Facial Expression Identification- Final Python project Code

October 20, 2017

In [ ]:
```
#Implementing SVM on FER2013 dataset
```

In [ ]:
```
import cv2
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
from skimage.feature import hog
from sklearn import datasets, svm, metrics
from time import time
import logging
import matplotlib.pyplot as plt
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import fetch_lfw_people
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.svm import SVC
```

In [ ]:
```
seed=1712
np.random.seed(seed)
```

In [ ]:
```
data = pd.read_csv('fer2013.csv')
train_ind = np.array(np.where(data['Usage'] == 'Training'))
train_ind = train_ind.flatten()
test_ind = np.arange(train_ind.shape[0], data.shape[0])
```

In [ ]:
```
h = w = 48
# Training
X_train = np.zeros([train_ind.shape[0], 1, h, w])
y_train = np.zeros(train_ind.shape)
```

In [ ]:
```
count=0
for i in train_ind:
    X_train[count] = np.reshape(np.array([int(s)
```

```
                                for s in data['pixels'][i].split(' ')]), (48, 48))
            y_train[count] = np.float(data['emotion'][i])
            count+=1
```

In [ ]:
```
# Testing
X_test = np.zeros([test_ind.shape[0], 1, h, w])
y_test = np.zeros([test_ind.shape[0], 1])
```

In [ ]:
```
count=0
for i in test_ind:
    X_test[count] = np.reshape(np.array([int(s)
                                for s in data['pixels'][i].split(' ')]), (48, 48))
    y_test[count] = np.float(data['emotion'][i])
    count+=1
```

In [ ]:
```
nb_classes = len(np.unique(y_train))

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

In [ ]:
```
#============================================================================
# # Create a classifier: a Support Vector Classifier
#============================================================================


# ###########################################################################
# Compute a PCA (eigenfaces) on the face dataset (treated as unlabeled
# dataset): unsupervised feature extraction / dimensionality reduction
```

In [ ]:
```
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))
```

In [ ]:
```
n_components = 150
print("Extracting the top %d eigenfaces from %d faces"
      % (n_components, X_train.shape[0]))
t0 = time()
pca = PCA(n_components=n_components, svd_solver='randomized',
          whiten=True).fit(X_train)
print("done in %0.3fs" % (time() - t0))
```

In [ ]:
```
eigenfaces = pca.components_.reshape((n_components, h, w))

print("Projecting the input data on the eigenfaces orthonormal basis")
t0 = time()
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("done in %0.3fs" % (time() - t0))
```

In [ ]:
```
# ###########################################################################
# Grid search for best parameters of the SVM classification model
```

```
In [ ]: print("Fitting the classifier to the training set")
        t0 = time()
        param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
                      'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
        clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'), param_grid)
        clf = clf.fit(X_train_pca, y_train)
        print("done in %0.3fs" % (time() - t0))
        print("Best estimator found by grid search:")
        print(clf.best_estimator_)

In [ ]: #Best estimator found by grid search:
        #SVC(C=1000.0, cache_size=200, class_weight='balanced', coef0=0.0,
        #  decision_function_shape=None, degree=3, gamma=0.01, kernel='rbf',
        #  max_iter=-1, probability=False, random_state=None, shrinking=True,
        #  tol=0.001, verbose=False)

In [ ]: # SVM model using bes parameters
        classifier = svm.SVC(C=1000.0, cache_size=200, class_weight='balanced',
                             coef0=0.0,
          decision_function_shape=None, degree=3, gamma=0.01, kernel='rbf',
          max_iter=-1, probability=True, random_state=None, shrinking=True,
          tol=0.001, verbose=True)

In [ ]: # Train SVM using bes parameters
        classifier.fit(X_train_pca, y_train)

        # Now predict the value of the digit on the second half:
        expected = y_test
        predicted = classifier.predict(X_test_pca)

        print("Classification report for classifier %s:\n%s\n"
              % (classifier, metrics.classification_report(expected, predicted)))
        print("Confusion matrix:\n%s" % metrics.confusion_matrix(expected, predicted))

In [ ]: #Implementing CNN on FER2013 dataset

In [ ]: import os
        #import cv2
        import keras
        import numpy as np
        import pandas as pd
        import warnings
        warnings.filterwarnings("ignore")
        from keras.regularizers import l2
        from keras.models import Sequential
        from keras.utils import np_utils
        from keras.preprocessing.image import ImageDataGenerator
        from keras.layers.convolutional import Convolution2D as Conv2D,
        ZeroPadding2D as Zero2D
```

```
from keras.layers.core import Flatten, Activation, Dense, Dropout
from keras.layers.normalization import BatchNormalization
from keras.layers.pooling import MaxPooling2D
from keras import backend as K
K.set_image_dim_ordering('th')
```

In [ ]:
```
seed=1712
np.random.seed(seed)
weight_decay=1e-4
```

In [ ]:
```
def net():

