Pepito, Alyssa Mae and Soleño, Keziah Antonette August 1, 2022

CS171 | BM1

<u>Term Project</u>

Applying Data Science Life Cycle in Emotion Recognition through Facial Expression

I. Data Acquisition

The dataset used in this project was acquired from the following website: https://www.kaggle.com/datasets/jonathanoheix/face-expression-recognition-dataset

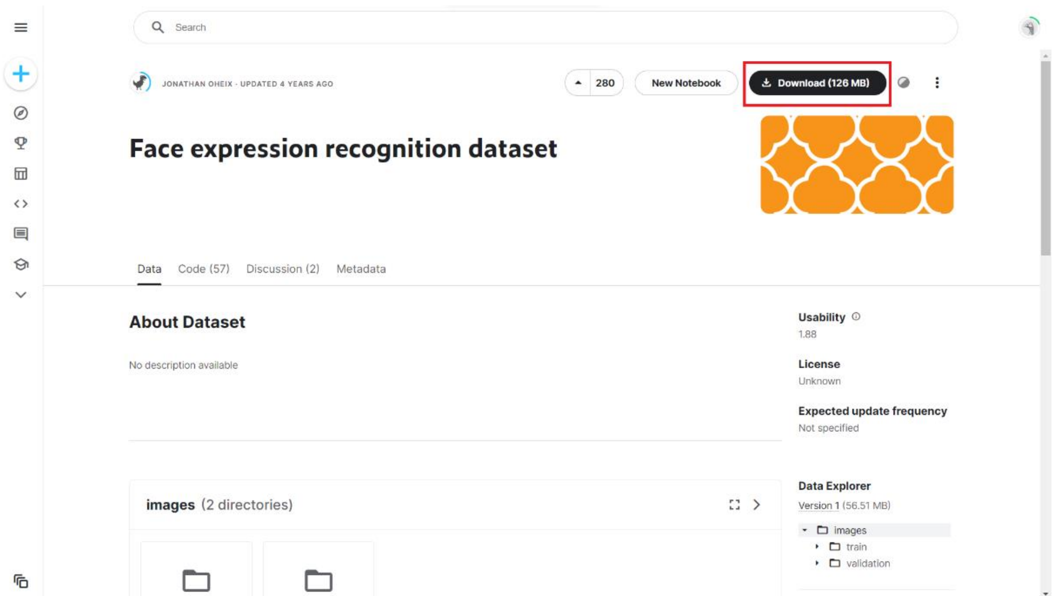

Figure 1 Face Expression Recognition Dataset from Kaggle

After downloading the dataset, its .zip file was extracted into the following folders under the file name "images."

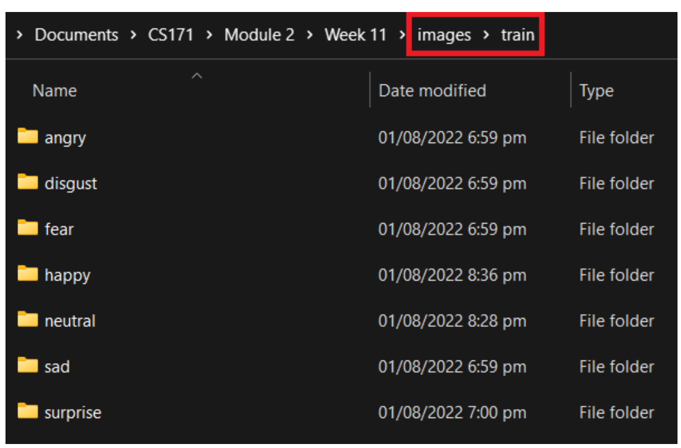

Figure 2.1 Snippet of the 'train' folder from the downloaded dataset

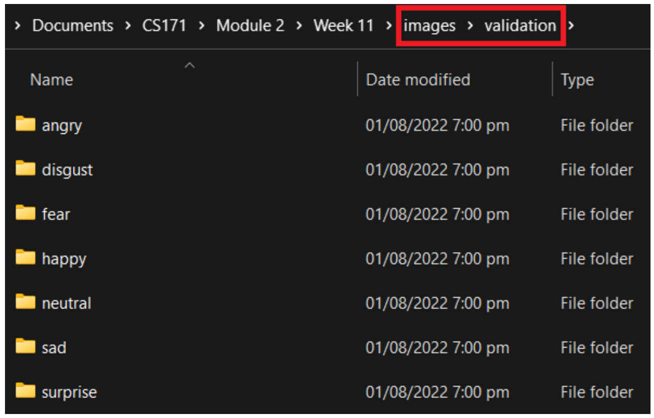

Figure 2.2 Snippet of the 'validation' folder from the downloaded dataset

Now that the dataset has been downloaded, twenty-five files per emotion were selected to be used as the data for facial feature extraction.

