Predictive Modeling - Image Analysis

Author: Chairuna, Marsya; Cheng, Lu; Heagy, David; Jiao, Hanbo; Pan, Zhongtian

Initial Settings

Before we start, we will need to install several packages required to run the modeling, e.g., pandas, numpy, sklearn, tensorflow, keras, etc.

In [ ]:

```
# pip install xgboost
```

In [ ]:

```
# conda install tensorflow
```

In [ ]:

```
# conda install keras
```

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
```

Top

In [5]:

```
import os
import pandas as pd
import numpy as np
import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.externals import joblib
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
import xgboost
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from scipy.io import loadmat
from sklearn.preprocessing import StandardScaler
import scipy
```

```
C:\Users\marsy\anaconda3\lib\site-packages\sklearn\externals\joblib\__init__.py:15:
FutureWarning: sklearn.externals.joblib is deprecated in 0.21 and will be removed in
0.23. Please import this functionality directly from joblib, which can be installed
with: pip install joblib. If this warning is raised when loading pickled models, you
may need to re-serialize those models with scikit-learn 0.21+.
  warnings.warn(msg, category=FutureWarning)
```

1. Feature Extraction and Splitting Test and Train Dataset

Type *Markdown* and LaTeX: α^2

In the first part, we will conduct feature extraction and split the dataset into test and train set for training and testing the accuracy.

Load the data

In [6]:

```
filepath=os.path.dirname(os.path.dirname(os.path.realpath("main.ipynb")))
# ...\\Spring2020-Project3-group2
os.chdir(filepath)
```

Top

In [7]:

```
filepath
```

Out[7]:

```
'C:\\Users\\marsy\\Documents\\GitHub\\Spring2020-Project3-group2'
```

Note: If the above output is not correct. Please enter the filepath below to '..../Spring2020-Project3-group2/' manually. Thank you.

In [2]:

```
filepath = 'C:/Users/marsy/Documents/GitHub/Spring2020-Project3-group2/'
#####################   Modify above path   ########################
```

Create the path for train and test dataset.

In [6]:

```
os.chdir(filepath)

# testpath=filepath+'/data/test_set/'
trainpath=filepath+'/data/train_set/'

# test_image_dir = testpath + "images/"
# test_pt_dir = testpath + "points/"
train_image_dir = trainpath + "images/"
train_pt_dir = trainpath + "points/"
```

Calculate the pairwise distance

We will use sklearn to calculate the pairwise distance between the fiducial points. We will use the pairwise distance as features in our model.

In [7]:

```
import sklearn.metrics.pairwise
def pairwise_dist(vec):
    dist  = sklearn.metrics.pairwise_distances(vec, metric='euclidean')
    np.fill_diagonal(dist, np.nan)
    return dist
def feature(fiducial_pt_list,index):
    pairwise_dist_feature = pairwise_dist(fiducial_pt_list[index]).flatten()
    pairwise_dist_feature = pairwise_dist_feature[~np.isnan(pairwise_dist_feature)]
    return pairwise_dist_feature
```

Top

In [8]:

```
f0 = time.time()
dataDir = train_pt_dir
fiducial_pt_list = []
filelist = []
for file in os.listdir(dataDir):
    filelist.append(file)
filelist.sort()
for file in filelist:
    fiducial_pt_list.append(scipy.io.loadmat(dataDir+file))
    l = []
for i in range(len(fiducial_pt_list)):
    if 'faceCoordinatesUnwarped' in fiducial_pt_list[i].keys():
        l.append(fiducial_pt_list[i]['faceCoordinatesUnwarped'])
    else:
        l.append(fiducial_pt_list[i]['faceCoordinates2'])

fiducial_pt_list = l


X = pd.DataFrame(np.zeros((2500, 6006)))
for i in range(2500):
    X.iloc[i,:] = np.round(feature(fiducial_pt_list, i).flatten(), 0)
y =pd.read_csv(trainpath+'label.csv')['emotion_idx']
f1 = time.time()-f0
```

Here, we can observe that the time taken to extract the feature is ~2 seconds.

In [10]:

