```
!pip install pyreadr
!pip install PyDrive
```

```
import numpy as np
import os
import pandas as pd
import time
import xgboost as xgb
import pyreadr
import scipy.io as scio
from collections import OrderedDict
from google.colab import auth
from oauth2client.client import GoogleCredentials
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from scipy.io import loadmat
from scipy.spatial.distance import cdist
from sklearn import datasets
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import accuracy_score,classification_report
from sklearn.ensemble import BaggingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import scale
```

ADVANCED MODEL

Instruction

1. Upload training data file in the google drive
2. Get shareable link of the data file
3. Get file ID (the file ID can be obstained from the link.)
4. replace the file ID in corresponding code. (Detailed instruction also come with the code through

Part 0: set up control and work directories, extract paths.

```
####Authenticate the google drive account
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
```

```
drive = GoogleDrive(gauth)

from sklearn.model_selection import train_test_split, GridSearchCV #Perforing grid se
import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 12, 4
```

Part 1: Import Data

```
#####get the file shareable link = https://drive.google.com/open?id=1oliwM2-sH8CD_3q3
#####The file ID is the letter after "id=".

#####please replace the id of your file
download = drive.CreateFile({'id': '1oliwM2-sH8CD_3q3yUbLF836U9A1-Lc0'})
download.GetContentFile('train_set.zip')
!unzip train_set.zip

#####Run the code, and go to the URL in th output, enter the authorization code, done
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=

Enter your authorization code:
··········
Mounted at /content/drive
```

I. Our Advanced Model

We are using PCA with Bagging-SVM as our Advacned Model

Notation on These functions:

1. extract_mat():

   TAKES IN a list returned by a loadmat function.

   RETURN an array that have all the points in the mat file.

2. get_f():

   TAKES IN a direction that contains a *single* .mat file.

   RETURN an ndarray contains the pairwise euclidian distance between the coordinate contains i

3. feature_extraction():

   TAKES IN a direction that contains the direction that contains *all* the .mat file for the train.

   RETURNS a ndarray contains the train_x with features set as pairwise euclidian distance betwe contains in the .mat file.

4. f_pca():

   TAKES IN a ndarray contains all the train_x.

   RETURNS a ndarray contains decomposed x and the decompositon model.

5. BaggingSVM_w_pca():

   TAKES IN two ndarrays as train_x(without decomposition) and train_y.

   RETURNS the SVM-Bagging model trained with decomposed-train_x and train_y.

6. claim_possible_acc_BSVM():

   TAKES IN three arguments which is the direction that contains _all_the .mat file for x, the directi named *label.csv* that have a column named as emotion_idx as the train_y.

