Step 0: set work directories, extract paths, summarize

```
import os
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
import scipy.io as scio
from collections import OrderedDict
import numpy as np
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale
from time import time


auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
print("Auth Success")
```

```
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-
  * https://github.com/tensorflow/addons
  * https://github.com/tensorflow/io (for I/O related ops)
If you depend on functionality not listed there, please file an issue.

Auth Success
```

First upload train_set.zip in data folder to google drive, and please replace the *id* of the zip file. (By the train_set.zip file in the google drive and selecting "Get Sharable Link", you can get an ID.)

```
download = drive.CreateFile({'id': '1VnzsmUSgP_IqXvlgWMCVPpbPW665fjaI'}) #please replace the id
download.GetContentFile('train_set.zip')
!unzip train_set.zip
```

Step 1: set up controls for evaluation experiments.

```
import pandas as pd
import numpy as np
import time
from sklearn.ensemble import GradientBoostingClassifier  #GBM algorithm
from sklearn import metrics
from sklearn.model_selection import cross_val_score
```

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

Step 2: import data 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
```

Step 3: construct features and responses

```
########## 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_state=42, tes
print('success')
```

```
success
```

Step 4: Train a classification model with training features and responses

GBM & CV (Baseline Model)

```
start = time.time()
gbm0 = GradientBoostingClassifier(random_state=42, max_depth=1)
gbm0.fit(points_train, y_train)
```

```
finish = time.time()
print("Time：%f s" %(finish-start))
pred = gbm0.predict(points_test)
print("Baseline GBM Accuracy : %.4g" % metrics.accuracy_score(y_test, pred))
```

```
Time：500.429408 s
Baseline GBM Accuracy : 0.446
```

The default setting of GBM model gives accuracy 44.6%, next I'll tune the model in the following or n_estimators, min_samples_leaf, subsample, learning_rate.

Tuning Process

```
param_test1 = {'n_estimators':range(100,301,10)}
gsearch1 = GridSearchCV(estimator = GradientBoostingClassifier(learning_rate=0.1, min_samples_sp
                        param_grid = param_test1, scoring='accuracy',n_jobs=4,iid=False, cv=5)
gsearch1.fit(np.array(points_train), np.array(y_train))
gsearch1.best_estimator_, gsearch1.best_params_, gsearch1.best_score_
```

```
(GradientBoostingClassifier(criterion='friedman_mse', init=None,
                            learning_rate=0.1, loss='deviance', max_depth=1,
                            max_features='sqrt', max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None
                            min_samples_leaf=50, min_samples_split=500,
                            min_weight_fraction_leaf=0.0, n_estimators=230,
                            n_iter_no_change=None, presort='auto',
                            random_state=42, subsample=0.8, tol=0.0001,
                            validation_fraction=0.1, verbose=0,
                            warm_start=False),
 {'n_estimators': 230},
 0.45619765122118566)
```

```
param_test2 = {'min_samples_leaf':range(30,101,10)}
gsearch2 = GridSearchCV(estimator = GradientBoostingClassifier(learning_rate=0.1, n_estimators=2
                        param_grid = param_test2, scoring='accuracy',n_jobs=4,iid=False, cv=5)
gsearch2.fit(points_train, y_train)
gsearch2.best_estimator_, gsearch2.best_params_, gsearch2.best_score_
```

```
(GradientBoostingClassifier(criterion='friedman_mse', init=None,
                            learning_rate=0.1, loss='deviance', max_depth=1,
                            max_features='sqrt', max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None
                            min_samples_leaf=90, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, n_estimators=230,
                            n_iter_no_change=None, presort='auto',
                            random_state=42, subsample=0.8, tol=0.0001,
                            validation_fraction=0.1, verbose=0,
                            warm_start=False),
 {'min_samples_leaf': 90},
 0.4602770254674405)
```

After cross validating, the GBM model we choose will be learning_rate=0.01, n_estimators=900, max_depth=1,min_samples_leaf=90, subsample=0.85

Prediction