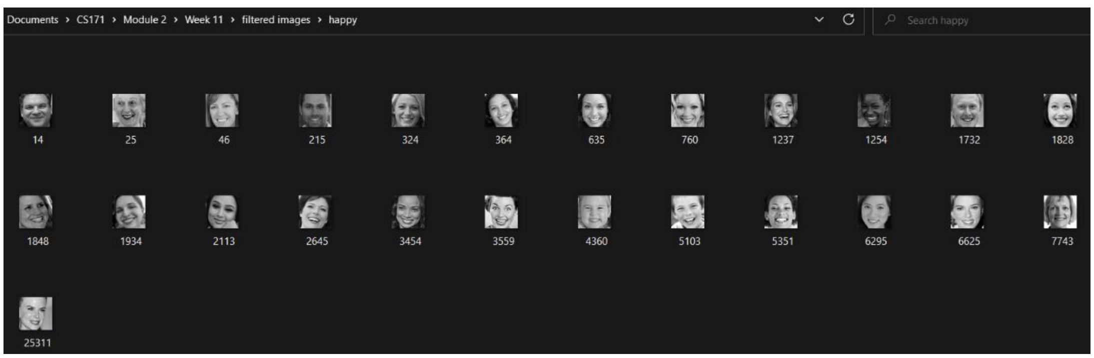

Figure 3 Twenty-five files from the 'happy' folder were selected

Note: These images were not selected at random. Images with evident and clear facial features were chosen as the data for Python to better identify the facial features that were to be extracted from each image. This process was performed for all emotions.

The next step after selecting useful data is extracting the facial features from these images.

II. Facial Feature Extraction

The main Python library used in this project was Detector, which can be accessed through the Python library called py-feat. Since we did not have the library installed yet, we performed the following code in command prompt to install the required library.

```
Command Prompt
Microsoft Windows [Version 10.0.19043.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS>pip install py-feat
Requirement already satisfied: py-feat in c:\users\asus\appdata\local\programs\python\python310\lib\site-packa
ges (0.4.0)
Requirement already satisfied: numpy>=1.9 in c:\users\asus\appdata\local\programs\python\python310\lib\site-pa
ckages (from py-feat) (1.22.4)
Requirement already satisfied: pandas>=0.20 in c:\users\asus\appdata\local\programs\python\python310\lib\site-
packages (from py-feat) (1.4.2)
Requirement already satisfied: torchvision in c:\users\asus\appdata\local\programs\python\python310\lib\site-p
ackages (from py-feat) (0.12.0+cu113)
Requirement already satisfied: nltools>=0.3.6 in c:\users\asus\appdata\local\programs\python\python310\lib\sit
e-packages (from py-feat) (0.4.5)
Requirement already satisfied: pywavelets>=0.3.0 in c:\users\asus\appdata\local\programs\python\python310\lib\
site-packages (from py-feat) (1.3.0)
Requirement already satisfied: tqdm in c:\users\asus\appdata\local\programs\python\python310\lib\site-packages
 (from py-feat) (4.64.0)
Requirement already satisfied: celluloid in c:\users\asus\appdata\local\programs\python\python310\lib\site-pac
kages (from py-feat) (0.2.0)
Requirement already satisfied: opencv-contrib-python>=4.4.0.46 in c:\users\asus\appdata\local\programs\python\
python310\lib\site-packages (from py-feat) (4.6.0.66)
Requirement already satisfied: seaborn>=0.7.0 in c:\users\asus\appdata\local\programs\python\python310\lib\sit
e-packages (from py-feat) (0.11.2)
Requirement already satisfied: torch in c:\users\asus\appdata\local\programs\python\python310\lib\site-package
s (from py-feat) (1.11.0+cu113)
Requirement already satisfied: joblib in c:\users\asus\appdata\local\programs\python\python310\lib\site-packag
es (from py-feat) (1.1.0)
```

Figure 4 Installing Py-Feat using the command prompt

We imported the py-feat library along with various libraries that would be used later for pre-processing, machine learning, model training, etc.

```python
from feat import Fex
from feat import Detector

import numpy as np
import glob
import pandas as pd
import os
import csv # Preprocessing
import matplotlib.pyplot as plt

# Libraries for model training
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
```

Figure 5 Imported Libraries

The Detector class is a great tool for performing feature extraction on images as it demonstrates how to detect faces, facial landmarks, action units, and emotions from images. (n.d.) When using the Detector class, specific or default models may be used and loaded. In this example, default models were loaded and defined as follows:

```python
detector = Detector(
    face_model="retinaface",
    landmark_model="mobilefacenet",
    au_model='svm',
    emotion_model="resmasknet",
    facepose_model="img2pose",
)

detector
```

✓ 10.3s Python

```
feat.detector.Detector(face_model=retinaface, landmark_model=mobilefacenet, au_model=svm,
emotion_model=resmasknet, facepose_model=img2pose)
```

Figure 6 Initializing the Py-Feat Detector

After initializing the facial expression detector, we proceeded with saving our images labels into a csv file. The code goes through each subfolder from the 'images' folder and classify each image based on the name of their subfolder. For example, the image from the 'fear' subfolder is labeled 'fear' as it resides within that subfolder.

```python
# Saving the image labels into a CSV file
header = 'label'
header = header.split()

file = open('emotionslabel.csv', 'w', newline='')
with file:
    writer = csv.writer(file)
    writer.writerow(header)
emotions = 'anger disgust fear happy neutral sad surprise'.split()

for m in emotions:
    for filename in os.listdir(f'images/{m}'):
        filename = f'images/{m}/{filename}'
        to_append = f'{m}'
        file = open('emotionslabel.csv', 'a', newline='')
        with file:
            writer = csv.writer(file)
            writer.writerow(to_append.split())
```

Figure 7 Saving the image labels into a CSV file

The Detector class is also flexible enough to process multiple image files simultaneously if it is passed a list of images. The group used the glob library to return a list of path names that match the string that indicates the path specification. The asterisk symbol will match any files and zero or more directories and subdirectories. The code snippet below shows how this was done.

```python
images = glob.glob('images/*/*.jpg', recursive=False)
```

Figure 8 Returning a list of images using the glob module

Then, predictions were made by using the loaded models from the initialized detector instance and then applying the detect_image() method to perform facebox, face landmark, facial action units, facepose, and emotion detection on the images.

