Indoor Wireless Localization

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.linear_model import Perceptron, LinearRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier



train_data = pd.read_csv('D_Train1.csv')
train_data =train_data.values

X = train_data[:,1:]
y = train_data[:,0]

test_data = pd.read_csv('D_Test1.csv')
test_data = test_data.values

X_test = test_data[:,1:]
y_test = test_data[:,0]



PCA
pca = PCA(n_components=2)
# pca = pca.fit(X)
# print(pca.explained_variance_ratio_)
X = pca.fit_transform(X)
X_test = pca.transform(X_test)
```

```
#Standardize
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_test = scaler.transform(X_test)


#Normalize(Better)
# scaler = MinMaxScaler()
# X = scaler.fit_transform(X)
# X_test = scaler.transform(X_test)


def VisualizeResult(X_test,y_test,classifier,title):
        x_set, y_set = X_test, y_test
        x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:,
0].max() + 1, step = 0.01), np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:,
1].max() + 1, step = 0.01))
        plt.figure()

        y = classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape)

        plt.contourf(x1, x2, y, alpha = 0.75, cmap = ListedColormap(('red',
'green','purple','black')))
        plt.xlim(x1.min(), x1.max())
        plt.ylim(x2.min(), x2.max())

        for i, j in enumerate(np.unique(y_set)):
                plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c =
ListedColormap(('orange', 'blue','red','yellow'))(i), label = j)

        plt.title(title)
        plt.xlabel('pc1')
        plt.ylabel('pc2')
        plt.legend()
        plt.show()
```

```
#NB
#########################################################################
####
skf = StratifiedKFold(shuffle=True)
table =[]
for train_index, val_index in skf.split(X,y):
    X_train, X_val = X[train_index], X[val_index]
    y_train, y_val = y[train_index], y[val_index]

    clf = GaussianNB()
    clf.fit(X_train,y_train)
    val_acc = clf.score(X_val,y_val)
    table.append(val_acc)


y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print("Naive Bayes:",round(100*acc,2),"%")
print("cross_val_acc mean:",round(np.mean(table),3))
print("cross_val_acc std:",round(np.std(table),3))
print(cm,"\n")

# VisualizeResult(X_test, y_test, clf,'Naive Bayes(Testing set)')




#SVM
#########################################################################
####
skf = StratifiedKFold(shuffle=True)
table =[]
for train_index, val_index in skf.split(X,y):
    X_train, X_val = X[train_index], X[val_index]
    y_train, y_val = y[train_index], y[val_index]
```

```
        clf2 = SVC(C=100)
        clf2.fit(X_train,y_train)
        val_acc = clf2.score(X_val,y_val)
        table.append(val_acc)

y_pred2 = clf2.predict(X_test)
acc2 = accuracy_score(y_test, y_pred2)
cm2 = confusion_matrix(y_test, y_pred2)
print("SVM:",round(100*acc2,2),"%")
print("cross_val_acc mean:",round(np.mean(table),3))
print("cross_val_acc std:",round(np.std(table),3))
print(cm2,"\n")

# VisualizeResult(X_test, y_test, clf2,'SVM(Testing set)' )



#Perceptron
#########################################################################
####
skf = StratifiedKFold(shuffle=True)
table =[]
for train_index, val_index in skf.split(X,y):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        clf3 = Perceptron()
        clf3.fit(X_train,y_train)
        val_acc = clf3.score(X_val,y_val)
        table.append(val_acc)

y_pred3 = clf3.predict(X_test)
acc3 = accuracy_score(y_test, y_pred3)
cm3 = confusion_matrix(y_test, y_pred3)
print("Perceptron:",round(100*acc3,2),"%")
print("cross_val_acc mean:",round(np.mean(table),3))
print("cross_val_acc std:",round(np.std(table),3))
```

```
print(cm3,"\n")

