Image dataset

December 21

2023

The FER 2013 dataset is a collection of 35,887 grayscale images of human faces with different expressions1. The images are labeled with one of the seven emotions: Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral2. The dataset was created by scraping Google Images for each emotion and its synonyms3. It is widely used as a benchmark for facial expression recognition models4.

For FER 2013 dataset

info :

Number of classes : 5 (Angry , Disgust , Fear , Happy , Sad)

Size of image : 48 *48

Num of samples : 28709 samples

Num of train samples : 16458 samples

Num of test samples : 4115 samples

Read Dataset:

import nessisary liberary

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score , ConfusionMatrixDisplay
from skimage import color, feature, exposure
from skimage.feature import hog
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss
from sklearn.metrics import roc_curve, roc_auc_score, auc
from sklearn.preprocessing import label_binarize
```

load data

```
In [2]: # load the data
data_train = pd.read_csv('D:/Machine and ai/FER with Kmean/challenges-in-representation-learning-facial-expression-recognition-ch
data_train.head()
```

- ✓ We read the dataset file from the path and display the first five row.

Out[2]:

	emotion	pixels
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...
1	0	151 150 147 155 148 133 111 140 170 174 182 15...
2	2	231 212 156 164 174 138 161 173 182 200 106 38...
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...
4	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...

Preprocessing :

- ✓ The current dataset has two column (emotion , pixels), the pixels column has datatype of string value separated with space which refer to the dimension of the image, so we need to convert it to list of array that make preprocessing possible.

preprocessing

```
In [3]: # make the datset to an array of images of pixels
image_array =[]
for i, row in enumerate(data_train.index):
        image = np.fromstring(data_train.loc[row, 'pixels'], dtype=int, sep=' ')
        image_array.append(image.flatten())
```

```
In [4]: image_array
```

```
Out[4]: [array([ 70,  80,  82, ..., 106, 109,  82]),
 array([151, 150, 147, ..., 193, 183, 184]),
 array([231, 212, 156, ...,  88, 110, 152]),
 array([ 24,  32,  36, ..., 142, 143, 142]),
 array([ 4,  0,  0, ..., 30, 29, 30]),
 array([55, 55, 55, ..., 34, 30, 57]),
 array([ 20,  17,  19, ...,  99, 107, 118]),
 array([ 77,  78,  79, ..., 125,  67,  68]),
 array([85, 84, 90, ..., 58, 73, 84]),
 array([255, 254, 255, ..., 254, 255, 255]),
 array([ 30,  24,  21, ..., 172, 173, 173]),
 array([39, 75, 78, ..., 84, 83, 87]),
 array([219, 213, 206, ...,   0,   0,   0]),
 array([148, 144, 130, ..., 112, 111, 111]),
 array([ 4,  2, 13, ...,  3,  7, 12]),
 array([107, 107, 109, ...,  83,  84, 106]),
 array([14, 14, 18, ...,  9, 10, 10]),
 array([255, 255, 255, ...,  79,  79,  83]),
 array([134, 124, 167, ...,  34,  28, 139]),
```

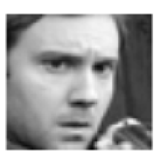
Normalization :

```
In [7]: flat_images = np.array(image_array)
        target = np.array(lables)
        # normalization
        flat_images = flat_images / 255
```

- ✓ We make normalization to scale the pixels value to be from 0 to 1, which can improve the performance of image processing.

show some images

```
In [8]: for i in range(5):
            fig, ax1 = plt.subplots(1,figsize=(2,1), sharex=True, sharey=True)
            ax1.axis('off')
            ax1.imshow(image_array[i].reshape(48,48), cmap=plt.cm.gray)
            plt.show()
```



- ✓ We convert the array from 1d to 2d.

convert it into a dataframe