Since multiple images of different emotions are being used, detect_image() must be passed as a list of images to process the images simultaneously. Batch_size indicates how many batches of images we want to run at one shot, so the group used 20.

(Note: This method always returns a Fex data instance.)

```python
mixed_prediction = detector.detect_image(images, batch_size=20)
mixed_prediction
```

Figure 9 Detect features of images in every class

As we have 175 images and many features for each image, it took more than 10 minutes to process the Fex file. Shown below are a few images from the 'anger' and 'surprise' classes.

	frame	FaceRectX	FaceRectY	FaceRectWidth	FaceRectHeight	FaceScore	x_0	x_1	x_2	x_3	...	Roll	Yaw	anger	disgust	fear	happiness	sadness	surprise	neutral	input
0	0	4.420787	-0.212259	37.338142	48.322983	0.80321	5.465919	5.57361	6.345286	7.812287	...	-1.257063	-0.45221	0.999492	0.000101	0.000131	0.000189	0.000002	0.000079	0.000006	images\anger\10154.jpg
1	1	1.568839	1.130273	34.597195	45.819508	0.969486	6.71379	7.089191	7.940617	9.390182	...	13.945234	47.604616	0.999814	0.000029	0.000053	0.00006	0.000001	0.000041	0.000003	images\anger\117.jpg
2	2	2.253817	-0.586807	39.564362	51.220837	0.959381	4.784771	3.508948	2.882077	2.8502	...	-3.289934	-9.403058	0.893103	0.075909	0.004582	0.001043	0.01655	0.004363	0.004449	images\anger\13675.jpg
3	3	8.023793	3.277001	34.895699	41.650475	0.995364	4.181868	4.755292	6.000817	7.787922	...	-1.908027	5.804783	0.998642	0.000029	0.000246	0.000171	0.000053	0.000759	0.000099	images\anger\15471.jpg
4	4	5.486054	2.378933	37.340008	45.483555	0.959245	6.705106	7.438615	8.374623	9.714442	...	-6.976108	4.558059	0.937213	0.001742	0.000348	0.000179	0.025373	0.00021	0.034935	images\anger\1558.jpg
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
170	170	1.242419	-3.257163	38.798569	55.596889	0.764192	3.380751	2.296906	2.087315	2.932202	...	1.292842	-10.128955	0.020178	0.000295	0.016201	0.000964	0.00366	0.93779	0.020912	images\surprise\61.jpg
171	171	5.536549	-0.476156	34.460732	43.026371	0.940884	1.395814	2.007655	3.205238	5.235259	...	-12.333209	7.169076	0.100073	0.000097	0.026983	0.006621	0.003553	0.746252	0.116421	images\surprise\6419.jpg
172	172	2.495527	1.249969	40.716858	49.492813	0.721187	9.044719	10.154835	11.856644	13.907263	...	-25.37455	2.710789	0.000579	0.000001	0.001267	0.000035	0.000041	0.99796	0.000116	images\surprise\7034.jpg
173	173	6.408597	2.893928	38.228298	47.306564	0.901755	2.15712	4.222015	6.931758	10.174375	...	-2.029263	3.520131	0.133269	0.000318	0.009837	0.012914	0.000159	0.836194	0.007309	images\surprise\8698.jpg
174	174	3.786113	1.826852	39.762043	48.248592	0.93119	6.148463	6.562805	7.575388	8.934242	...	-3.851691	-11.247305	0.001869	0.000002	0.003112	0.000148	0.000738	0.989016	0.005115	images\surprise\88.jpg

Figure 10 Features extracted from the images

The next step is to save all the tabulated data to a CSV file after extracting the face features from each image. The facebox, AUs score, and face position will all be included in the tabulated CSV file. The method from Figure 11.1 will allow us to get the features extracted from the images in each class and convert them into a CSV file. Once we get the csv file with all the columns of interests, we then merge the previous csv file with the labels to it as seen in Figure 11.3. All the data should now be combined into one single CSV file containing all the emotion classes, as seen in Figure 11.4.

```python
# Convert FEX to .csv
def fexToCsv(fex):
    facebox_fex = fex.facebox
    aus_fex = fex.aus
    facepose_fex = fex.facepose

    facebox_df = pd.DataFrame(facebox_fex)
    aus_df = pd.DataFrame(aus_fex)
    facepose_df = pd.DataFrame(facepose_fex)

    result_df = pd.concat([facebox_df, aus_df, facepose_df], axis=1, join='inner')
    result_df.to_csv('emotionsinterest.csv', index=False)

fexToCsv(mixed_prediction)
```