```
gbm_trained = GradientBoostingClassifier(
    random_state=42,
    learning_rate=0.1,
    n_estimators=230,
    max_depth=1,
    min_samples_leaf=90,
    subsample=0.85)
gbm_trained.fit(points_train, y_train)
start = time.time()
pred = gbm_trained.predict(points_train)
finish = time.time()
print("Time：%f s" %(finish-start))
print("GBM Accuracy on training data: %.4g" % metrics.accuracy_score(y_train, pred))

start = time.time()
pred = gbm_trained.predict(points_test)
finish = time.time()
print("Time：%f s" %(finish-start))
print("GBM Accuracy on test data: %.4g" % metrics.accuracy_score(y_test, pred))
```

```
Time：0.173605 s
GBM Accuracy on training data: 0.965
Time：0.031174 s
GBM Accuracy on test data: 0.468
```

```
start = time.time()
gbm_trained = GradientBoostingClassifier(
    random_state=42,
    learning_rate=0.1,
    n_estimators=230,
    max_depth=1,
    min_samples_leaf=90,
    subsample=0.85)
gbm_trained.fit(pair_dist, y)
finish = time.time()
print("Time：%f s" %(finish-start))
```

```
Time：1165.229694 s
```

```
from sklearn.externals import joblib
joblib.dump(gbm_trained, "gbm_trained.m")
```

```
['gbm_trained.m']
```

SVM & CV

```
########## Scaling datasets ##########
points_train_np = np.array(points_train)
points_test_np = np.array(points_test)
y_train_np = np.array(y_train)
y_test_np = np.array(y_test)
```

```
points_train_scale = scale(points_train_np)
points_test_scale = scale(points_test_np)


########## PCA Modules ###########

#n_comp = 1000
#pca = PCA(n_components=n_comp, svd_solver='randomized',
#           whiten=True).fit(points_train_np)
#points_train_pca = pca.transform(points_train_np)
#points_test_pca = pca.transform(points_test_np)
```

Tuning Process

```
###### Tunning Modules #########
tunning = False
t0 = time.time()

if tunning:
  param_grid = {'C': [1,5,10],
              'gamma': [0.00001,0.0001,0.01], }
  clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'),
                   param_grid, cv=5, iid=False)
  clf = clf.fit(points_train_scale, y_train)
  print("done in %0.3fs" % (time.time() - t0))
  print("Best estimator found by grid search:")
  print(clf.best_estimator_)
```

Training Model

```
########## Training Modules ##########
###### if tunning is true, there is no need to train the model again ########
t0 = time.time()
clf = SVC(kernel="rbf", class_weight="balanced",C=10,gamma=0.0001)
clf = clf.fit(points_train_scale, y_train)
print("Training done in %0.3fs" %(time.time() -t0))
```

```
Training done in 16.100s
```

Prediction

```
######## Prediction on test_set #########
t0 = time.time()
y_pred = clf.predict(points_test_scale)
acc_pred = np.sum(y_pred == y_test)/y_test.shape[0]
print("Prediction on test_set done in %0.3fs" % (time.time() - t0))
print("Test_set accurarcy is %0.3f" %acc_pred)
```

```
Prediction on test_set done in 5.152s
Test_set accurarcy is 0.512
```

```
######## Prediction on train_set #########
t0 = time.time()
```

```
y_pred_train = clf.predict(points_train_scale)
acc_pred_train = np.sum(y_pred_train == y_train)/y_train.shape[0]
print("Prediction on train_set done in %0.3fs" % (time.time() - t0))
print("Train_set accurarcy is %0.3f" %acc_pred_train)
```

```
Prediction on train_set done in 20.572s
Train_set accurarcy is 0.821
```

Train on 2500 Images

```
########### Using 2500 images as train_set ##########
pair_dist_scale = scale(pair_dist)
t0 = time.time()
clf = SVC(kernel="rbf", class_weight="balanced",C=10,gamma=0.0001)
clf = clf.fit(pair_dist_scale, y)
print("Training on 2500 images takes %0.3fs" %(time.time() -t0))
```

```
Training on 2500 images takes 23.467s
```

```
t0 = time.time()
y_pred_train_2500 = clf.predict(pair_dist_scale)
acc_pred_train_2500 = np.sum(y_pred_train_2500 == y)/y.shape[0]
print("Prediction on 2500 images takes %0.3fs" % (time.time() - t0))
print("Training accurarcy is %0.3f" %acc_pred_train_2500)
```

```
Prediction on 2500 images takes 32.248s
Training accurarcy is 0.798
```

```
joblib.dump(clf, "svm_final.m")
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/__init__.py:15
  warnings.warn(msg, category=DeprecationWarning)
['svm_final.m']
```

Xgboost & CV

```
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_predprob = alg.predict_proba(dtrain)[:,1]

  #Print model report:
```

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


start = time.time()
xgb1 = XGBClassifier(
 objective= 'multi:softmax',
 num_class= 23,
 seed=27)

modelfit(xgb1, np.array(points_train), np.array(y_train))
finish = time.time()
print("Prediction on train_set done in %0.3fs" % (finish-start))
```

```
Model Report
Accuracy : 1
Prediction on train_set done in 782.057s
```

```
start = time.time()
preds = xgb1.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)
```

```
Prediction on test_set done in 0.492s
Test_set accurarcy is 0.482
```

We can see that before we change any paramters, the accuracy rate of xgboost is 48.2%,which is b
baseline. The speed is better as well. Hence, we think we can consider this method.

Tuning Process

As it takes a lot of time to fit the model, we only tune the most important parameters.

As I tried to change many parameters in the model, I found 'n_estimators' and 'min_child_weight' ir
performance a lot.

Step1: n_estimators

```
start = time.time()
xgb2 = XGBClassifier(
 n_estimators = 1000,
 objective= 'multi:softmax',
 num_class= 23,
 max_depth =5,
 min_child_weight =1,
 nthread =4,
 subsample = 0.8,
 colsample_bytree  = 0.8,
 scale_pos_weight = 1,
 seed=27)
```

```
modelfit(xgb2, np.array(points_train), np.array(y_train))
finish = time.time()
print("Time：%f s" %(finish-start))
print("Test_set accurarcy is %0.3f" %acc_pred)
```

```
Model Report
Accuracy : 1
Time：800.118329 s
```

```
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)
```

```
Prediction on test_set done in 0.634s
Test_set accurarcy is 0.528
```

We changed n_estimators to 1000. Then the accuracy rate of the model increased a lot and the ru
not increase much. Hence, we tuned n_estimators to 1000. And we tuned 'subsample' and 'colsam
more reasonable value.

Step2: Tune min_child_weight

```
param_test1 = {
 'min_child_weight':range(1,6,2)
}
gsearch1 = GridSearchCV(estimator = XGBClassifier(n_estimators = 1000,
 objective= 'multi:softmax',
 num_class= 23,
 max_depth =5,
 min_child_weight =1,
 nthread =4,
 subsample = 0.8,
 colsample_bytree  = 0.8,
 scale_pos_weight = 1,
 seed=27),
param_grid = param_test1, scoring = 'accuracy',iid=False, cv=2)
gsearch1.fit(np.array(points_train), np.array(y_train))
gsearch1.best_params_, gsearch1.best_score_, gsearch1.cv_results_
```

```
({'min_child_weight': 1},
 0.45233928421423175,
 {'mean_fit_time': array([302.53986263, 267.62051725, 259.8642416 ]),
  'mean_score_time': array([0.34943461, 0.34187734, 0.33939075]),
  'mean_test_score': array([0.45233928, 0.44191391, 0.444447  ]),
  'param_min_child_weight': masked_array(data=[1, 3, 5],
               mask=[False, False, False],
         fill_value='?',
              dtype=object),
  'params': [{'min_child_weight': 1},
   {'min_child_weight': 3},
   {'min_child_weight': 5}],
  'rank_test_score': array([1, 3, 2], dtype=int32),
  'split0_test_score': array([0.47912525, 0.45626243, 0.45328032]),
  'split1_test_score': array([0.42555332, 0.42756539, 0.43561368]),
  'std_fit_time': array([2.66214442, 4.87970066, 2.17809272]),
  'std_score_time': array([0.00725555, 0.01867068, 0.00683665]),
  'std_test_score': array([0.02678596, 0.01434852, 0.00883332])})
```

We found that 'min_child_weight' = 1 is the best. And we get our final xgboost model.