```
print("Feature Extraction time: %0.3fs" % (f1))
```

```
Feature Extraction time: 32.368s
```

Before implementing the machine learning technique, we wil use StandardScaler to transform our data such that its distribution will have a mean value 0 and standard deviation of 1.

In [11]:

```
scaler = StandardScaler()
X = scaler.fit_transform(X)

train_x_dis,test_x_dis,train_y_dis,test_y_dis=train_test_split(X,y,test_size=0.2,random_state=3662)
```

Model Results

Firstly, we use the baseline model GBM to compute the accuracy. For the advance model, we experiment with other 7 models as follows:

- (1) KNN
- (2) improved GBM
- (3) XGBoost
- (4) RandomForest
- (5) Logistic Regression

Top

- (6) Support Vector Machine (SVM)
- (7) MLP Classifier

For all the candidate models, we observe the claimed accuracy, training time, and testing time. We will select the best model based on these three performance parameters.

In [61]:

```
data = {'Model':['Bseline model:GBM','KNN', 'Improved GBM', 'XGboost','RandomForestClassifier','Logi
        'Claimed Accuracy':['41.92%', '30.36%', '43.32%','47.12%',  '45.48%', '54.00%', '55.20%', '5
        'Training Time/s':['472.039s', '0.481s', '1024.112s','129.966s',  '7.565s', '37.204s', '12.0
       'Testing Time/s':['0.023s', '10.944s', '0.034s','0.213s',  ' 0.035', '0.009s', '0.001s', '5.6
pd.DataFrame(data)
```

Out[61]:

	Model	Claimed Accuracy	Training Time/s	Testing Time/s
0	Bseline model:GBM	41.92%	472.039s	0.023s
1	KNN	30.36%	0.481s	10.944s
2	Improved GBM	43.32%	1024.112s	0.034s
3	XGboost	47.12%	129.966s	0.213s
4	RandomForestClassifier	45.48%	7.565s	0.035
5	LogisticRegression	54.00%	37.204s	0.009s
6	LogisticRegression with PCA	55.20%	12.074s	0.001s
7	SVM	50.04%	20.159s	5.694s
8	MLPClassifier	49.44%	328.237s	0.202s
9	Final model: VotingClassifier	54.32%	492.865s	6.098s

Baseline model - GBM (Claimed accuracy: 41.92%)

The baseline model we used is Boosted Decision Stumps.

In [14]:

```
gbm_baseline= GradientBoostingClassifier(n_estimators=100  , max_depth= 1,learning_rate=0.1)
```

Top

In [15]:

```
model=gbm_baseline
t0 = time.time()
model.fit(train_x_dis,train_y_dis)
t1 = time.time()
training_acc=model.score(train_x_dis,train_y_dis)
t2 = time.time()
pred=model.predict(test_x_dis)
t3 = time.time()
testing_acc=model.score(test_x_dis,test_y_dis)

print("Model fit time: %0.3fs; " % (t1-t0))
print("Training Accuracy : %0.1f%% %0.3fs" % (training_acc*100,t2-t1))
print("Testing Accuracy : %0.1f%% %0.3fs" % (testing_acc*100,t3-t2))
```

```
Model fit time: 472.039s;
Training Accuracy : 78.1% 0.115s
Testing Accuracy : 40.4% 0.023s
```

In [16]:

```
print(classification_report(test_y_dis,pred))
```

```
              precision    recall  f1-score   support

           1       0.50      0.67      0.57        24
           2       0.65      0.71      0.68        24
           3       0.34      0.36      0.35        28
           4       0.23      0.41      0.30        17
           5       0.58      0.44      0.50        25
           6       0.25      0.20      0.22        20
           7       0.50      0.31      0.38        26
           8       0.66      0.86      0.75        22
           9       0.56      0.47      0.51        19
          10       0.50      0.26      0.34        31
          11       0.40      0.45      0.43        22
          12       0.25      0.29      0.27        21
          13       0.31      0.15      0.21        26
          14       0.47      0.74      0.57        19
          15       0.29      0.45      0.36        11
          16       0.60      0.62      0.61        24
          17       0.36      0.40      0.38        30
          18       0.39      0.30      0.34        23
          19       0.21      0.29      0.24        17
          20       0.13      0.14      0.13        22
          21       0.30      0.35      0.32        23
          22       0.33      0.15      0.21        26

    accuracy                           0.40       500
   macro avg       0.40      0.41      0.39       500
weighted avg       0.41      0.40      0.39       500
```

Advanced Model

Top

We will then observe the performance of the candidate advance models as previously stated.

1. KNN (Claimed accuracy: 30.36%)

We then proceed to run the first model using KNN. However, we decide not to proceed further with this model because the accuracy is even less than the baseline model, which is not acceptable.

In [13]:

```
knn=KNeighborsClassifier(n_neighbors=24)
```

In [14]:

```
model=knn
t0 = time.time()
model.fit(train_x_dis,train_y_dis)
t1 = time.time()
training_acc=model.score(train_x_dis,train_y_dis)
t2 = time.time()
pred=model.predict(test_x_dis)
t3 = time.time()
testing_acc=model.score(test_x_dis,test_y_dis)

print("Model fit time: %0.3fs; " % (t1-t0))
print("Training Accuracy : %0.1f%% %0.3fs" % (training_acc*100,t2-t1))
print("Testing Accuracy : %0.1f%% %0.3fs" % (testing_acc*100,t3-t2))
```

```
Model fit time: 4.069s;
Training Accuracy : 42.4% 95.965s
Testing Accuracy : 30.4% 23.103s
```

2. Improved GBM (Claimed accuracy: 43.32%)

Secondly, We try to improve our baseline model by tuning one of the hyperparameter: increasing the "max_depth". We observed an improvement in the accuracy. However, the model fit time is significantly longer (472.039s vs. 1024.112s).

In [19]:

```
gbm_improved= GradientBoostingClassifier(n_estimators=100  , max_depth= 2,learning_rate=0.1)
```

Top

In [20]:

```
model=gbm_improved
t0 = time.time()
model.fit(train_x_dis,train_y_dis)
t1 = time.time()
training_acc=model.score(train_x_dis,train_y_dis)
t2 = time.time()
pred=model.predict(test_x_dis)
t3 = time.time()
testing_acc=model.score(test_x_dis,test_y_dis)

print("Model fit time: %0.3fs; " % (t1-t0))
print("Training Accuracy : %0.1f%% %0.3fs" % (training_acc*100,t2-t1))
print("Testing Accuracy : %0.1f%% %0.3fs" % (testing_acc*100,t3-t2))
```

```
Model fit time: 1024.112s;
Training Accuracy : 99.0% 0.174s
Testing Accuracy : 44.2% 0.034s
```

We then proceed with other advanced models, including XGBoost, RandomForestClassifier, LogisticRegression, SVM, MLPclassifier. We observe improvements in all accuracies and fitting times for all models.

3. XGboost (Claimed accuracy: 47.12%)

In [21]:

```
xgboost_model_final =xgboost.XGBClassifier(max_depth=4,n_estimators=50,learning_rate=0.1,
                        min_child_weight=1,gamma=0,subsample=0.8,colsample_bytree=0.8,reg_alpha=0.00
```

In [22]:

```
model=xgboost_model_final
t0 = time.time()
model.fit(train_x_dis,train_y_dis)
t1 = time.time()
training_acc=model.score(train_x_dis,train_y_dis)
t2 = time.time()
pred=model.predict(test_x_dis)
t3 = time.time()
testing_acc=model.score(test_x_dis,test_y_dis)

print("Model fit time: %0.3fs; " % (t1-t0))
print("Training Accuracy : %0.1f%% %0.3fs" % (training_acc*100,t2-t1))
print("Testing Accuracy : %0.1f%% %0.3fs" % (testing_acc*100,t3-t2))
```

```
Model fit time: 129.966s;
Training Accuracy : 99.8% 0.738s
Testing Accuracy : 46.8% 0.213s
```

4. RandomForestClassifier (Claimed accuracy: 45.48%)

Top

In [23]:

```
randomforest_model_final=RandomForestClassifier(n_estimators = 100, criterion = 'gini',
                                              random_state = 42, min_samples_leaf=1, max_features='sqrt')
```

In [24]:

```
model=randomforest_model_final
t0 = time.time()
model.fit(train_x_dis,train_y_dis)
t1 = time.time()
training_acc=model.score(train_x_dis,train_y_dis)
t2 = time.time()
pred=model.predict(test_x_dis)
t3 = time.time()
testing_acc=model.score(test_x_dis,test_y_dis)

print("Model fit time: %0.3fs; " % (t1-t0))
print("Training Accuracy : %0.1f%% %0.3fs" % (training_acc*100,t2-t1))
print("Testing Accuracy : %0.1f%% %0.3fs" % (testing_acc*100,t3-t2))
```

```
Model fit time: 7.565s;
Training Accuracy : 100.0% 0.126s
Testing Accuracy : 42.0% 0.035s
```

5. LogisticRegression (Claimed accuracy: 54.00%)

In [25]:

```
logistic_model_final =LogisticRegression(C=0.01,
                                          dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1,
                                          max_iter=300,
                                          multi_class='multinomial',
                                          penalty='l2',
                                          solver='newton-cg',
                                          tol=0.0001)
```

Top

In [26]:

```
model=logistic_model_final
t0 = time.time()
model.fit(train_x_dis,train_y_dis)
t1 = time.time()
training_acc=model.score(train_x_dis,train_y_dis)
t2 = time.time()
pred=model.predict(test_x_dis)
t3 = time.time()
testing_acc=model.score(test_x_dis,test_y_dis)

print("Model fit time: %0.3fs; " % (t1-t0))
print("Training Accuracy : %0.1f%% %0.3fs" % (training_acc*100,t2-t1))
print("Testing Accuracy : %0.1f%% %0.3fs" % (testing_acc*100,t3-t2))
```

```
Model fit time: 37.204s;
Training Accuracy : 82.7% 0.030s
Testing Accuracy : 55.4% 0.009s
```

6. LogisticRegression with PCA (Claimed accuracy: 55.2%)

In [58]:

```
logistic_model_final_2 =LogisticRegression(C=0.01,
                                           dual=False,
                                           fit_intercept=True,
                                           intercept_scaling=1,
                                           max_iter=300,
                                           multi_class='multinomial',
                                           penalty='l2',
                                           solver='newton-cg',
                                           tol=0.0001)
```

In [56]:

```
# Determine the number of components such that 99.9% variance is retained
pca = PCA(.999)
pca.fit(train_x_dis)
train_x_dis_pca = pca.transform(train_x_dis)
test_x_dis_pca = pca.transform(test_x_dis)
pca.n_components_ # See the number of components
```

Out[56]:

178

Top

In [59]:

```
model = logistic_model_final_2
t0 = time.time()
model.fit(train_x_dis_pca,train_y_dis)
t1 = time.time()
training_acc= model.score(train_x_dis_pca,train_y_dis)
t2 = time.time()
pred=model.predict(test_x_dis_pca)
t3 = time.time()
testing_acc=model.score(test_x_dis_pca,test_y_dis)

print("Model fit time: %0.3fs; " % (t1-t0))
print("Training Accuracy : %0.1f%% %0.3fs" % (training_acc*100,t2-t1))
print("Testing Accuracy : %0.1f%% %0.3fs" % (testing_acc*100,t3-t2))
```

```
Model fit time: 12.074s;
Training Accuracy : 81.8% 0.005s
Testing Accuracy : 55.2% 0.001s
```

7. SVM (Claimed accuracy: 50.04%)

In [27]:

```
svm_model_final =SVC(C=0.1,decision_function_shape='ovr',degree=2,gamma=0.1,kernel='linear')
```

In [28]:

```
model=svm_model_final
t0 = time.time()
model.fit(train_x_dis,train_y_dis)
t1 = time.time()
training_acc=model.score(train_x_dis,train_y_dis)
t2 = time.time()
pred=model.predict(test_x_dis)
t3 = time.time()
testing_acc=model.score(test_x_dis,test_y_dis)

print("Model fit time: %0.3fs; " % (t1-t0))
print("Training Accuracy : %0.1f%% %0.3fs" % (training_acc*100,t2-t1))
print("Testing Accuracy : %0.1f%% %0.3fs" % (testing_acc*100,t3-t2))
```

```
Model fit time: 20.159s;
Training Accuracy : 100.0% 22.935s
Testing Accuracy : 48.6% 5.694s
```

8. Neural Network-MLPClassifier (Claimed accuracy: 49.44%)

In [29]:

```
MLP_model_final =MLPClassifier(early_stopping=True,
                               hidden_layer_sizes=(3000,),
                               learning_rate='adaptive',
                               solver='lbfgs',
                               validation_fraction=0.2)
```

Top

In [30]:

```
model=MLP_model_final
t0 = time.time()
model.fit(train_x_dis,train_y_dis)
t1 = time.time()
training_acc=model.score(train_x_dis,train_y_dis)
t2 = time.time()
pred=model.predict(test_x_dis)
t3 = time.time()
testing_acc=model.score(test_x_dis,test_y_dis)

print("Model fit time: %0.3fs; " % (t1-t0))
print("Training Accuracy : %0.1f%% %0.3fs" % (training_acc*100,t2-t1))
print("Testing Accuracy : %0.1f%% %0.3fs" % (testing_acc*100,t3-t2))
```

```
Model fit time: 328.237s;
Training Accuracy : 100.0% 0.573s
Testing Accuracy : 48.8% 0.202s
```

Final model: VotingClassifier (Claimed accuracy: 54.32%)

Finally, we use VotingClassifier to combine the top models together as the final model. We can see in the following output that the testing accuracy is 53%.

In [31]:

```
voting_clf = VotingClassifier(
    estimators=[('rf_clf',randomforest_model_final),
                ('log_clf', logistic_model_final),
                ('svm_clf', svm_model_final),
               ('xgb_clf', xgboost_model_final),
                ('MLP_clf',MLP_model_final)],
    voting='hard',
    weights=[1,1.5,1.25,1,1.25],

)
```

Top

In [32]:

```
model=voting_clf
t0 = time.time()
model.fit(train_x_dis,train_y_dis)
t1 = time.time()
training_acc=model.score(train_x_dis,train_y_dis)
t2 = time.time()
pred=model.predict(test_x_dis)
t3 = time.time()
testing_acc=model.score(test_x_dis,test_y_dis)

print("Model fit time: %0.3fs; " % (t1-t0))
print("Training Accuracy : %0.1f%% %0.3fs" % (training_acc*100,t2-t1))
print("Testing Accuracy : %0.1f%% %0.3fs" % (testing_acc*100,t3-t2))
```

```
Model fit time: 492.865s;
Training Accuracy : 100.0% 24.127s
Testing Accuracy : 53.6% 6.098s
```

In [33]:

```
print(classification_report(test_y_dis,model.predict(test_x_dis)))
```

```
              precision    recall  f1-score   support

           1       0.61      0.83      0.70        24
           2       0.75      0.75      0.75        24
           3       0.48      0.46      0.47        28
           4       0.34      0.59      0.43        17
           5       0.80      0.80      0.80        25
           6       0.62      0.50      0.56        20
           7       0.58      0.58      0.58        26
           8       0.70      0.95      0.81        22
           9       0.75      0.63      0.69        19
          10       0.60      0.39      0.47        31
          11       0.45      0.45      0.45        22
          12       0.32      0.33      0.33        21
          13       0.32      0.23      0.27        26
          14       0.60      0.79      0.68        19
          15       0.39      0.64      0.48        11
          16       0.68      0.79      0.73        24
          17       0.61      0.47      0.53        30
          18       0.46      0.48      0.47        23
          19       0.15      0.12      0.13        17
          20       0.38      0.36      0.37        22
          21       0.45      0.43      0.44        23
          22       0.47      0.31      0.37        26

    accuracy                           0.54       500
   macro avg       0.52      0.54      0.52       500
weighted avg       0.53      0.54      0.53       500
```

Top