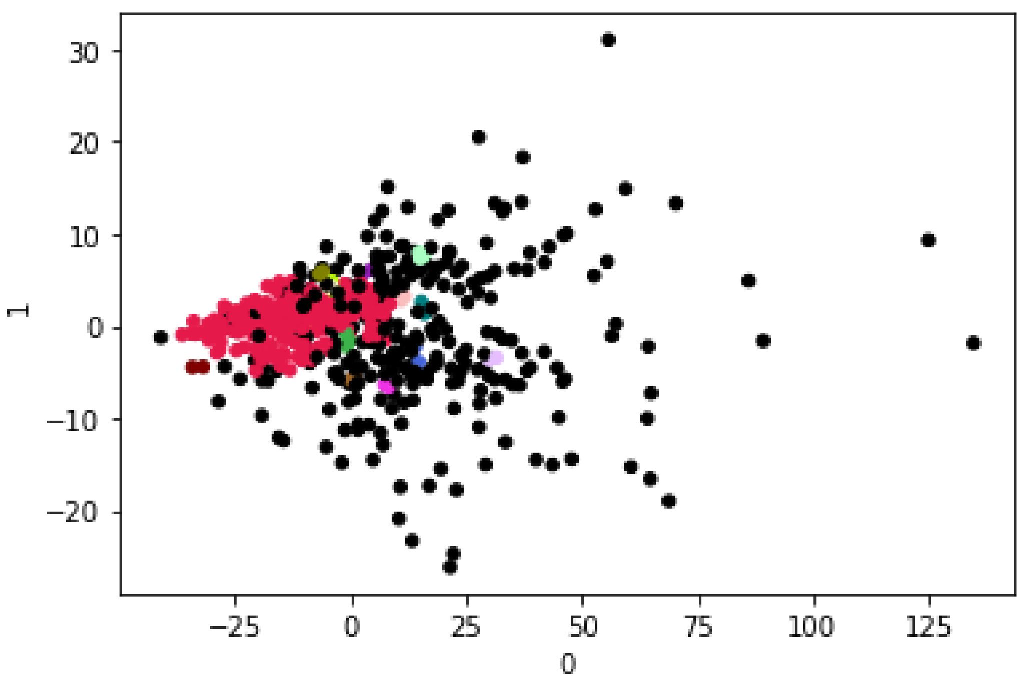
In [68]:
```
colors = ['#e6194b', '#3cb44b', '#ffe119', '#4363d8', '#f58231', '#911eb4', '#46f0f0'
plotcolors = [colors[i] for i in dlabels]
```

0.0.5 Plotting

We plot the principal components with their ground truth colors below. The black color represents unclassified.

In [69]:
```
principalDf.iloc[:,0:2].plot(kind='scatter',x=0, y=1, color=plotcolors)
```

Out[69]: `<matplotlib.axes._subplots.AxesSubplot at 0x4b0cff0>`



```
In [70]: colors = ['#e6194b', '#3cb44b', '#ffe119', '#4363d8', '#f58231', '#911eb4', '#46f0f0'
         plotcolors = [colors[i] for i in dlabels]
```

We plot the principal components with their ground truth colors below. In this case, we have removed the unclassified points.

```
In [71]: principalDf.iloc[:,0:2].plot(kind='scatter',x=0, y=1, color=plotcolors)

Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x4b51cf0>
```