```
In [9]: df = pd.DataFrame(flat_images)
        df['target']=target
```

```
In [10]: df
```

Out[10]:

	0	1	2	3	4	5	6	7	8	9	...	2295	2296	2297	2298	2299
0	0.274510	0.313725	0.321569	0.282353	0.227451	0.227451	0.235294	0.247059	0.211765	0.227451	...	0.713725	0.717647	0.533333	0.415686	0.454902
1	0.592157	0.588235	0.576471	0.607843	0.580392	0.521569	0.435294	0.549020	0.666667	0.682353	...	0.423529	0.372549	0.423529	0.400000	0.262745
2	0.905882	0.831373	0.611765	0.643137	0.682353	0.541176	0.631373	0.678431	0.713725	0.784314	...	0.541176	0.596078	0.478431	0.447059	0.396078
3	0.094118	0.125490	0.141176	0.117647	0.125490	0.090196	0.074510	0.078431	0.117647	0.160784	...	0.494118	0.517647	0.517647	0.521569	0.533333
4	0.015686	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.133333	0.121569	0.121569	0.121569	0.105882
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28704	0.329412	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333	0.333333	0.337255	0.337255	...	0.133333	0.137255	0.141176	0.156863	0.105882
28705	0.447059	0.439216	0.443137	0.443137	0.435294	0.435294	0.439216	0.443137	0.450980	0.443137	...	0.368627	0.419608	0.470588	0.552941	0.564706
28706	0.290196	0.317647	0.341176	0.349020	0.372549	0.392157	0.384314	0.364706	0.411765	0.470588	...	0.839216	0.827451	0.819608	0.784314	0.764706
28707	0.870588	0.890196	0.796078	0.352941	0.337255	0.352941	0.329412	0.301961	0.368627	0.341176	...	0.545098	0.552941	0.568627	0.537255	0.545098
28708	0.764706	0.780392	0.803922	0.807843	0.803922	0.796078	0.807843	0.819608	0.815686	0.823529	...	0.364706	0.262745	0.133333	0.023529	0.027451

28709 rows × 2305 columns

- ✓ Dependent to our task we should have only five classes, so we have to drop two classes as our dataset has seven clases.

drop some rows to be only 5 classes

In [11]:
```
df= df[df['target'] != 5]
df= df[df['target'] != 6]
```

In [12]:
```
# names of classes
df['target'].unique()
```

Out[12]:
```
array([0, 2, 4, 3, 1])
```

In [13]:
```
count0= len(df[df['target'] == 0])
count1= len(df[df['target'] == 1])
count2= len(df[df['target'] == 2])
count3= len(df[df['target'] == 3])
count4= len(df[df['target'] == 4])
print('number of Angry images: ',count0)
print('number of Disgust images: ',count1)
print('number of Fear images: ',count2)
print('number of Happy images: ',count3)
print('number of Sad images: ',count4)
```

```
number of Angry images:  3995
number of Disgust images:  436
number of Fear images:  4097
number of Happy images:  7215
number of Sad images:  4830
```

We split the dataset to x and y, as x the features and y the label.

split the data to X and y ... features and targets

In [14]:
```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

Hog:

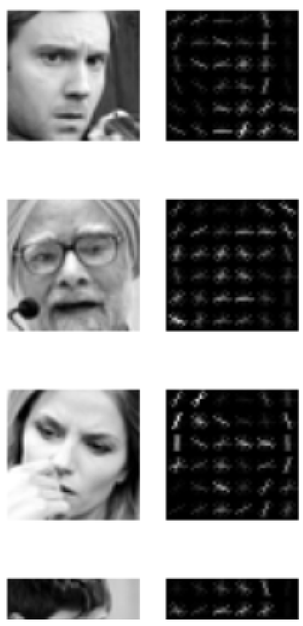
apply hog algorithm

In [15]:
```
def extract_hog_features(image):
    gray_image = image
    # Calculate HOG features
    hog_features, hog_image = feature.hog(gray_image, visualize=True)

    # Enhance the contrast of the HOG image for better visualization
    hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))
    return hog_features , hog_image_rescaled
```

In [16]:
```
hog_features_list = []
hog_images=[]
for index, row in X.iterrows():
    image_pixels = row.values.reshape(48, 48)
    hog_features ,hog_image = extract_hog_features(image_pixels)
    hog_features_list.append(hog_features)
    hog_images.append(hog_image)

hog_features_array = np.array(hog_features_list)
```

plot some images before and after hog

In [19]:
```
for i in range(5):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(2,1), sharex=True, sharey=True)
    ax1.axis('off')
    ax1.imshow(image_array[i].reshape(48,48), cmap=plt.cm.gray)
    ax2.axis('off')
    ax2.imshow(hog_images[i], cmap=plt.cm.gray)
    plt.show()
```



Logistic Regression :

We use another algo to train the model.

split the data to train and test

In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(hog_features_array, y, test_size=0.2, random_state=42)
```

apply the Logistic Regression model and fit it

In [21]:
```
model = LogisticRegression(max_iter=1000)
```

In [22]:
```
model.fit(X_train,y_train)
```

Out[22]:
```
LogisticRegression(max_iter=1000)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

predict the lables from the test and calculate the accurcy

In [23]:
```
y_predict = model.predict(X_test)
```

In [24]:
```
y_predict
```

Out[24]:
```
array([0, 3, 4, ..., 3, 4, 2])
```

The accuracy is 0.51

predict the lables from the test and calculate the accurcy

In [23]:
```
y_predict = model.predict(X_test)
```

In [24]:
```
y_predict
```

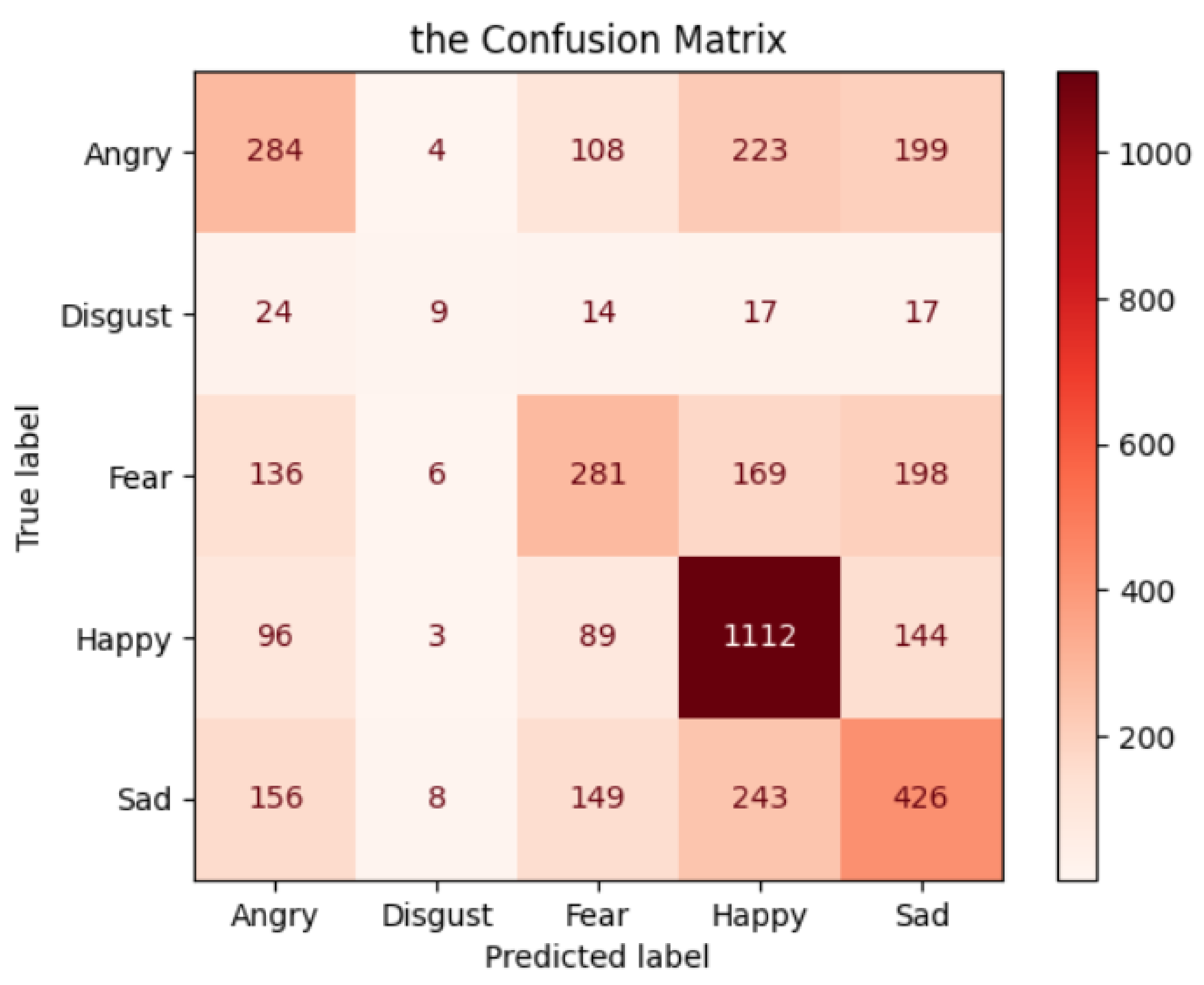
Out[24]:
```
array([0, 3, 4, ..., 3, 4, 2])
```

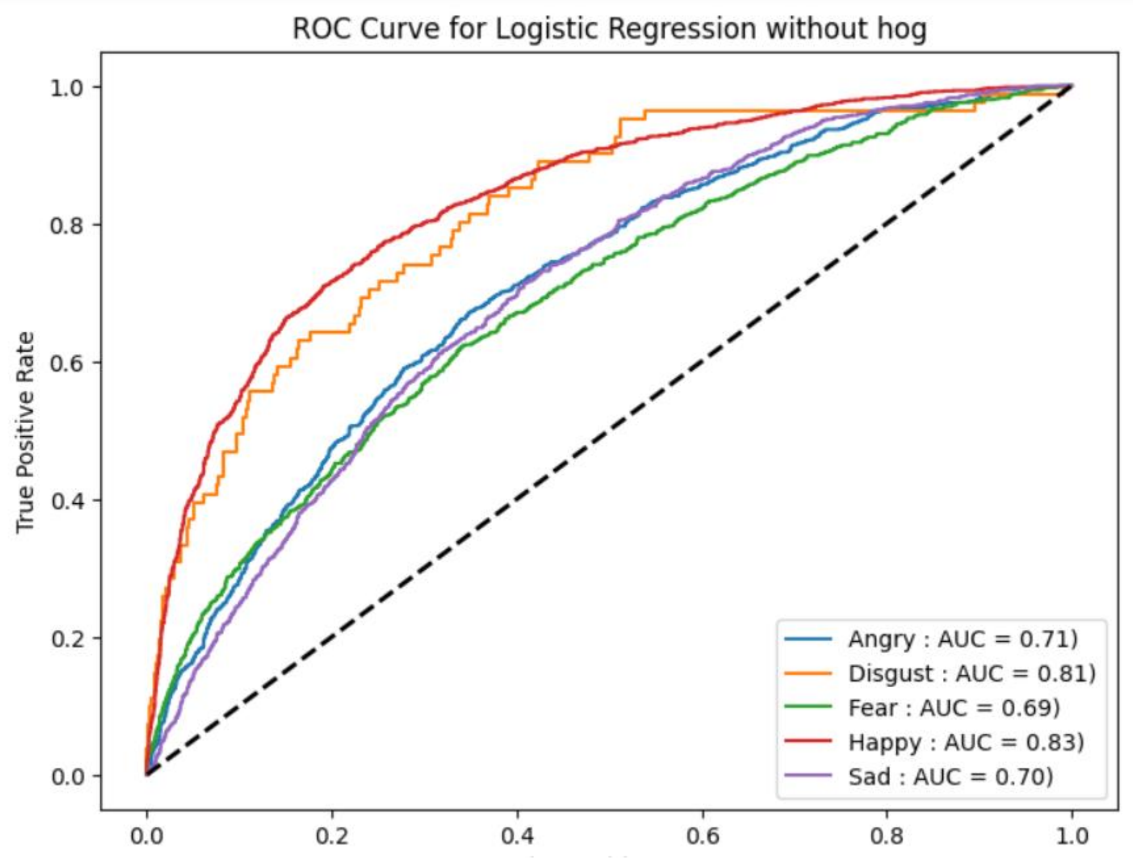
In [25]:
```
y_test
```

Out[25]:
```
12365    0
5640     4
20979    4
15528    4
18323    3
        ..
8482     0
19828    2
16040    3
17672    3
21181    0
Name: target, Length: 4115, dtype: int32
```

In [26]:
```
test_list = y_test.to_list()
```

In [27]:
```
accuracy = accuracy_score(test_list,y_predict)
print(accuracy)
```





K-mean :

apply the Kmean model and fit it

In [18]:
```
n_clusters = 5
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans.fit(hog_features_list)
```

```
c:\Users\ahmed\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

Out[18]: KMeans(n_clusters=5, random_state=42)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [19]:
```
labels = kmeans.labels_
```

Visualize clusters in a scatter plot

In [20]:
```
centers = kmeans.cluster_centers_

# Visualize clusters in a scatter plot
plt.scatter(hog_features_array[:, 0], hog_features_array[:, 1], c=labels, cmap='viridis')

# Plot cluster centers
plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='*', label='Centroids')
plt.title('the five clusters')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```