   RETURNS the possible accuracy of the Logistic-Bagging model.

```
def extract_mat(x):
    v = list(x.keys())[-1]
    return x[v]

def get_f(file_dir):
    '''Argument:
        file_dir: The whole direction contain the exact mat file

       Return:
        a np.array contains the featrues of single X'''
    a = extract_mat(loadmat(file_dir))
    b = cdist(a, a)
    r = b[np.triu_indices(b.shape[1], 1)].flatten()
    return r

def f_pca(x):
    my_pca = PCA(n_components = 130)
    new_X = my_pca.fit_transform(x)
    compo = sum(my_pca.explained_variance_ratio_)*100
    print(f'The Decomposition take up {compo: 0.4f}% Information of original Data')

    return new_X, my_pca

def feature_extraction(dir_x):
    if (dir_x[-1] != '/'):
        dir_x = dir_x + '/'
```

```
        fea_start = time.time()

        filenames = list(os.listdir(dir_x))
        filenames.sort()
        X = np.array(list(map(get_f, ((dir_x + i) for i in filenames))))

        fea_end = time.time()
        fea_time = fea_end - fea_start

        print('Feature Extraction Completed!')
        print(f'Feature Extraction Cost: {fea_time: 0.2f} Seconds')
        return X

    def BaggingSVM_w_pca(train_X, train_y):

        train_X, pca_mode = f_pca(train_X)

        start_SVM = time.time()
        S_svm = SVC(C = 0.1,
                    kernel = 'linear',
                    shrinking = True,
                    decision_function_shape = 'ovo')
        Bagg_SVM = BaggingClassifier(S_svm,
                                     n_estimators = 80,
                                     n_jobs = 5,
                                     bootstrap_features = True)
        Bagg_SVM.fit(train_X, train_y)
        end_SVM = time.time()

        Train_time = end_SVM - start_SVM
        print(f'The Time for train is: {Train_time: 0.2f} Seconds')
        return Bagg_SVM, pca_mode


    def claim_possible_acc_BSVM(X_path, y_path, n_iter = 1):
        X = feature_extraction(X_path)
        y = pd.read_csv(y_path).emotion_idx

        accs = []
        for i in range(n_iter):
            trainx, testx, trainy, testy = train_test_split(X, y, test_size = .2)
            model, pca_mode= BaggingSVM_w_pca(trainx, trainy)
            new_testx = pca_mode.transform(testx)
            testy_hat = model.predict(new_testx)
            accs.append(accuracy_score(testy, testy_hat))
        ret = np.mean(accs)*100
        return print(f'Our model should have about {ret: 0.4f}% accuracy')


    # This line can output the Claimed Accuracy
```

```
# You don't really need run it
claim_possible_acc_BSVM('train_set/points', 'train_set/label.csv', 15)
```

```
Feature Extraction Completed!
Feature Extraction Cost:  0.89 Seconds
The Decomposition take up  99.9079% Information of original Data
The Time for train is:  132.73 Seconds
The Decomposition take up  99.9084% Information of original Data
The Time for train is:  133.39 Seconds
The Decomposition take up  99.9082% Information of original Data
The Time for train is:  150.10 Seconds
The Decomposition take up  99.9053% Information of original Data
The Time for train is:  150.06 Seconds
The Decomposition take up  99.9091% Information of original Data
The Time for train is:  141.05 Seconds
The Decomposition take up  99.9090% Information of original Data
The Time for train is:  126.75 Seconds
The Decomposition take up  99.9083% Information of original Data
The Time for train is:  145.79 Seconds
The Decomposition take up  99.9084% Information of original Data
The Time for train is:  148.31 Seconds
The Decomposition take up  99.9078% Information of original Data
The Time for train is:  139.24 Seconds
The Decomposition take up  99.9133% Information of original Data
The Time for train is:  142.79 Seconds
The Decomposition take up  99.9082% Information of original Data
The Time for train is:  125.05 Seconds
The Decomposition take up  99.9081% Information of original Data
The Time for train is:  125.84 Seconds
The Decomposition take up  99.9074% Information of original Data
The Time for train is:  148.81 Seconds
The Decomposition take up  99.9096% Information of original Data
The Time for train is:  133.39 Seconds
The Decomposition take up  99.9078% Information of original Data
The Time for train is:  147.61 Seconds
Our model should have about  52.2533% accuracy
```

After doing some test, we can claim that Our Advanced Model would have 52.25% Accuracy.

The train time for our Advanced Model is about 110 seconds

You can use The Code below to train the model on the new 2500 data set

```
# Extract feature and do the train_test_split
X = feature_extraction('train_set/points')
y = pd.read_csv('train_set/label.csv').emotion_idx
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .2)
```

```
Feature Extraction Completed!
Feature Extraction Cost:  1.06 Seconds
```

```
# Train the model using X_train and y_train
advanced_model, pca_sub_model = BaggingSVM_w_pca(X_train, y_train)
```

```
# Test the model using X_test and y_test
X_test_decomp = pca_sub_model.transform(X_test)
y_test_hat = advanced_model.predict(X_test_decomp)
```

```
The Decomposition take up  99.9115% Information of original Data
The Time for train is:  166.45 Seconds
```