Figure 11.1 Converting the Fex data to a CSV file

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
1	FaceRectX	FaceRectY	FaceRectW	FaceRectH	FaceScore	AU01	AU02	AU04	AU05	AU06	AU07	AU09	AU10	AU11	AU12	AU14	AU15	AU17	AU20	AU23	AU24	AU25	AU26	AU28	AU43	Pitch	Roll	Yaw
2	4.420787	-0.21226	37.33814	48.32298	0.80321	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	1	1	0	0	0	-14.7084	-1.25706	-0.45221
3	1.568839	1.130273	34.59719	45.81951	0.969486	0	0	0	1	0	0	0	0	1	0	1	0	0	0	1	1	0	0	0	0	10.26451	13.94523	47.60462
4	2.253817	-0.58681	39.56436	51.22084	0.959381	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-0.30705	-3.28993	-9.40306
5	8.023793	3.277001	34.8957	41.65047	0.995364	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	15.23892	-1.90803	5.804783
6	5.486054	2.378933	37.34001	45.48355	0.959245	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	1	0	-3.4572	-6.97611	4.558059
7	3.908175	0.051001	37.96545	52.14682	0.78632	0	0	1	0	0	0	0	1	0	0	1	0	1	1	0	1	0	0	0	0	-4.24947	9.407203	-3.55693
8	8.120564	2.9598	37.87998	46.36599	0.625188	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	5.377179	-5.70019	3.097113
9	5.594892	-2.12369	35.79593	43.5168	0.853117	0	1	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	1	0	0	-17.3466	-10.3362	5.878182
10	2.132802	-1.33541	42.50682	50.59157	0.963297	1	1	1	0	1	1	1	1	1	1	0	0	0	1	0	0	1	1	0	0	-7.1939	-15.9349	3.682726
11	0.077703	-4.95872	49.1109	60.05334	0.896314	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	-13.6299	-7.21763	23.0463
12	7.250512	0.837546	36.64266	47.63544	0.970731	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6.804367	3.759684	3.764784
13	3.355298	-0.00121	32.41553	41.03495	0.994922	0	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	0	0	0	0	-6.79418	-0.40489	-0.83889
14	7.764138	4.130642	36.92719	45.40901	0.977502	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	7.587843	-1.13703	15.02902
15	6.279296	0.507933	37.9538	45.20261	0.794922	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	6.483654	5.963393	6.257051
16	4.579592	0.862261	37.76651	47.72536	0.934823	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	1	0	-9.51784	1.021501	2.69864
17	6.286959	0.92731	32.17962	42.33619	0.982986	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	-3.65821	-1.16078	-1.32357
18	3.025513	-0.81953	38.6695	50.48632	0.967009	1	1	0	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	-2.94841	-3.26912	-7.54051
19	-0.28926	-2.76511	40.66259	54.79255	0.950243	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	11.90327	-9.96382	-21.2638
20	5.938268	6.411468	36.49416	43.11793	0.912805	1	1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	4.546521	-0.01104	10.42936
21	5.580225	1.549541	40.40001	48.52195	0.93339	1	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	1	1	0	0	0.402494	2.018407	-1.76114
22	-0.18423	1.106021	39.08453	47.2288	0.921842	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1	1	0	0	9.272866	8.124351	16.05533
23	6.301706	6.776884	33.083	39.6397	0.997905	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	2.365857	-2.93018	5.578523
24	6.801912	-0.27674	38.59058	46.94227	0.879743	0	0	1	0	0	0	0	1	1	0	1	0	1	0	1	1	0	0	0	0	-1.1018	-16.9684	5.88174
25	4.838948	2.112511	39.25636	47.60329	0.953006	0	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	6.853572	-6.17777	2.24616
26	3.104146	1.80235	37.4917	44.26422	0.907245	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	4.084557	-3.21101	-9.92384
27	1.278261	2.118862	41.36033	47.31823	0.989313	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	2.497608	-7.19315	-5.29931

Figure 11.2 Output of the Extracted Features without labels

```python
# Merge the csv with labels to the csv with the columns of interest (AUs, facebox, etc)
emotions_label = pd.read_csv('./emotionslabel.csv')
emotions_interests = pd.read_csv('./emotionsinterest.csv')

emotions = pd.concat([emotions_label, emotions_interests], axis=1, join='outer')
emotions.to_csv('emotionsdataset.csv', index=True)
```

Figure 11.3 Code for merging of the two CSV files

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD
1		label	FaceRect(	FaceRect(	FaceRect(	FaceRect(	FaceScore	AU01	AU02	AU04	AU05	AU06	AU07	AU09	AU10	AU11	AU12	AU14	AU15	AU17	AU20	AU23	AU24	AU25	AU26	AU28	AU43	Pitch	Roll	Yaw
2	0	anger	4.420787	-0.21226	37.33814	48.32298	0.80321	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	1	1	0	0	0	-14.7084	-1.25706	-0.45221
3	1	anger	1.568839	1.130273	34.59719	45.81951	0.969486	0	0	0	1	0	0	0	0	1	0	1	0	0	0	1	1	0	0	0	0	10.26451	13.94523	47.60462
4	2	anger	2.253817	-0.58681	39.56436	51.22084	0.959381	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-0.30705	-3.28993	-9.40306
5	3	anger	8.023793	3.277001	34.8957	41.65047	0.995364	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	15.23892	-1.90803	5.804783
6	4	anger	5.486054	2.378933	37.34001	45.48355	0.959245	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	1	0	-3.4572	-6.97611	4.558059
7	5	anger	3.908175	0.051001	37.96545	52.14682	0.78632	0	0	1	0	0	0	0	1	0	0	1	0	1	1	0	1	0	0	0	0	-4.24947	9.407203	-3.55693
8	6	anger	8.120564	2.9598	37.87998	46.36599	0.625188	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	5.377179	-5.70019	3.097113
9	7	anger	5.594892	-2.12369	35.79593	43.5168	0.853117	0	1	0	0	0	0	0	1	0	1	1	0	0	0	0	0	1	1	0	0	-17.3466	-10.3362	5.878182
10	8	anger	2.132802	-1.33541	42.50682	50.59157	0.963297	1	1	1	0	1	1	1	1	1	1	0	0	0	1	0	0	1	1	0	0	-7.1939	-15.9349	3.682726
11	9	anger	0.077703	-4.95872	49.1109	60.05334	0.896314	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	-13.6299	-7.21763	23.0463
12	10	anger	7.250512	0.837546	36.64266	47.63544	0.970731	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6.804367	3.759684	3.764784
13	11	anger	3.355298	-0.00121	32.41553	41.03495	0.994922	0	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	0	0	0	0	-6.79418	-0.40489	-0.83889
14	12	anger	7.764138	4.130642	36.92719	45.40901	0.977502	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	7.587843	-1.13703	15.02902
15	13	anger	6.279296	0.507933	37.9538	45.20261	0.794922	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	6.483654	5.963393	6.257051
16	14	anger	4.579592	0.862261	37.76651	47.72536	0.934823	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	1	0	-9.51784	1.021501	2.69864
17	15	anger	6.286959	0.92731	32.17962	42.33619	0.982986	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	-3.65821	-1.16078	-1.32357
18	16	anger	3.025513	-0.81953	38.6695	50.48632	0.967009	1	1	0	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	-2.94841	-3.26912	-7.54051
19	17	anger	-0.28926	-2.76511	40.66259	54.79255	0.950243	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	11.90327	-9.96382	-21.2638
20	18	anger	5.938268	6.411468	36.49416	43.11793	0.912805	1	1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	4.546521	-0.01104	10.42936
21	19	anger	5.580225	1.549541	40.40001	48.52195	0.93339	1	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	1	1	0	0	0.402494	2.018407	-1.76114
22	20	anger	-0.18423	1.106021	39.08453	47.2288	0.921842	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1	1	0	0	9.272866	8.124351	16.05533
23	21	anger	6.301706	6.776884	33.083	39.6397	0.997905	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	2.365857	-2.93018	5.578523
24	22	anger	6.801912	-0.27674	38.59058	46.94227	0.879743	0	0	1	0	0	0	0	1	1	0	1	0	1	0	1	1	0	0	0	0	-1.1018	-16.9684	5.88174
25	23	anger	4.838948	2.112511	39.25636	47.60329	0.953006	0	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	6.853572	-6.17777	2.24616
26	24	anger	3.104146	1.80235	37.4917	44.26422	0.907245	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	4.084557	-3.21101	-9.92384