    # 2Conv-1FC
        model = Sequential()
        model.add(keras.layers.convolutional.Cropping2D(cropping=((2, 2), (2, 2)),
                                                        input_shape=(1, 48, 48)))
        model.add(Conv2D(32, 5, 5, bias=0.1, W_regularizer=l2(weight_decay),
                         input_shape=(1, 48, 48)))
        model.add(BatchNormalization(mode=0, axis=1,
                                     gamma_regularizer=l2(weight_decay),
                                     beta_regularizer=l2(weight_decay)))
        model.add(Activation('relu'))
        #print (model.output_shape)
        model.add(MaxPooling2D(pool_size = (3, 3), strides=(2, 2)))
        model.add(Dropout(0.2))
        #print (model.output_shape)

        model.add(Conv2D(32, 4, 4, bias=0.1, W_regularizer=l2(weight_decay)))
        model.add(BatchNormalization(mode=0, axis=1,
                                     gamma_regularizer=l2(weight_decay),
                                     beta_regularizer=l2(weight_decay)))
        model.add(Activation('relu'))
        #print (model.output_shape)
        model.add(MaxPooling2D(pool_size = (3, 3), strides=(2, 2)))

        model.add(Flatten())
        model.add(Dense(512, bias=0.1, W_regularizer=l2(weight_decay),
                        b_regularizer=l2(weight_decay)))
        model.add(Activation('relu'))
        model.add(Dropout(0.2))
        # model.add(Dense(512, bias=0.1, W_regularizer=l2(weight_decay),
        b_regularizer=l2(weight_decay)))
        # model.add(Activation('relu'))
        # #print (model.output_shape)
        model.add(Dense(7, activation='softmax', W_regularizer=l2(weight_decay),
                        b_regularizer=l2(weight_decay)))
        # print (model.output_shape)
```

```
# 3Conv-1FC
    model = Sequential()
    model.add(keras.layers.convolutional.Cropping2D(cropping=((2, 2), (2, 2)),
                                                    input_shape=(1, 48, 48)))
    model.add(Conv2D(32, 5, 5, bias=0.1, W_regularizer=l2(weight_decay),
                     input_shape=(1, 48, 48)))
    model.add(BatchNormalization(mode=0, axis=1,
                                 gamma_regularizer=l2(weight_decay),
                                 beta_regularizer=l2(weight_decay)))
    model.add(Activation('relu'))
    #print (model.output_shape)
    model.add(MaxPooling2D(pool_size = (3, 3), strides=(2, 2)))
    model.add(Dropout(0.2))
    #print (model.output_shape)

    model.add(Conv2D(32, 4, 4, bias=0.1, W_regularizer=l2(weight_decay)))
    model.add(BatchNormalization(mode=0, axis=1,
                                 gamma_regularizer=l2(weight_decay),
                                 beta_regularizer=l2(weight_decay)))
    model.add(Activation('relu'))
    #print (model.output_shape)
    model.add(MaxPooling2D(pool_size = (3, 3), strides=(2, 2)))

    model.add(Conv2D(64, 3, 3, bias=0.1, W_regularizer=l2(weight_decay)))
    model.add(BatchNormalization(mode=0, axis=1,
                                 gamma_regularizer=l2(weight_decay),
                                 beta_regularizer=l2(weight_decay)))
    model.add(Activation('relu'))
    #print (model.output_shape)
    model.add(MaxPooling2D(pool_size = (3, 3), strides=(2, 2)))

    model.add(Flatten())
    model.add(Dense(512, bias=0.1, W_regularizer=l2(weight_decay),
                    b_regularizer=l2(weight_decay)))
    model.add(Activation('relu'))
    model.add(Dropout(0.2))
    # model.add(Dense(512, bias=0.1, W_regularizer=l2(weight_decay),
    b_regularizer=l2(weight_decay)))
    # model.add(Activation('relu'))
    # #print (model.output_shape)
    model.add(Dense(7, activation='softmax', W_regularizer=l2(weight_decay),
                    b_regularizer=l2(weight_decay)))
    # print (model.output_shape)



# 3Conv-2FC
```

```
model = Sequential()
model.add(keras.layers.convolutional.Cropping2D(cropping=((2, 2), (2, 2)),
                                                input_shape=(1, 48, 48)))
model.add(Conv2D(32, 5, 5, bias=0.1, W_regularizer=l2(weight_decay),
                 input_shape=(1, 48, 48)))
model.add(BatchNormalization(mode=0, axis=1,
                             gamma_regularizer=l2(weight_decay),
                             beta_regularizer=l2(weight_decay)))
model.add(Activation('relu'))
#print (model.output_shape)
model.add(MaxPooling2D(pool_size = (3, 3), strides=(2, 2)))
model.add(Dropout(0.2))
#print (model.output_shape)

model.add(Conv2D(32, 4, 4, bias=0.1, W_regularizer=l2(weight_decay)))
model.add(BatchNormalization(mode=0, axis=1,
                             gamma_regularizer=l2(weight_decay),
                             beta_regularizer=l2(weight_decay)))
model.add(Activation('relu'))
#print (model.output_shape)
model.add(MaxPooling2D(pool_size = (3, 3), strides=(2, 2)))

model.add(Conv2D(64, 3, 3, bias=0.1, W_regularizer=l2(weight_decay)))
model.add(BatchNormalization(mode=0, axis=1,
                             gamma_regularizer=l2(weight_decay),
                             beta_regularizer=l2(weight_decay)))
model.add(Activation('relu'))
#print (model.output_shape)
model.add(MaxPooling2D(pool_size = (3, 3), strides=(2, 2)))

model.add(Flatten())
model.add(Dense(512, bias=0.1, W_regularizer=l2(weight_decay),
                b_regularizer=l2(weight_decay)))
model.add(Activation('relu'))
model.add(Dropout(0.2))
# model.add(Dense(512, bias=0.1, W_regularizer=l2(weight_decay),
b_regularizer=l2(weight_decay)))
# model.add(Activation('relu'))
# #print (model.output_shape)
model.add(Dense(7, activation='softmax', W_regularizer=l2(weight_decay),
                b_regularizer=l2(weight_decay)))
# print (model.output_shape)

 model.add(Flatten())
model.add(Dense(512, bias=0.1, W_regularizer=l2(weight_decay),
                b_regularizer=l2(weight_decay)))
model.add(Activation('relu'))
model.add(Dropout(0.2))
```

```
    # model.add(Dense(512, bias=0.1, W_regularizer=l2(weight_decay),
    b_regularizer=l2(weight_decay)))
    # model.add(Activation('relu'))
    # #print (model.output_shape)
    model.add(Dense(7, activation='softmax', W_regularizer=l2(weight_decay),
                    b_regularizer=l2(weight_decay)))
    # print (model.output_shape)


# 4Conv-1FC
    model = Sequential()
    model.add(Conv2D(64, 5, 5, border_mode='same', input_shape=(1, 48, 48),
                     bias=False, W_regularizer=l2(weight_decay)))
    model.add(BatchNormalization(mode=0, axis=1, gamma_regularizer=l2(weight_decay),
                                 beta_regularizer=l2(weight_decay)))
    model.add(Activation('relu'))
    #print (model.output_shape)
    model.add(MaxPooling2D((3,3), strides=(2,2)))
    #print (model.output_shape)

    model.add(Conv2D(64, 5, 5, border_mode='same', bias=False,
                     W_regularizer=l2(weight_decay)))
    model.add(BatchNormalization(mode=0, axis=1,
                                 gamma_regularizer=l2(weight_decay),
                                 beta_regularizer=l2(weight_decay)))
    model.add(Activation('relu'))
    #print (model.output_shape)
    model.add(MaxPooling2D((3,3), strides=(2,2)))
    #print (model.output_shape)

    model.add(Conv2D(32, 3, 3, border_mode='same', bias=False,
                     W_regularizer=l2(weight_decay)))
    model.add(BatchNormalization(mode=0, axis=1,
                                 gamma_regularizer=l2(weight_decay),
                                 beta_regularizer=l2(weight_decay)))
    model.add(Activation('relu'))
    #print (model.output_shape)
    model.add(MaxPooling2D((3,3), strides=(2,2)))
    model.add(Dropout(0.25))
    #print (model.output_shape)

    model.add(Conv2D(32, 3, 3, border_mode='same', bias=False,
                     W_regularizer=l2(weight_decay)))
    model.add(BatchNormalization(mode=0, axis=1,
                                 gamma_regularizer=l2(weight_decay),
                                 beta_regularizer=l2(weight_decay)))
    model.add(Activation('relu'))
    #print (model.output_shape)
```

```
        model.add(MaxPooling2D((3,3), strides=(2,2)))
        model.add(Dropout(0.25))

        model.add(Flatten())
        model.add(Dense(512, bias=0.1, W_regularizer=l2(weight_decay),
                        b_regularizer=l2(weight_decay)))
        model.add(Activation('relu'))
        model.add(Dropout(0.2))
        #print (model.output_shape)
        model.add(Dense(7, activation='softmax', W_regularizer=l2(weight_decay),
                        b_regularizer=l2(weight_decay)))
        #print (model.output_shape)



        return model
```

In [ ]:
```
np.random.seed(seed)
data = pd.read_csv('fer2013.csv')
train_ind = np.array(np.where(data['Usage'] == 'Training'))
train_ind = train_ind.flatten()
test_ind = np.arange(train_ind.shape[0], data.shape[0])

# Training
X_train = np.zeros([train_ind.shape[0], 1, 48, 48])
y_train = np.zeros([train_ind.shape[0], 1])

count=0
for i in train_ind:
    X_train[count] = np.reshape(np.array([int(s)
                        for s in data['pixels'][i].split(' ')]), (48, 48))
    y_train[count] = np.float(data['emotion'][i])
    count+=1
```

In [ ]:
```
# Testing
X_test = np.zeros([test_ind.shape[0], 1, 48, 48])
y_test = np.zeros([test_ind.shape[0], 1])

count=0
for i in test_ind:
    X_test[count] = np.reshape(np.array([int(s)
                        for s in data['pixels'][i].split(' ')]), (48, 48))
    y_test[count] = np.float(data['emotion'][i])
    count+=1

nb_classes = len(np.unique(y_train))
```

In [ ]:
```
# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, nb_classes)
```

```
        Y_test = np_utils.to_categorical(y_test, nb_classes)

        X_train = X_train.astype('float32')
        X_test = X_test.astype('float32')
```

In [ ]:
```
if K.image_dim_ordering() == "th":
    for i in range(1):
        mean = np.mean(X_train[:, i, :, :])
        std = np.std(X_train[:, i, :, :])
        X_train[:, i, :, :] = (X_train[:, i, :, :] - mean) / std
        X_test[:, i, :, :] = (X_test[:, i, :, :] - mean) / std

elif K.image_dim_ordering() == "tf":
    for i in range(1):
        mean = np.mean(X_train[:, i, :, :])
        std = np.std(X_train[:, i, :, :])
        X_train[:, i, :, :] = (X_train[:, i, :, :] - mean) / std
        X_test[:, i, :, :] = (X_test[:, i, :, :] - mean) / std
```

In [ ]:
```
np.random.seed(seed)
model = net()

# sgd = keras.optimizers.SGD(lr=0.01, momentum=0.95, decay=0.002, nesterov=False)
# model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

def scheduler(epoch):
    if epoch == int(0.5 * epochs):
        K.set_value(model.optimizer.lr, np.float32(lrate / 10.))
    if epoch == int(0.75 * epochs):
        K.set_value(model.optimizer.lr, np.float32(lrate / 100.))
    # K.set_value(model.optimizer.lr, np.float32(K.get_value(sgd.lr) - diff))
    #K.set_value(model.optimizer.momentum,
    #K.get_value(model.optimizer.momentum) + diff)
    return float(K.get_value(sgd.lr))
```

In [ ]:
```
epochs=200
lrate = 0.1
diff = 0.49/epochs
batch_size = 128
change_lr = keras.callbacks.LearningRateScheduler(scheduler)
reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor='val_acc', factor=0.75,
                                              patience=10, min_lr=0.0001)
sgd = keras.optimizers.SGD(lr=lrate, momentum=0.9, decay=0.0002,
                           nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd,
              metrics=['accuracy'])

# np.random.seed(seed)
```

```
# hist = model.fit(X_train, Y_train,  validation_data=(X_test[:3589],
#Y_test[:3589]), batch_size=batch_size, nb_epoch=epochs, verbose=1,
#callbacks=[change_lr])

datagen = ImageDataGenerator(zoom_range=0.2, horizontal_flip=True)
# randomly flip images

# Compute quantities required for feature-wise normalization
# (std, mean, and principal components if ZCA whitening is applied).
datagen.fit(X_train)
#np.random.seed(seed)
# Fit the model on the batches generated by datagen.flow().
np.random.seed(seed)
hist = model.fit_generator(datagen.flow(X_train, Y_train, batch_size=batch_size),
                           steps_per_epoch=X_train.shape[0] // batch_size,
                           epochs=epochs,
                           validation_data=(X_test[:3589], Y_test[:3589]),
                           verbose=1, callbacks=[reduce_lr])

score = model.evaluate(X_test[3589:], Y_test[3589:], verbose=1)
print('Test accuracy:', score[1])

model_yaml = model.to_yaml()
with open("face_emot.yaml", "w") as yaml_file:
   yaml_file.write(model_yaml)
# serialize weights to HDF5
model.save_weights("face_emot.h5")
print("Saved model to disk")
np.save("face_emot_loss.npy", hist.history['loss'])
np.save("face_emot_acc.npy", hist.history['acc'])
np.save("face_emot_val_loss.npy", hist.history['val_loss'])
np.save("face_emot_val_acc.npy", hist.history['val_acc'])
```