II. XGBOOST Model

Part 0: Feature Extration and Train/Test Split

```
##### Importing the fidusial points
import scipy.io as scio
from collections import OrderedDict
points_path = 'train_set/points'
points = [p for p in sorted(os.listdir(points_path))]
all_points = []
for p in points:
  poiFile = os.path.join(points_path, p)
  poi = scio.loadmat(poiFile)
  poi = OrderedDict(poi)
  all_points.append(poi.popitem()[1])
y = pd.read_csv('train_set/label.csv')['emotion_idx']

print('success')
```

```
success
```

```
##### Calculating pairwise distance
pair_dist = []
for i in range(len(all_points)):
  pair_dist.append(metrics.pairwise_distances(all_points[i])[np.triu_indices(78)])

##### Split train_set & test_set
points_train, points_test, y_train, y_test = train_test_split(pair_dist, y, random_st
print('success')
```

```
success
```

```
##### Feature Extration/Calculating pairwise distance and the time for feature extrat
import time

allpoints_train, allpoints_test, y_train, y_test = train_test_split(all_points, y, ra
print('success')
```

```
train_pair_dist = []
for i in range(len(allpoints_train)):
  pair_dist.append(metrics.pairwise_distances(allpoints_train[i])[np.triu_indices(78)

test_pair_dist = []
for i in range(len(allpoints_test)):
  pair_dist.append(metrics.pairwise_distances(allpoints_test[i])[np.triu_indices(78)]


start = time.time()
pair_dist = []
for i in range(len(all_points)):
  pair_dist.append(metrics.pairwise_distances(all_points[i])[np.triu_indices(78)])
finish = time.time()
print("Time on feature selection done in %0.3fs" % (finish-start))

start = time.time()
train_pair_dist = []
for i in range(len(allpoints_train)):
  pair_dist.append(metrics.pairwise_distances(allpoints_train[i])[np.triu_indices(78)
finish = time.time()
print("Time on feature selection training set done in %0.3fs" % (finish-start))

start = time.time()
test_pair_dist = []
for i in range(len(allpoints_test)):
  pair_dist.append(metrics.pairwise_distances(allpoints_test[i])[np.triu_indices(78)]
finish = time.time()
print("Time on feature selection test set done in %0.3fs" % (finish-start))
```

Part 1: XGBoost Training

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from xgboost.sklearn import XGBClassifier
import time
import numpy as np


def modelfit(alg, dtrain, predictors, cv_folds=10):
  #Fit the algorithm on the data
  alg.fit(dtrain, predictors)

  #Predict training set:
  dtrain_predictions = alg.predict(dtrain)
```

```
  dtrain_predictions = alg.predict(dtrain)
  dtrain_predprob = alg.predict_proba(dtrain)[:,1]

  #Print model report:
  print("\nModel Report")
  print("Accuracy : %.4g" % metrics.accuracy_score(predictors, dtrain_predictions))
```

Part 2:XGBoost Default setting

```
####XGBOOST base model with default setting
start = time.time()
xgb_base = XGBClassifier(
 objective= 'multi:softmax',
 num_class= 22,
 seed=1000)

modelfit(xgb_base, np.array(points_train), np.array(y_train))
finish = time.time()
print("Prediction on train_set done in %0.3fs" % (finish-start))
```

```
start = time.time()
preds = xgb_base.predict(points_test)
acc_preds = metrics.accuracy_score(preds, y_test)
finish = time.time()
print("Prediction on test_set done in %0.3fs" % (finish - start))
print("Test_set accurarcy is %0.3f" %acc_preds)
```

Tuning Process(comment it out because the process is time comsuming)

```
####tune hyperparameter

## tune the max_depth and min_child_weight parameter
# param_test1 = {
#  'max_depth': range(4,5,6),
#  'min_child_weight': range(4,5,6)
# }
# gsearch1 = GridSearchCV(xgb_base, param_grid = param_test1, scoring ='accuracy', cv
# gsearch1.fit(np.array(points_train), np.array(y_train))
# best_parameters1 = gsearch1.best_estimator_.get_params()
# for param_name in sorted(param_test1.keys()):
```

```
#     print("\t%s: %r" % (param_name, best_parameters1[param_name]))

## use the best_parameter above to xgb2
# start = time.time()
# xgb2 = XGBClassifier(
#  objective= 'multi:softmax',
#  num_class= 22,
#  max_depth=4,
#  min_child_weight=4,
#  seed=1000)
# modelfit(xgb2, np.array(points_train), np.array(y_train))
# finish = time.time()
# print("Prediction on train_set done in %0.3fs" % (finish-start))
# start = time.time()
# preds = xgb2.predict(points_test)
# acc_pred = metrics.accuracy_score(preds, y_test)
# finish = time.time()
# print("Prediction on test_set done in %0.3fs" % (finish - start))
# print("Test_set accurarcy is %0.3f" %acc_pred)

## However, the accuracy is lower than that of the base model, so we keep the same pa
## as before, and tune other parameters.