Figure 11.4 Output of the Extracted Features with labels

III. Splitting Data using Stratified Sampling

Before we start training the model classifiers, we first split our dataset into two subsets, which are the training set and the testing set. The training set consists of 70% of the dataset, while the testing set comprises 30% of the dataset. In addition, stratified sampling was performed, as evidenced by the train_test_split's parameters.

Furthermore, we wanted to use normalized data to train our model, so we applied the MinMaxScaler to our training data. This is done by calling the *fit_transform()* function. This data scaling is an important pre-processing step since we are working with many machine learning algorithms.

```
# Training and Testing Data (creating the model)
# Note: The data will be divided in a ratio of 70:30 (training data:testing data) through stratified sampling
col_names = ['label','FaceRectX', 'FaceRectY', 'FaceRectWidth',
             'FaceRectHeight','FaceScore','AU01','AU02','AU04','AU05','AU06','AU07',
             'AU09','AU10','AU11','AU12','AU14','AU15','AU17','AU20',
             'AU23','AU24','AU25','AU26','AU28','AU43','Pitch','Roll',
             'Yaw']
emotions = pd.read_csv('emotionsdataset.csv')

# FaceRectX, FaceRectY, FaceRectWidth, FaceRectHeight, FaceScore, Pitch, Roll, Yaw
x = emotions.iloc[:,[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28]].values
# label
y = emotions.iloc[:,1].values

x_train,x_test,y_train,y_test = train_test_split(x, y, test_size=0.3, random_state=100, stratify=y)
```

Figure 12.1 Splitting of Data using Stratified sampling

```
from sklearn.preprocessing import MinMaxScaler
s = MinMaxScaler()
x_train = s.fit_transform(x_train)
x_test = s.fit_transform(x_test)
```

Figure 12.2 Normalizing the dataset using the scikit-learn object MinMaxScaler

IV. Model Development and Testing

1. SVM Classifier

The first classifier we used to train the model was the SVM (Support vector machines) classifier. We trained our model using the SVM classifier from the Sklearn Python package. As stated previously, our test/train split was 30% and 70%.

```python
# Training the SVM Classifier

svm_clf = svm.SVC(kernel='linear')
svm_clf.fit(x_train, y_train)
svm_predictions = svm_clf.predict(x_test)

print("Test Classes:")
print(y_test)

print("Predicted classes:")
print(svm_predictions)

print(classification_report(y_test, svm_predictions))
cm = confusion_matrix(y_test, svm_predictions, labels=svm_clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=svm_clf.classes_)
disp.plot()
plt.show()
```

Figure 13.1 SVM Classifier Code

Figure 13.2 shows the model testing for the SVM Classifier. Figures 13.3 and 13.4 are visualizations used for us to analyze on the performances of this model.

```
Test Classes:
['neutral' 'surprise' 'neutral' 'anger' 'neutral' 'surprise' 'anger'
 'neutral' 'happy' 'fear' 'sad' 'anger' 'neutral' 'anger' 'happy' 'sad'
 'happy' 'disgust' 'anger' 'surprise' 'disgust' 'disgust' 'neutral'
 'anger' 'sad' 'surprise' 'disgust' 'happy' 'surprise' 'anger' 'happy'
 'anger' 'sad' 'fear' 'fear' 'surprise' 'surprise' 'neutral' 'fear'
 'surprise' 'fear' 'sad' 'disgust' 'neutral' 'happy' 'disgust' 'sad'
 'disgust' 'sad' 'happy' 'happy' 'fear' 'fear']
Predicted classes:
['sad' 'fear' 'neutral' 'sad' 'neutral' 'surprise' 'neutral' 'disgust'
 'anger' 'anger' 'disgust' 'sad' 'neutral' 'disgust' 'fear' 'neutral'
 'happy' 'disgust' 'neutral' 'surprise' 'neutral' 'neutral' 'fear'
 'disgust' 'neutral' 'anger' 'neutral' 'happy' 'surprise' 'happy'
 'disgust' 'fear' 'happy' 'surprise' 'fear' 'neutral' 'neutral' 'neutral'
 'surprise' 'surprise' 'fear' 'neutral' 'surprise' 'fear' 'sad' 'anger'
 'neutral' 'happy' 'disgust' 'surprise' 'happy' 'surprise' 'anger']
```