Training Model

```
start = time.time()
xgb_final = XGBClassifier(
 n_estimators = 1000,
 objective= 'multi:softmax',
 num_class= 23,
 max_depth =5,
 min_child_weight =1,
 nthread =4,
 subsample = 0.8,
 colsample_bytree  = 0.8,
 scale_pos_weight = 1,
 seed=27)

modelfit(xgb_final, np.array(points_train), np.array(y_train))
finish = time.time()
print("Time：%f s" %(finish-start))
```

```
Model Report
Accuracy : 1
Time：807.873069 s
```

Prediction

```
start = time.time()
preds = xgb_final.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)
```

```
Prediction on test_set done in 0.629s
Test_set accurarcy is 0.528
```

```
start = time.time()
modelfit(xgb_final, np.array(pair_dist), np.array(y))
finish = time.time()
print("Time：%f s" %(finish-start))
```

```
Model Report
Accuracy : 1
Time：1133.331869 s
```

```
joblib.dump(xgb_final, "xgb_final.m")
```

Step 5: Run test on test images

```
## zip
download = drive.CreateFile({'id': '1I8sqauDEV_iNB--2_xu7ChAsVdPXjepa'}) #please replace the id
download.GetContentFile('test_set_sec1.zip')
!unzip test_set_sec1.zip
```

```
## points
points_path_final = 'test_set_sec1/points'
points_final = [p for p in sorted(os.listdir(points_path_final))]
all_points_final = []
for p in points_final:
  poiFile_final = os.path.join(points_path_final, p)
  poi_final = scio.loadmat(poiFile_final)
  poi_final = OrderedDict(poi_final)
  all_points_final.append(poi_final.popitem()[1])

print('success')
```

```
success
```

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

GBM

```
gbm_final=joblib.load("gbm_trained.m")

start_gbm = time.time()
pred_gbm_final = gbm_final.predict(pair_dist_final)
finish_gbm = time.time()
print(pred_gbm_final)
```

```
[22  7  9 ...  4 20 19]
```

```
gbm = np.asarray(pred_gbm_final)
np.savetxt("gbm_final_result.csv", gbm, delimiter=",")
```

XGBOOST

```
xgb_final=joblib.load("xgb_final.m")

start_xgb = time.time()
preds_xgb_final = xgb_final.predict(pair_dist_final)
finish_xgb = time.time()
print(preds_xgb_final)
```

```
[22  7  9 ...  4 19 19]
```

```
xgb = np.asarray(preds_xgb_final)
np.savetxt("xgb_final_result.csv", xgb, delimiter=",")
```

SVM

```
########## Scaling datasets ##########
points_test_np_final = np.array(pair_dist_final)
points_test_scale_final = scale(points_test_np_final)
```

```
svm_final=joblib.load("svm_final.m")
start_svm = time.time()
y_pred_svm_final = svm_final.predict(points_test_scale_final)
finish_svm = time.time()
print(y_pred_svm_final)
```

```
[22  7  9 ...  4 19 21]
```

```
svm = np.asarray(y_pred_svm_final)
np.savetxt("svm_final_result.csv", svm, delimiter=",")
```

Summarize Running Time

```
print("Time：%f s" %(finish_gbm-start_gbm)) # GBM
print("Time：%f s" %(finish_xgb-start_xgb)) # XGBOOST
print("Time：%f s" %(finish_svm-start_svm)) # SVM
```

```
Time：0.358157 s
Time：2.750383 s
Time：32.181925 s
```