## tune the gamma parameter
# param_test2 = {
#  'gamma':[i/10.0 for i in range(0,5)]
# }
# gsearch2 = GridSearchCV(xgb1, param_grid = param_test2, scoring ='accuracy', cv = 5
# gsearch2.fit(np.array(points_train), np.array(y_train))
# best_parameters2 = gsearch2.best_estimator_.get_params()
# for param_name in sorted(param_test2.keys()):
#     print("\t%s: %r" % (param_name, best_parameters2[param_name]))

# start = time.time()
# xgb3 = XGBClassifier(
#  objective= 'multi:softmax',
#  num_class= 22,
#  gamma=0.4,
#  seed=1000)

# modelfit(xgb3, np.array(points_train), np.array(y_train))
# finish = time.time()
# print("Prediction on train_set done in %0.3fs" % (finish-start))

# start = time.time()
# preds = xgb3.predict(points_test)
# acc_pred = metrics.accuracy_score(preds, y_test)
# finish = time.time()
# print("Prediction on test_set done in %0.3fs" % (finish - start))
# print("Test_set accurarcy is %0.3f" %acc_pred)

## However, the accuracy is lower than that of the base model, so we keep the same pa
```

```
## as before, and tune other parameters.

## tune the subsample and colsample_bytree parameters
#param_test = {
 #'subsample':[i/10.0 for i in range(6,10)],
 #'colsample_bytree':[i/10.0 for i in range(6,10)]
#}
#gsearch = GridSearchCV(xgb_base, param_grid = param_test, scoring ='accuracy', cv =
#gsearch.fit(np.array(points_train), np.array(y_train))
#best_parameters = gsearch.best_estimator_.get_params()
#for param_name in sorted(param_test.keys()):
    #print("\t%s: %r" % (param_name, best_parameters[param_name]))

# start = time.time()
# xgb4 = XGBClassifier(
#  objective = 'multi:softmax',
#  num_class = 22,
#  seed = 1000,
#  colsample_bytree=0.6,
#  subsample=0.7)

# modelfit(xgb4, np.array(points_train), np.array(y_train))
# finish = time.time()
# print("Prediction on train_set done in %0.3fs" % (finish-start))

# start = time.time()
# preds = xgb4.predict(points_test)
# acc_pred = metrics.accuracy_score(preds, y_test)
# finish = time.time()
# print("Prediction on test_set done in %0.3fs" % (finish - start))
# print("Test_set accurarcy is %0.3f" %acc_pred)

## We use the best parameters above because the accuracy increases and the prediction
## decreases. Then, we tune other parameter based on the xgb4.

##tune reg_alpha parameter
#param_test4 = {
#  'reg_alpha':[1e-5, 1e-2, 0.1, 1, 100]
# }
# gsearch4 = GridSearchCV(xgb4, param_grid = param_test4, scoring ='accuracy', cv = 5
# gsearch4.fit(np.array(points_train), np.array(y_train))
# best_parameters4 = gsearch4.best_estimator_.get_params()
# for param_name in sorted(param_test4.keys()):
#     print("\t%s: %r" % (param_name, best_parameters4[param_name]))

# start = time.time()
# xgb5 = XGBClassifier(
#  objective= 'multi:softmax',
#  num_class= 22,
#  seed=1000,
#  colsample_bytree=0.7,
#  subsample=0.6,
```

```
#  reg_alpha=1)

# modelfit(xgb5, np.array(points_train), np.array(y_train))
# finish = time.time()
# print("Prediction on train_set done in %0.3fs" % (finish-start))

# start = time.time()
# preds = xgb5.predict(points_test)
# acc_pred = metrics.accuracy_score(preds, y_test)
# finish = time.time()
# print("Prediction on test_set done in %0.3fs" % (finish - start))
# print("Test_set accurarcy is %0.3f" %acc_pred)

##Since the best parameter of reg_alpha=1e-05, and the accuracy is so close to that o
##model, we decide to use the xgb5 as our final model.
```

Part 3: The improved XGboost model after tuning the parameters

```
####XGBOOSTING improved model
start = time.time()
xgb5 = XGBClassifier(
 objective= 'multi:softmax',
 num_class= 22,
 seed=1000,
 colsample_bytree=0.6,
 subsample=0.7,
 reg_alpha=1)

modelfit(xgb5, np.array(points_train), np.array(y_train))
finish = time.time()
print("Prediction on train_set done in %0.3fs" % (finish-start))
```