Figure 13.2 SVM Model Testing

	precision	recall	f1-score	support
anger	0.00	0.00	0.00	8
disgust	0.14	0.14	0.14	7
fear	0.29	0.29	0.29	7
happy	0.50	0.38	0.43	8
neutral	0.27	0.50	0.35	8
sad	0.00	0.00	0.00	7
surprise	0.44	0.50	0.47	8
...				
accuracy			0.26	53
macro avg	0.23	0.26	0.24	53
weighted avg	0.24	0.26	0.24	53

Figure 13.3 SVM Classification Report

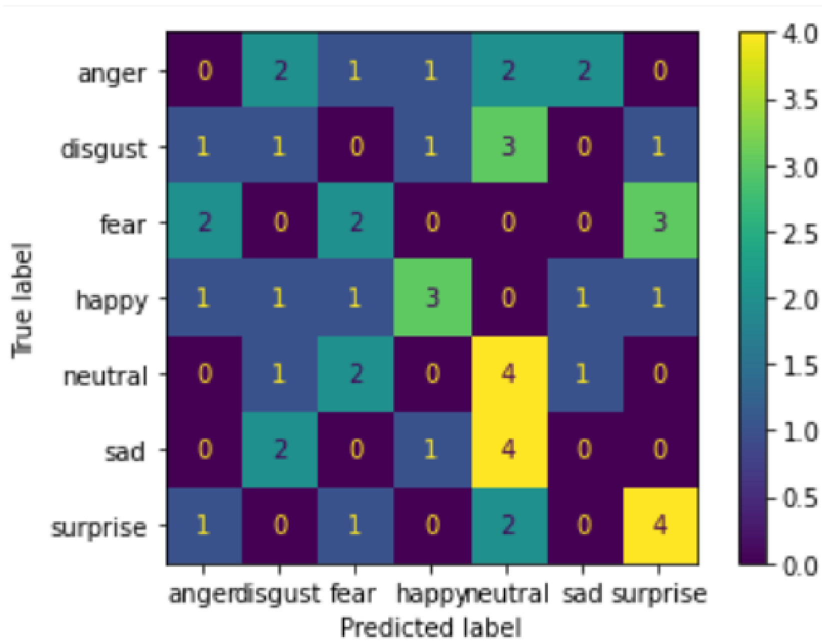

Figure 13.4 SVM Confusion Matrix

2. Decision Tree Classifier

The Decision Tree Classifier is the second model that we selected. As stated previously, our test/train split was 30% and 70%. For our decision tree parameters, we use gini because it determines how well a decision tree was split. Basically, it helps us to determine which splitter is best so that we can build a pure decision tree. We then use the training dataset for our decision tree classifier.

```python
# Training the Decision Tree Classifier using Gini index attribute
clf_model = DecisionTreeClassifier(criterion="gini", random_state=100, max_depth=None, min_samples_leaf=3)
clf_model.fit(x_train,y_train)
tree_predict = clf_model.predict(x_test)

print("Test Classes:")
print(y_test)

print("Predicted classes:")
print(tree_predict)

print(classification_report(y_test, tree_predict))
cm = confusion_matrix(y_test, tree_predict, labels=clf_model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf_model.classes_)
disp.plot()
plt.show()
```

Figure 14.1 Decision Tree Classifier Code

Just like the SVM Classifier, we utilized model testing and visualize the performance by creating a classification report and a confusion matrix.

```
Test Classes:
['neutral' 'surprise' 'neutral' 'anger' 'neutral' 'surprise' 'anger'
 'neutral' 'happy' 'fear' 'sad' 'anger' 'neutral' 'anger' 'happy' 'sad'
 'happy' 'disgust' 'anger' 'surprise' 'disgust' 'disgust' 'neutral'
 'anger' 'sad' 'surprise' 'disgust' 'happy' 'surprise' 'anger' 'happy'
 'anger' 'sad' 'fear' 'fear' 'surprise' 'surprise' 'neutral' 'fear'
 'surprise' 'fear' 'sad' 'disgust' 'neutral' 'happy' 'disgust' 'sad'
 'disgust' 'sad' 'happy' 'happy' 'fear' 'fear']
Predicted classes:
['sad' 'fear' 'sad' 'anger' 'fear' 'neutral' 'anger' 'sad' 'happy' 'anger'
 'happy' 'happy' 'neutral' 'neutral' 'anger' 'sad' 'happy' 'happy'
 'disgust' 'surprise' 'disgust' 'fear' 'sad' 'happy' 'neutral' 'happy'
 'anger' 'happy' 'surprise' 'happy' 'sad' 'disgust' 'anger' 'surprise'
 'neutral' 'disgust' 'anger' 'neutral' 'sad' 'sad' 'anger' 'sad' 'sad'
 'disgust' 'happy' 'fear' 'anger' 'happy' 'anger' 'happy' 'happy'
 'neutral' 'anger']
```

