Ground Truth Validation

May 8, 2020

0.0.1 Ground Truth Validation

This notebook provides ground truth validation accuracy testing.

```
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_nar
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 [16]: #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 [8]: db = DBSCAN(eps=3, min_samples=2).fit(next_n)
        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:
12 classified labels
298 unclassified points out of 603
```

```
In [9]: 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 [10]: c, t = accuracy(dlabels, n_class)

0.5713700803521162


In [11]: c/t

Out[11]: 0.5713700803521162
```