```
# -*- coding: utf-8 -*-
"""
Created on Fri May  8 14:45:10 2020

@author: mrhaboon
"""

# -*- coding: utf-8 -*-
"""
Created on Fri May  8 00:37:07 2020

@author: mrhaboon
"""

# -*- coding: utf-8 -*-
"""
Created on Wed May  6 21:21:40 2020

@author: mrhaboon
"""

# -*- coding: utf-8 -*-
"""
Created on Wed May  6 13:23:47 2020

@author: mrhaboon
"""
import csv
from sklearn import preprocessing
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from itertools import combinations
from sklearn.model_selection import cross_val_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.decomposition import PCA
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import GridSearchCV
import os

os.chdir('C:/Users/mrhaboon/Desktop/usc_coursework/ee_559/Project')
#par=20
#par=7
#fold=3
fold=5
poly_count=3

train1=[]
test1=[]

with open('D_Train1.csv') as csv_file:
    reader=csv.reader(csv_file)
    j=0
    for i in reader:
        if j==0:
            j+=1
            continue
```

```
        else:
            train1.append([float(x) for x in i])
    train1=np.array(train1)

j=0
with open('D_Test1.csv') as csv_file:
    reader=csv.reader(csv_file)
    for i in reader:
        if j==0:
            j+=1
            continue
        else:
            test1.append([float(x) for x in i])
    test1=np.array(test1)

#%%

train1_data=train1[:,1:]
label_train=train1[:,0]

test1_data=test1[:,1:]
label_test=test1[:,0]

scaler=preprocessing.StandardScaler().fit(train1_data)
std_train=np.concatenate((label_train[:,np.newaxis],scaler.transform(train1_data)),axis=1)
std_test=np.concatenate((label_test[:,np.newaxis],scaler.transform(test1_data)),axis=1)


normer=preprocessing.MinMaxScaler()
norm_train=np.concatenate((label_train[:,np.newaxis],normer.fit_transform(train1_data)),axis=1)
norm_test=np.concatenate((label_test[:,np.newaxis],normer.fit_transform(test1_data)),axis=1)


param_grid=[{'criterion':['gini'],'max_depth':[i+3 for i in range(15)]},{'criterion':['entropy'],'m


#%%
class find_best_tree:

    def __init__(self,train,test,dim='none'):
        self.train=train
        self.train_data=self.train[:,1:]
        self.train_labels=self.train[:,0]
        self.test=test
        self.test_data=self.test[:,1:]
        self.test_labels=self.test[:,0]
        self.dim=dim

        if dim=='none':
            print('before gen')
            best_model=self.gen_model(self.train_data,self.train_labels)
            print('after gen')
            best_params=best_model[2]
            best_p_stats=best_model[1]
            best_dim_stats=best_model[0]
```

```
            self.best={'best params':best_params,'best_p_stats':best_p_stats,'best_dim_stats':best_
            self.final_model=best_model[-1]
```

```
            self.final_model.fit(self.train_data,self.train_labels)
            self.performance(self.train_data,self.test_data)

        elif dim=='perm':
            self.gen_perm()
            best_params=[]
            best_p_stats=(0,0)
            best_dim_stats=(0,0)

            for i in self.perm:
                tmp_train=self.dim_transform(self.train_data,i)
                curr_model=self.gen_model(tmp_train,self.train_labels)
                curr_score=(curr_model[0][0]+curr_model[1][0])/2
                if curr_score>(best_p_stats[0]+best_dim_stats[0]):
                    best_params=curr_model[2]
                    best_p_stats=curr_model[1]
                    best_dim_stats=curr_model[0]
                    best_model=curr_model[-1]
                    best_dim=i
```

```
            self.best={'best params':best_params,'best_p_stats':best_p_stats,'best_dim_stats':best_

            self.final_model=best_model
            new_train=self.dim_transform(self.train_data,best_dim)
            new_test=self.dim_transform(self.test_data,best_dim)
            self.final_model.fit(new_train,self.train_labels)
            self.performance(new_train,new_test)

        elif dim=='PCA':
            best_params=[]
            best_p_stats=(0,0)
            best_dim_stats=(0,0)

            for i in range(7):
                print('progress')
                tmp=PCA(n_components=i+1)
                tmp.fit(self.train_data)
                tmp_train=tmp.transform(self.train_data)
                print('before gen')

                curr_model=self.gen_model(tmp_train,self.train_labels)
                print('after gen')
                curr_score=(curr_model[0][0]+curr_model[1][0])/2
                if curr_score>(best_p_stats[0]+best_dim_stats[0]):
                    best_params=curr_model[2]
                    best_p_stats=curr_model[1]
                    best_dim_stats=curr_model[0]
                    best_model=curr_model[-1]
                    best_dim=i+1

            self.best={'best params':best_params,'best_p_stats':best_p_stats,'best_dim_stats':best_

            self.final_model=best_model
```

```
            tran=PCA(n_components=best_dim)
            tran.fit(self.train_data)
            new_train=tran.transform(self.train_data)
            new_test=tran.transform(self.test_data)
            self.final_model.fit(new_train,self.train_labels)
            self.performance(new_train,new_test)


        elif dim=='Fisher':
            best_params=[]
            best_p_stats=(0,0)
            best_dim_stats=(0,0)

            for i in range(3):
                print('progress')
                tran=LinearDiscriminantAnalysis(n_components=i+1)
                tran.fit(self.train_data,self.train_labels)
                tmp_train=tran.transform(self.train_data)


                print('before gen')
                curr_model=self.gen_model(tmp_train,self.train_labels)
                print('after gen')
                curr_score=(curr_model[0][0]+curr_model[1][0])/2
                if curr_score>(best_p_stats[0]+best_dim_stats[0]):
                    best_params=curr_model[2]
                    best_p_stats=curr_model[1]
                    best_dim_stats=curr_model[0]
                    best_model=curr_model[-1]
                    best_dim=i+1

            self.best={'best params':best_params,'best_p_stats':best_p_stats,'best_dim_stats':best_

            self.final_model=best_model
```

```
            tran=LinearDiscriminantAnalysis(n_components=best_dim)
            tran.fit(self.train_data,self.train_labels)
            new_train=tran.transform(self.train_data)
            new_test=tran.transform(self.test_data)
            self.final_model.fit(new_train,self.train_labels)
            self.performance(new_train,new_test)

        elif dim=='poly':
            best_params=[]
            best_p_stats=(0,0)
            best_dim_stats=(0,0)


            for i in range(poly_count):
                poly=PolynomialFeatures(i+1)
                tmp_train=poly.fit_transform(self.train_data)
```

```
                curr_model=self.gen_model(tmp_train,self.train_labels)
                curr_score=(curr_model[0][0]+curr_model[1][0])/2
                if curr_score>(best_p_stats[0]+best_dim_stats[0]):
                    best_params=curr_model[2]
                    best_p_stats=curr_model[1]
                    best_dim_stats=curr_model[0]
                    best_model=curr_model[-1]
                    best_dim=i+1

            self.best={'best params':best_params,'best_p_stats':best_p_stats,'best_dim_stats':best_
            self.final_model=best_model
```

```
            poly=PolynomialFeatures(best_dim)
            new_train=poly.fit_transform(self.train_data)
            new_test=poly.fit_transform(self.test_data)
            self.final_model.fit(new_train,self.train_labels)
            self.performance(new_train,new_test)

    def performance(self,train,test):
        self.train_conf=confusion_matrix(self.train_labels,self.final_model.predict(train))
        self.test_conf=confusion_matrix(self.test_labels,self.final_model.predict(test))
        self.acc=self.best
#         self.train_acc=
#         self.test_acc=

    def gen_perm(self):
        self.perm=[]
        tmp=[]
        data_length=len(self.train_data[0])
        for i in range(data_length):
            tmp.append(i+1)
        for i in range(data_length):
            for i in combinations(tmp,i+1):
                self.perm.append(i)

    def dim_transform(self,data,dim):
        result=[]
        for i in data:
            tmp_data=[]
            for j in dim:
                tmp_data.append(i[j-1])
            result.append(tmp_data)
        return result

    def gen_model(self,data,labels):
        tmp=DecisionTreeClassifier()
        clf=GridSearchCV(tmp,param_grid)
        clf.fit(data,labels)
        best_index=clf.best_index_
        scores=[]
        for i in range(fold):
            print(self.dim)
            test_model=clf.best_estimator_
            scores.extend(list(cross_val_score(test_model,data,labels,cv=5)))
```

```
        scores=np.array(scores)
        print('out')
        return ((scores.mean(),scores.std()**2),(clf.cv_results_['mean_test_score'][best_index],cl
        
        
        
        
#%%
#
#test=find_best_svm(train1,test1)

data_sets=[[train1,test1],[std_train,std_test],[norm_train,norm_test]]
#data_sets=[[test1,test1],[std_test,std_test],[norm_test,norm_test]]
dim_choice=['none','Fisher','PCA']


file1=open('tree_results.txt','w')

i=0
for k in data_sets:
    if i==0:
        file1.write('no std:\n')
    elif i==1:
        file1.write('std:\n')
    else:
         file1.write('norm:\n')
    for j in dim_choice:
        print('we movin')
        file1.write('---'+j+':\n')
        tmp=find_best_tree(k[0],k[1],j)
        file1.write('All stats'+str(tmp.acc)+'\n')
        file1.write('train confusion matrix:'+str(tmp.train_conf)+'\n')
        file1.write('test confusion'+str(tmp.test_conf)+'\n')
        
    file1.write('------------------------------------------------------------'+'\n')
    i+=1

file1.close()

    #%%
#test_model=SVC(kernel='rbf')
#scores=(cross_val_score(test_model,train1[:,1:],train1[:,0],cv=5))
##
###
###
#
#test=SVC()
#clf=GridSearchCV(test,param_grid)
#clf.fit(train1[:,1:],train1[:,0])
##
##clf.best_estimator_.score(train1[:,1:],train1[:,0])


#test=train1[:,1:]
#label=train1[:,0]
#
```

```
#result=np.concatenate((label[:,np.newaxis],test),axis=1)
#print(result==train1)
#
#%%

#tmp_test=find_best_knn(train1,test1,dim='perm')
#print(train1_data==tmp_test.dim_transform(train1_data,tmp_test.perm[-1]))
```