Figure 14.2 Decision Tree Model Testing

	precision	recall	f1-score	support
anger	0.18	0.25	0.21	8
disgust	0.20	0.14	0.17	7
fear	0.00	0.00	0.00	7
happy	0.46	0.75	0.57	8
neutral	0.29	0.25	0.27	8
sad	0.20	0.29	0.24	7
surprise	0.67	0.25	0.36	8
...				
accuracy			0.28	53
macro avg	0.29	0.28	0.26	53
weighted avg	0.29	0.28	0.27	53

Figure 14.3 Decision Tree Classification Report

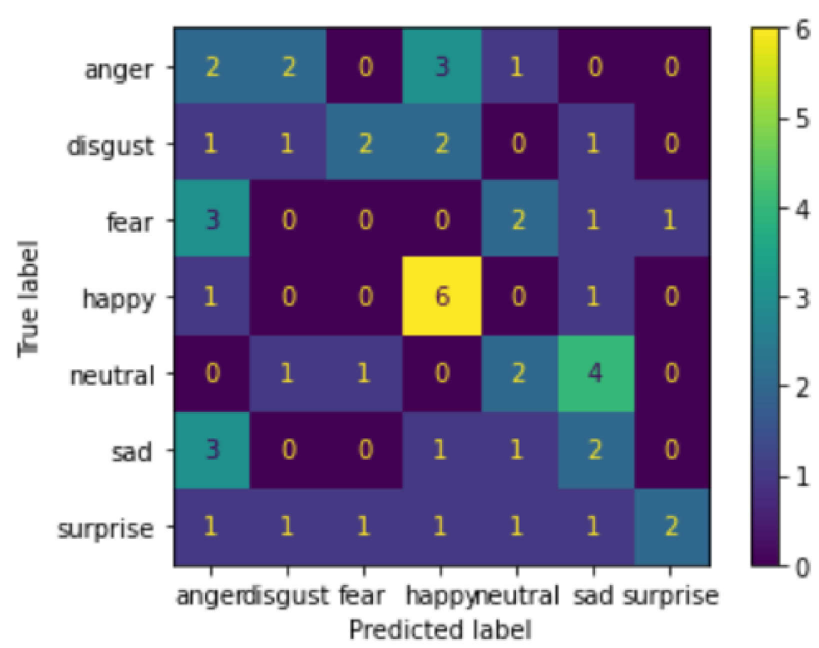

Figure 14.4 Decision Tree Confusion Matrix

3. KNN Classifier

Another model used to classify the extracted features dataset is the KNN Classifier. Before applying the KNeighborsClassifier class to implement this third classifier, the derived dataset was initially split into training and testing data, where 30% was allocated for training and 70% for testing. To perform the KNN classification, the KNeighborsClassifier was called to define an attribute *k* (n_neighbors) equal to 30. The images below show the test and predicted classes and their classification report derived from the dataset. A confusion matrix was also displayed to allow a better and visual understanding of the data.

```
# Training the KNN Classifier
knn = KNeighborsClassifier(n_neighbors = 30)
knn.fit(x_train, y_train)
knn_predictions = knn.predict(x_test)

print("Test Classes:")
print(y_test)

print("Predicted classes:")
print(knn_predictions)

print(classification_report(y_test, knn_predictions))
cm = confusion_matrix(y_test, knn_predictions, labels=svm_clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=svm_clf.classes_)
disp.plot()
plt.show()
```

Figure 15.1 KNN Classifier Code

```
Test Classes:
['neutral' 'surprise' 'neutral' 'anger' 'neutral' 'surprise' 'anger'
 'neutral' 'happy' 'fear' 'sad' 'anger' 'neutral' 'anger' 'happy' 'sad'
 'happy' 'disgust' 'anger' 'surprise' 'disgust' 'disgust' 'neutral'
 'anger' 'sad' 'surprise' 'disgust' 'happy' 'surprise' 'anger' 'happy'
 'anger' 'sad' 'fear' 'fear' 'surprise' 'surprise' 'neutral' 'fear'
 'surprise' 'fear' 'sad' 'disgust' 'neutral' 'happy' 'disgust' 'sad'
 'disgust' 'sad' 'happy' 'happy' 'fear' 'fear']
Predicted classes:
['sad' 'surprise' 'neutral' 'sad' 'anger' 'surprise' 'anger' 'happy'
 'happy' 'neutral' 'fear' 'surprise' 'neutral' 'disgust' 'neutral' 'anger'
 'happy' 'happy' 'fear' 'surprise' 'neutral' 'fear' 'neutral' 'fear'
 'neutral' 'happy' 'disgust' 'happy' 'surprise' 'happy' 'happy' 'anger'
 'happy' 'fear' 'neutral' 'anger' 'neutral' 'neutral' 'surprise'
 'surprise' 'fear' 'neutral' 'surprise' 'neutral' 'happy' 'anger'
 'neutral' 'happy' 'happy' 'happy' 'happy' 'surprise' 'surprise']
```

Figure 15.2 KNN Classifier Model Testing

	precision	recall	f1-score	support
anger	0.33	0.25	0.29	8
disgust	0.50	0.14	0.22	7
fear	0.33	0.29	0.31	7
happy	0.50	0.88	0.64	8
neutral	0.38	0.62	0.48	8
sad	0.00	0.00	0.00	7
surprise	0.50	0.62	0.56	8
...				
accuracy			0.42	53
macro avg	0.36	0.40	0.35	53
weighted avg	0.37	0.42	0.36	53