# VisualizeResult(X_test, y_test, clf3,'Perceptron(Testing set)' )




#OVR
#########################################################################
####
skf = StratifiedKFold(shuffle=True)
table =[]
for train_index, val_index in skf.split(X,y):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        clf4 = OneVsRestClassifier(SVC())
        clf4.fit(X_train,y_train)
        val_acc = clf4.score(X_val,y_val)
        table.append(val_acc)

y_pred4 = clf4.predict(X_test)
acc4 = accuracy_score(y_test, y_pred4)
cm4 = confusion_matrix(y_test, y_pred4)
print("OVR:",round(100*acc4,2),"%")
print("cross_val_acc mean:",round(np.mean(table),3))
print("cross_val_acc std:",round(np.std(table),3))
print(cm4,"\n")


# VisualizeResult(X_test, y_test, clf4,'OneVsRest(Testing set)')
```

```
#KNN
#########################################################################
####
skf = StratifiedKFold(shuffle=True)
table =[]
for train_index, val_index in skf.split(X,y):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        clf5 = KNeighborsClassifier()
        clf5.fit(X_train,y_train)
        val_acc = clf5.score(X_val,y_val)
        table.append(val_acc)

y_pred5 = clf5.predict(X_test)
acc5 = accuracy_score(y_test, y_pred5)
cm5 = confusion_matrix(y_test, y_pred5)
print("KNN:",round(100*acc5,2),"%")
print("cross_val_acc mean:",round(np.mean(table),3))
print("cross_val_acc std:",round(np.std(table),3))
print(cm5,"\n")

# VisualizeResult(X_test, y_test, clf5,'KNN(Testing set)')




#DecisionTree
#########################################################################
####
skf = StratifiedKFold(shuffle=True)
table =[]
for train_index, val_index in skf.split(X,y):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]
```

```
        clf6 = DecisionTreeClassifier()
        clf6.fit(X_train,y_train)
        val_acc = clf6.score(X_val,y_val)
        table.append(val_acc)

y_pred6 = clf6.predict(X_test)
acc6 = accuracy_score(y_test, y_pred6)
cm6 = confusion_matrix(y_test, y_pred6)
print("DecisionTree:",round(100*acc6,2),"%")
print("cross_val_acc mean:",round(np.mean(table),3))
print("cross_val_acc std:",round(np.std(table),3))
print(cm6,"\n")

# VisualizeResult(X_test, y_test, clf6, 'Decision Tree(Testing set)')




#RandomForest
##########################################################################
####
skf = StratifiedKFold(shuffle=True)
table =[]
for train_index, val_index in skf.split(X,y):
        X_train, X_val = X[train_index], X[val_index]
        y_train, y_val = y[train_index], y[val_index]

        clf7 = RandomForestClassifier(max_depth=50)
        clf7.fit(X_train,y_train)
        val_acc = clf7.score(X_val,y_val)
        table.append(val_acc)

y_pred7 = clf7.predict(X_test)
acc7 = accuracy_score(y_test, y_pred7)
cm7 = confusion_matrix(y_test, y_pred7)
print("RandomForest:",round(100*acc7,2),"%")
print("cross_val_acc mean:",round(np.mean(table),3))
```

```
print("cross_val_acc std:",round(np.std(table),3))
print(cm7,"\n")

# VisualizeResult(X_test, y_test, clf7,'Random Forest(Testing set)')
```

Hand Postures

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from matplotlib.colors import ListedColormap
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.linear_model import Perceptron, LinearRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.utils import shuffle
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier


train_data = pd.read_csv('D_train.csv')
train_data = pd.DataFrame.sort_index(train_data,axis=1)

# train_data = train_data.fillna(0)

#define new fetures
x_m = np.mean(train_data.iloc[:,3:15],axis=1)
y_m = np.mean(train_data.iloc[:,15:27],axis=1)
z_m = np.mean(train_data.iloc[:,27:39],axis=1)
```

```
x_std = np.std(train_data.iloc[:,3:15],axis=1)
y_std = np.std(train_data.iloc[:,15:27],axis=1)
z_std = np.std(train_data.iloc[:,27:39],axis=1)
x_max = np.max(train_data.iloc[:,3:15],axis=1)
y_max = np.max(train_data.iloc[:,15:27],axis=1)
z_max = np.max(train_data.iloc[:,27:39],axis=1)
x_min = np.min(train_data.iloc[:,3:15],axis=1)
y_min = np.min(train_data.iloc[:,15:27],axis=1)
z_min = np.min(train_data.iloc[:,27:39],axis=1)
count = train_data.count(axis = 'columns')
count -= 3



train_data.insert(3,column='x_m', value = x_m)
train_data.insert(4,column='y_m', value = y_m)
train_data.insert(5,column='z_m', value = z_m)
train_data.insert(6,column='x_std', value = x_std)
train_data.insert(7,column='y_std', value = y_std)
train_data.insert(8,column='z_std', value = z_std)
train_data.insert(9,column='x_max', value = x_max)
train_data.insert(10,column='y_max', value = y_max)
train_data.insert(11,column='z_max', value = z_max)
train_data.insert(12,column='x_min', value = x_min)
train_data.insert(13,column='y_min', value = y_min)
train_data.insert(14,column='z_min', value = z_min)
train_data.insert(15,column='total', value = count)




# X = train_data.iloc[:,3:9].values
# # train_data = train_data.values
# # X = train_data[:,3:9].round(3)
# y = train_data.iloc[:,0].values
```

```
test_data = pd.read_csv('D_test.csv')
test_data = pd.DataFrame.sort_index(test_data,axis=1)

# test_data = test_data.fillna(0)

x_m = np.mean(test_data.iloc[:,3:15],axis=1)
y_m = np.mean(test_data.iloc[:,15:27],axis=1)
z_m = np.mean(test_data.iloc[:,27:39],axis=1)
x_std = np.std(test_data.iloc[:,3:15],axis=1)
y_std = np.std(test_data.iloc[:,15:27],axis=1)
z_std = np.std(test_data.iloc[:,27:39],axis=1)
x_max = np.max(test_data.iloc[:,3:15],axis=1)
y_max = np.max(test_data.iloc[:,15:27],axis=1)
z_max = np.max(test_data.iloc[:,27:39],axis=1)
x_min = np.min(test_data.iloc[:,3:15],axis=1)
y_min = np.min(test_data.iloc[:,15:27],axis=1)
z_min = np.min(test_data.iloc[:,27:39],axis=1)
count = test_data.count(axis = 'columns')
count -= 3


test_data.insert(3,column='x_m', value = x_m)
test_data.insert(4,column='y_m', value = y_m)
test_data.insert(5,column='z_m', value = z_m)
test_data.insert(6,column='x_std', value = x_std)
test_data.insert(7,column='y_std', value = y_std)
test_data.insert(8,column='z_std', value = z_std)
test_data.insert(9,column='x_max', value = x_max)
test_data.insert(10,column='y_max', value = y_max)
test_data.insert(11,column='z_max', value = z_max)
test_data.insert(12,column='x_min', value = x_min)
test_data.insert(13,column='y_min', value = y_min)
test_data.insert(14,column='z_min', value = z_min)
test_data.insert(15,column='total', value = count)
test_data = test_data.values

col_index = 16
X_test = test_data[:,3:col_index]
```

```
y_test = test_data[:,0]


#PCA
# pca = PCA(n_components=3)
# # pca = pca.fit(X)
# # print(pca.explained_variance_ratio_)
# X = pca.fit_transform(X)
# X_test = pca.transform(X_test)


#Normalize
# scaler = MinMaxScaler()
# X = scaler.fit_transform(X)
# X_test = scaler.transform(X_test)

#Standardize
# scaler = StandardScaler()
# X = scaler.fit_transform(X)
# X_test = scaler.transform(X_test)


# principalDf = pd.DataFrame(data=PrincipalComp, columns =
['principal1','principal2'])
# X = principalDf.iloc[:,:]


label = list(set (train_data["User"]))



def classifier(train_data,X_test,y_test,classifier,name):
    table =[]
    col_index = 16
    for i in set(train_data["User"]):

        scaler = StandardScaler()
        val = train_data.loc[train_data["User"]==i].iloc[:,:col_index]
```

```
        val = shuffle(val)
        train = train_data.loc[train_data["User"]!=i].iloc[:,:col_index]
        train = shuffle(train)

        X_val = val.iloc[:,3:col_index]
        X_train = train.iloc[:,3:col_index]
        y_val = val.iloc[:,0]
        y_train = train.iloc[:,0]

        X_train = scaler.fit_transform(X_train)
        X_val = scaler.transform(X_val)

        clf = classifier
        clf.fit(X_train,y_train)
        val_acc = clf.score(X_val,y_val)
        table.append(val_acc)
        maxacc = np.amax(table)

        result = np.where(maxacc == table)

print(table)
print(label[int(result[0])])

X_train = train_data.iloc[:,3:col_index]
y_train = train_data.iloc[:,0]
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)

clf = classifier
clf.fit(X_train,y_train)

X_test = scaler.transform(X_test)
print("cross_val_acc mean:",round(np.mean(table),3))
print("cross_val_acc std:",round(np.std(table),3))
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print(name,":",round(100*acc,2),"%\n")
```

```
    print(cm,"\n")

classifier(train_data, X_test,y_test, GaussianNB(),"Naive Bayes")
classifier(train_data, X_test,y_test, SVC(C=10,gamma =0.02),"SVM")
classifier(train_data, X_test,y_test, Perceptron(),"Perceptron")
classifier(train_data, X_test,y_test, OneVsRestClassifier(SVC()),"OVR")
classifier(train_data, X_test,y_test, KNeighborsClassifier(n_neighbors = 10),"KNN")
classifier(train_data, X_test,y_test, DecisionTreeClassifier(),"Decision Tree")
classifier(train_data, X_test,y_test, RandomForestClassifier(max_depth =
50),"Random Forest")

##find pararmeters

# Acc = np.zeros([10,10])
# DEV = np.zeros([10,10])
# r = np.logspace(-3,3,10,endpoint='True')
# C = np.logspace(-3,3,10,endpoint='True')



# maxAcc=[]
# bestDev=[]
# r_Array =[]
# C_Array =[]

# for i in range(10):
#     for j in range(10):
#         table =[]
#         col_index = 15
#         for k in set(train_data["User"]):

#             scaler = StandardScaler()
#             val = train_data.loc[train_data["User"]==k].iloc[:,:col_index]
#             val = shuffle(val)
#             train = train_data.loc[train_data["User"]!=k].iloc[:,:col_index]
#             train = shuffle(train)

#             X_val = val.iloc[:,3:col_index]
#             X_train = train.iloc[:,3:col_index]
```

```
#                     y_val = val.iloc[:,0]
#                     y_train = train.iloc[:,0]

#                     X_train = scaler.fit_transform(X_train)
#                     X_val = scaler.transform(X_val)

#                     clf = SVC(C=C[j],gamma=r[i],kernel='rbf')
#                     clf.fit(X_train,y_train)
#                     val_acc = clf.score(X_val,y_val)
#                     table.append(val_acc)
#                 print(j)
#                 DEV[i,j] = np.std(table)
#                 Acc[i,j]= np.mean(table)

# maxacc = np.amax(Acc)
# result = np.where(Acc == maxacc)
# listOfCordinates = list(zip(result[0], result[1]))
# for cord in listOfCordinates:
#         print(cord,r[cord[0]],C[cord[1]])
#         print('std:',DEV[cord[0],cord[1]])
```