```
start = time.time()
preds = xgb5.predict(points_test)
acc_pred = metrics.accuracy_score(preds, y_test)
finish = time.time()
print("Prediction on test_set done in %0.3fs" % (finish - start))
print("Test_set accurarcy is %0.3f" %acc_pred)
```

III. BAGGING-LOG MODEL

Notation on These functions:

1. extract_mat():

   TAKES IN a list returned by a loadmat function.

   RETURN an array that have all the points in the mat file.

2. get_f():

   TAKES IN a direction that contains a *single* .mat file.

   RETURN an ndarray contains the pairwise euclidian distance between the coordinate contains i

3. feature_extraction():

   TAKES IN a direction that contains the direction that contains *all* the .mat file for the train.

   RETURNS a ndarray contains the train_x with features set as pairwise euclidian distance betwe contains in the .mat file.

4. f_pca():

   TAKES IN a ndarray contains all the train_x.

   RETURNS a ndarray contains decomposed x and the decompositon model.

5. BaggingLR_w_pca():

   TAKES IN two ndarrays as train_x(without decomposition) and train_y.

   RETURNS the Logistic-Bagging model trained with decomposed-train_x and train_y.

6. claim_possible_acc_BL():

   TAKES IN three arguments which is the direction that contains _all_the .mat file for x, the directi named *label.csv* that have a column named as emotion_idx as the train_y.

   RETURNS the possible accuracy of the Logistic-Bagging model.

```
def extract_mat(x):
    v = list(x.keys())[-1]
    return x[v]

def get_f(file_dir):
    '''Argument:
        file_dir: The whole direction contain the exact mat file

       Return:
        a np.array contains the featrues of single X'''
    a = extract_mat(loadmat(file_dir))
    b = cdist(a, a)
```

```
    r = b[np.triu_indices(b.shape[1], 1)].flatten()
    return r

def feature_extraction(dir_x):
    if (dir_x[-1] != '/'):
        dir_x = dir_x + '/'

    fea_start = time.time()

    filenames = list(os.listdir(dir_x))
    filenames.sort()
    X = np.array(list(map(get_f, ((dir_x + i) for i in filenames))))

    fea_end = time.time()
    fea_time = fea_end - fea_start

    print('Feature Extraction Completed!')
    print(f'Feature Extraction Cost: {fea_time: 0.2f} Seconds')
    return X

def f_pca(x):
    my_pca = PCA(n_components = 130)
    new_X = my_pca.fit_transform(x)
    compo = sum(my_pca.explained_variance_ratio_)*100
    print(f'The Decomposition take up {compo: 0.2f}% Information of original Data')

    return new_X, my_pca

def BaggingLR_w_pca(train_X, train_y):

    train_X, pca_mode = f_pca(train_X)

    start_lr = time.time()
    lr = LogisticRegression(C = 1,
                            penalty = 'l2',
                            fit_intercept = False)
    Bag_lr = BaggingClassifier(lr,
                               n_estimators = 70,
                               n_jobs = 5,
                               bootstrap_features = True,
                               verbose = 7)
    Bag_lr.fit(train_X, train_y)
    end_lr = time.time()

    Train_time = end_lr - start_lr
    print(f'The Time for train is: {Train_time: 0.2f} Seconds')
    return Bag_lr, pca_mode


def claim_possible_acc_BL(X_path, y_path, n_iter = 1):
    X = feature_extraction(X_path)
    y = pd.read_csv(y_path).emotion_idx
```

```
    accs = []
    for i in range(n_iter):
        trainx, testx, trainy, testy = train_test_split(X, y, test_size = .2)
        model, pca_mode= BaggingLR_w_pca(trainx, trainy)
        new_testx = pca_mode.transform(testx)
        testy_hat = model.predict(new_testx)
        accs.append(accuracy_score(testy, testy_hat))
    ret = np.mean(accs)*100
    return print(f'The Bagging-Logistic model should have about {ret: 0.4f}% accuracy
```

```
claim_possible_acc_BL('train_set/points', 'train_set/label.csv',10)
```

The accuracy of Bagging-Logistic model may be a bit higher(53.6%) than our advanced model but aft model is not as stable as our advanced model.

reference: https://www.cnblogs.com/wj-1314/p/10422159.html