Figure 15.3 KNN Classifier Classification Report

Figure 15.4 KNN Confusion Matrix

V. Analysis and Interpretation

1. **SVM Classifier**

   SVMs (Support Vector Machines) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis (GeeksforGeeks, 2022). The SVM classifier was used for training and developing our model. We visualized its performance by creating a classification report (Figure 13.3) and confusion matrix (Figure 13.4). Based on the classification report, it had an accuracy of 26% which meant that the model is not accurate and was doing poorly in predicting the correct emotions as seen in Figure 13.2. The highest precision score (50%) was from the emotion 'happy' and the highest recall score (50%) was tied between 'neutral' and 'surprise'. It can be observed that the lowest scores (0%) all together was from 'anger' and 'sad'. Additionally, based on the confusion matrix, SVM is good at detecting 'neutral' and 'surprise'. Nevertheless, as we can see from both the classification report and confusion matrix, it is doing poorly in detecting the emotions 'anger' and 'sad'.

2. **Decision Tree Classifier**

   As its name suggests, the Decision Tree Classification algorithm is a "tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome." This type of algorithm produces a graphical representation of all the possible situations to a decision, or test, which are performed based on a given dataset. (Machine Learning Decision Tree Classification Algorithm - Javatpoint, n.d.) To enable a better understanding of this algorithm's influence on our emotion dataset, we displayed the tested and predicted classes through a classification report (Figure 14.3) and a confusion matrix (Figure 14.4). From the classification report, we see that the emotion 'happy' had the highest value for recall (0.75) and f1-score (0.57), and 'surprise' for precision (0.67). From the confusion matrix, the emotion 'happy' had the highest accuracy, based on the relationship between the true and predicted values. Although 'happy' was considered as the emotion with the highest accuracy, the Decision Tree classifier was only able to reach an accuracy score of 0.28 for the entire dataset, even after applying the model on the data. Since the decision tree classifier is more appropriate for determining the best solution to a problem, it is understandable that the accuracy score is not as ideal as the accuracy score produced by a KNN classifier. Compared to the KNN classifier, which produced a relatively good accuracy of 0.42, the Decision Tree classifier only produced an accuracy of 0.28 because of its nature to classify problems or data based on an answer of true or false (1 or 0). Therefore, Decision Tree was only good at identifying 'happy' and 'surprise' as it gave them higher scores than the rest of the other emotions.

3. **KNN Classifier**

   The K-Nearest Neighbor (KNN) Classification algorithm identifies data by comparing the similarities between the training and testing data derived from a given dataset. (Figure 15.2) To better observe the similarities between these two kinds of data, we illustrated their accuracy (similarities) through a classification report and confusion matrix. According to the classification report, implementing the KNN model on the data produced an accuracy rate of 0.42 and the highest precision, recall, and f1-score of 0.50, 0.88, and 0.64, respectively, for the emotion ‘happy.’ However, we noticed that it failed to predict any images for the emotion ‘sad’, as it resulted in 0% for precision, recall, and f1 score. According to the confusion matrix, on the other hand, ‘happy’ also had the highest accuracy, based on the relationship between the dataset’s true and predicted values. These values were derived under the condition of n_neighbors (*k*) being equal to 30. Additionally, these results were derived through trial and error and by finding the most appropriate value for *k* that will yield the highest, possible accuracy score. While we were figuring out which value of *k* was most appropriate for our dataset, we observed that *k* could only produce the highest, possible accuracy if it stayed below the value of 34 and above the value of 30. Any value beyond or below the stated values will always yield a lower accuracy score and a different emotion identified as the one with the highest accuracy, precision, recall, and f1-score.

VI. Conclusion

The group concluded that out of the three models that were developed, the KNN Classifier had the highest accuracy when it came to classifying emotions through facial expressions. However, it still must be improved to include and predict all emotions as it was unable to predict the emotion ‘sad’. It is observed that the emotion 'happy' fared the best out of all the other emotions regardless of the model. The emotions that consecutively failed to be predicted were 'anger' and 'sad’.

The other two models, Decision Tree and SVM, had poor performances since they were not able to predict emotions accurately. We gathered that all three model’s performances may be due to the lack of proper imagery and quality. It would be better if we had picked better quality images that clearly shows the emotions that we want to predict. We had also noticed that while we were extracting features, most of the AU, facepose, and facebox scores does not seem to be consistent for one emotion class. This may be due to how Py-Feat Detector perceives the image since the person could be exhibiting multiple emotions such as being surprised while being happy. Additionally, we recommend that it would be better to extract features that clearly shows which emotion class dominates in.

References

C. (n.d.). *Py-feat/02_detector_imgs.ipynb at main · cosanlab/py-feat*. GitHub. https://github.com/cosanlab/py-feat/blob/main/docs/basic_tutorials/02_detector_imgs.ipynb

GeeksforGeeks. (2022, June 29). *Classifying data using support vector machines(SVMs) in python*. https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-python/

Hagen, C. (2022, January 5). *Using Py-Feat to plot facial expression predictions*. Medium. https://towardsdatascience.com/using-py-feat-to-plot-facial-expression-predictions-86a9064990ce

K-Nearest neighbor(KNN) algorithm for machine learning - javatpoint. (n.d.). Www.Javatpoint.Com. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

Machine learning decision tree classification algorithm - javatpoint. (n.d.-a). Www.Javatpoint.Com. https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

Machine learning decision tree classification algorithm - javatpoint. (n.d.-b). Www.Javatpoint.Com. https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm