I confirm that I have not used any GPT-generated response for any part of this Homework.

1. Feature Analysis

a. Extract Raw Features of Pitch and Intensity using Praat and Parselmouth.

: (1) Feature Array filtered NaN or 0 values, (2) Min, (3) Max, (4) Mean.

To extract raw pitch, I used the “To Pitch (ac)” function based on the autocorrelation method. The pitch floor was set to 75 Hz and the pitch ceiling to 600 Hz, as specified in the instructions. All other parameters were left at their default values from the official documentation. I used pitch to extract the full pitch array from a given file path pitch.selected_array[“frequency”] and filtered out meaningless values such as NaNs and zeros.

For raw intensity extraction, I used the “To Intensity” function with a pitch floor of 75 Hz and a time step of 0.0, as instructed in the guidelines. Also, I extracted only the left audio channel through intensity.values[0], and again removed any NaNs and zeros values.

The code snippets below show the exact implementation for extracting raw pitch and intensity values:

```python
def extract_raw_pitch(file_path, time_step=0.0, pitch_floor=75, pitch_ceiling=600):
    snd = parselmouth.Sound(file_path)

    # Pitch extraction (autocorrelation, 75–600 Hz)
    """
        To Pitch (ac): time_step, pitch_floor, max_candidates, very_accurate,
                       silence_threshold, voicing_threshold, octave_cost, octave_jump_cost,
                       voiced_unvoiced_cost, pitch_ceiling
    """
    pitch = call(snd, "To Pitch (ac)", time_step, pitch_floor,
                 15, "off", 0.03, 0.45, 0.01, 0.35, 0.14, pitch_ceiling)
    pitch_array = pitch.selected_array["frequency"]

    # exclude values that are 0 or NaN
    pitch_array = pitch_array[~np.isnan(pitch_array)]
    pitch_array = pitch_array[pitch_array>0]

    return pitch_array
```

```python
def extract_raw_intensity(file_path, time_step=0.0, pitch_floor=75):
    snd = parselmouth.Sound(file_path)

    # Intensity extraction (75 Hz floor) & use only left channel
    subtract_mean = "yes"
    intensity = call(snd, "To Intensity", pitch_floor, time_step, subtract_mean)
    intensity_array = intensity.values[0] # use only channel 1

    # exclude values that are 0 or NaN
    intensity_array = intensity_array[~np.isnan(intensity_array)]
    intensity_array = intensity_array[intensity_array>0]

    return intensity_array
```

These extracted raw pitch and intensity arrays were saved for each speech segment. In addition, I computed the min, max, and mean values from each raw array and stored them alongside the arrays. All of these values, both the raw feature arrays and the extracted statistics, were saved into a unified DataFrame called raw_df, as shown in the table below.

	filename	speaker	emotion	pitch_array	intensity_array	pitch_min_raw	pitch_max_raw	pitch_mean_raw	intensity_min_raw	intensity_max_raw	intensity_mean_raw
0	mf_001_interest_2590.84_Eight-hundred-two.wav	mf	interest	[147.61835093558605, 146.88637319738152, 155.7...	[67.59469365894608, 67.6986717817256, 67.45391...	136.706006	396.699742	298.640085	36.614690	68.700588	55.459283
1	mf_001_anxiety_1171.28_Six-hundred-one.wav	mf	anxiety	[162.04235108975703, 159.36853694759952, 155.7...	[45.12859828944403, 47.51880799670061, 50.0611...	146.910323	240.097190	164.339495	45.128598	71.034028	63.217935
2	cc_001_pride_2501.34_March-twenty-fifth.wav	cc	pride	[137.41666301265246, 136.17449197038164, 141.3...	[38.29787682413948, 50.03917124188712, 57.6147...	92.859850	563.976473	150.376392	36.655536	70.336489	54.616907
3	cc_001_boredom_2278.62_Six-hundred-six.wav	cc	boredom	[111.36167226450702, 112.03084373200733, 107.9...	[33.667183652346885, 31.687729792455265, 32.76...	88.458167	218.513929	128.143332	31.687730	67.037833	54.804441
4	jg_001_panic_443.70_Fifty-seven.wav	jg	panic	[232.7958724531116, 219.80523357237019, 207.409...	[30.59913120724033, 32.4447219921277, 35.22801...	146.097492	232.795872	177.215006	29.769395	63.692273	45.360042

b. Please specify **your normalization** method and provide a detailed description of **how you calculated it** and **why you chose it**.

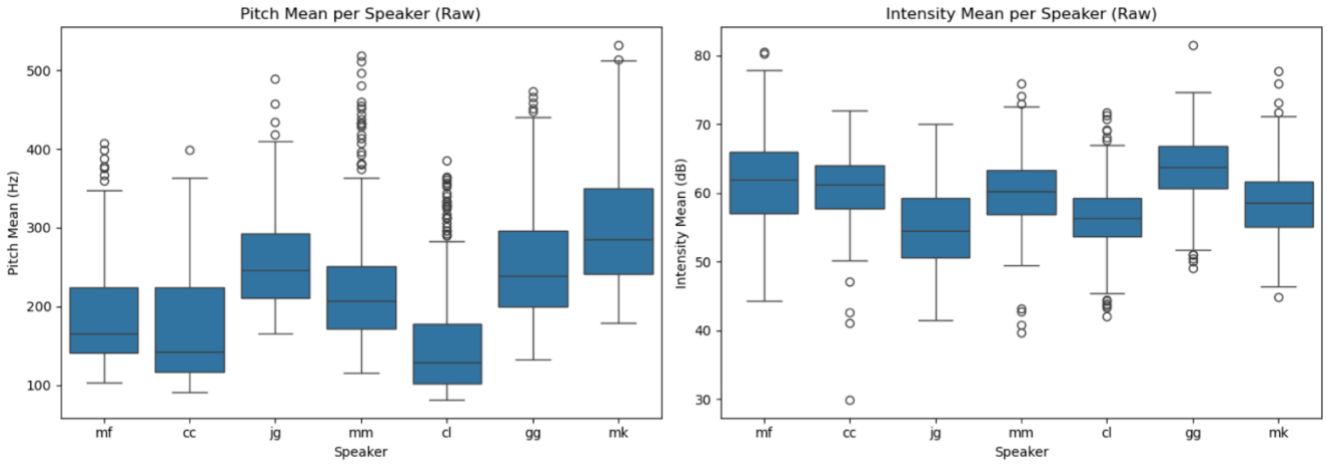
Normalization method: Z-score normalization over the individual speaker

I chose z-score normalization over individual speakers for this task. When selecting a normalization strategy, it is important to consider whether (1) the method reflects the global distribution and (2) the method generalizes well across all speakers without being biased toward a particular emotion. Z-score normalization meets these criteria, as it uses the mean and standard deviation calculated from all pitch or intensity values of each individual speaker. This ensures that normalization is statistically robust across all segments for each speaker, and the normalized values represent relative deviation from speaker-specific baselines.

As shown in the summary statistic (describe()) of the raw features, the minimum of pitch_mean_raw is approximately 80.95 Hz, and the maximum is 532.33 Hz, with a standard deviation of 87.14 Hz, which spans a wide range of 451 Hz. Similarly, intensity_mean_raw ranges from 29.86 dB to 81.52 dB, with a range of over 50 dB. These wide variations highlight the need for normalization.

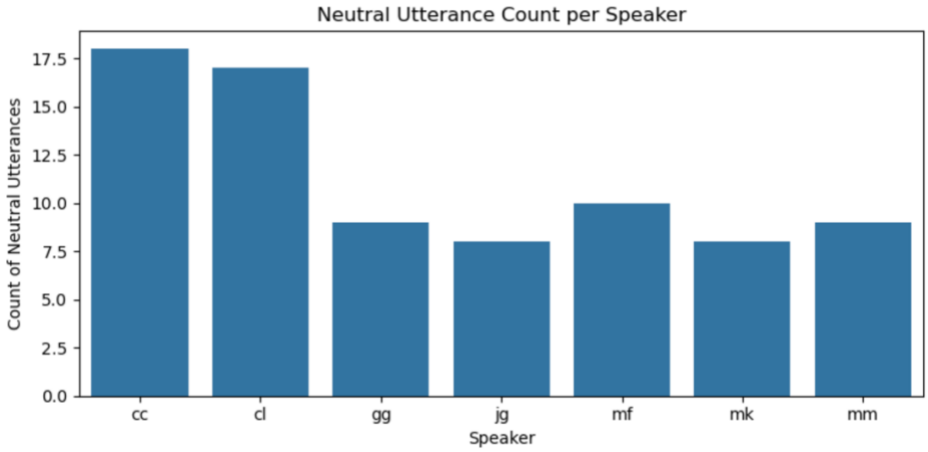
	pitch_min_raw	pitch_max_raw	pitch_mean_raw	intensity_min_raw	intensity_max_raw	intensity_mean_raw
count	2324.000000	2324.000000	2324.000000	2324.000000	2324.000000	2324.000000
mean	144.508300	374.966690	225.885146	37.057653	75.941924	59.574162
std	61.613467	155.973279	87.139629	7.040124	6.079626	6.119665
min	74.886730	98.522441	80.950962	7.670332	53.403073	29.859855
25%	92.140956	240.037827	157.639435	32.550513	71.949909	55.505481
50%	129.850187	368.611989	218.158854	37.214278	75.904770	59.565277
75%	182.589798	531.605007	276.865451	41.911083	80.132789	63.617430
max	464.472139	599.990828	532.334435	67.476922	92.928329	81.521196

To investigate the source of this variation, I visualized the distributions of pitch and intensity means across speakers using box plots. For example, in the pitch plot, speaker “cl” shows a median around 120 Hz, while speaker “mk” exceed 270 Hz. A similar trend is observed in the intensity plot: speaker “jp” averages around 55 dB, while “mf” exceeds 65 dB. Although intensity is generally less speaker-sensitive than pitch, a difference of over 10 dB is substantial enough to cause a classifier to overfit to speaker characteristics. Furthermore, since the emotion expression often depends on relative changes in pitch and intensity, z-score normalization effectively captures deviation from speaker-specific baselines, isolating emotion-related variations. This makes it particularly suitable for emotion classification tasks.



Secondly, I considered and rejected the approach of normalizing based on each speaker's neutral utterances. Although this method aligns pitch features to each speaker's neutral baseline, neutral is itself a labeled emotion in this task. As such, using it for normalization introduces instability and potential bias. As illustrated in the emotion class distribution and the number of neutral utterances per speaker, neutral is underrepresented compared to other emotion classes and unevenly distributed across speakers. Several speakers either lack neutral samples or have very few, making neutral-based normalization unreliable. For this reason, z-score normalization per speaker is a more stable and statistically appropriate method for this dataset.

```
Counter({'contempt': 180,
         'happy': 177,
         'interest': 176,
         'despair': 174,
         'disgust': 172,
         'anxiety': 170,
         'elation': 159,
         'boredom': 154,
         'cold-anger': 154,
         'sadness': 151,
         'pride': 150,
         'shame': 148,
         'panic': 141,
         'hot-anger': 139,
         'neutral': 79})
```



To perform z-score normalization over the individual speaker, I followed the steps below:

i. Collected all raw pitch and intensity arrays for each of the seven speakers.

```python
# 1. collect raw feature array by the individual speaker

speaker_pitch_values = defaultdict(list)
speaker_intensity_values = defaultdict(list)

for _, row in raw_df.iterrows():
    speaker_pitch_values[row["speaker"]].extend(row["pitch_array"])
    speaker_intensity_values[row["speaker"]].extend(row["intensity_array"])
```

ii. Calculated the overall mean and standard deviation of pitch and intensity for each speaker.

```
# 2. get OVERALL mean/std by the individual speakers

def get_speaker_stats(speaker_pitch_values, speaker_intensity_values):
    speaker_stats = {}

    for speaker in raw_df["speaker"].unique():
        pitch_all = np.array(speaker_pitch_values[speaker])
        intensity_all = np.array(speaker_intensity_values[speaker])

        pitch_mean, pitch_std = np.mean(pitch_all), np.std(pitch_all)
        intensity_mean, intensity_std = np.mean(intensity_all), np.std(intensity_all)

        speaker_stats[speaker] = {
            "pitch_mean": pitch_mean,
            "pitch_std": pitch_std,
            "intensity_mean": intensity_mean,
            "intensity_std": intensity_std
        }
    return speaker_stats
```

The resulting output includes the mean and standard deviation of pitch and intensity values per speaker, as shown in the table below.

	pitch_mean	pitch_std	intensity_mean	intensity_std
mf	192.611180	87.027605	61.764497	11.762053
cc	175.528953	91.332255	57.927207	14.569117
jg	255.948127	91.518895	55.344661	14.179310
mm	229.381680	108.017197	60.098054	11.268872
cl	155.349800	103.753304	56.294933	14.840809
gg	256.393090	102.187490	63.409916	11.207490
mk	302.764056	106.242946	58.365196	12.011213

iii. Applied z-score normalization to each segment using the formula (x-mean)/std, where the mean and std were speaker-specific values computed in step 2. This resulted in two normalized arrays per segment: pitch_array_norm and intensity_array_norm. And then, computed the min, max, and mean from each normalized array, and saved these features in a normalized DataFrame.

```
# 3. Normalized (x-mean)/std, which x is the element of the feature array

normalized_rows = []

for _, row in raw_df.iterrows():
    speaker = row["speaker"]
    pitch_array = np.array(row["pitch_array"])
    intensity_array = np.array(row["intensity_array"])

    stats = speaker_stats[speaker]
    pitch_array_norm = (pitch_array - stats["pitch_mean"]) / stats["pitch_std"]
    intensity_array_norm = (intensity_array - stats["intensity_mean"]) / stats["intensity_std"]

    normalized_rows.append({
        "filename": row["filename"],
        "speaker": speaker,
        "emotion": row["emotion"],
        "pitch_array_norm": pitch_array_norm,
        "intensity_array_norm": intensity_array_norm,
        "pitch_min_norm": np.min(pitch_array_norm),
        "pitch_max_norm": np.max(pitch_array_norm),
        "pitch_mean_norm": np.mean(pitch_array_norm),
        "intensity_min_norm": np.min(intensity_array_norm),
        "intensity_max_norm": np.max(intensity_array_norm),
        "intensity_mean_norm": np.mean(intensity_array_norm)
    })

normalized_df = pd.DataFrame(normalized_rows)
```

The normalized DataFrame structure followed the same format as the raw DataFrame, with normalized feature arrays and their corresponding summary statistics stored per segment.

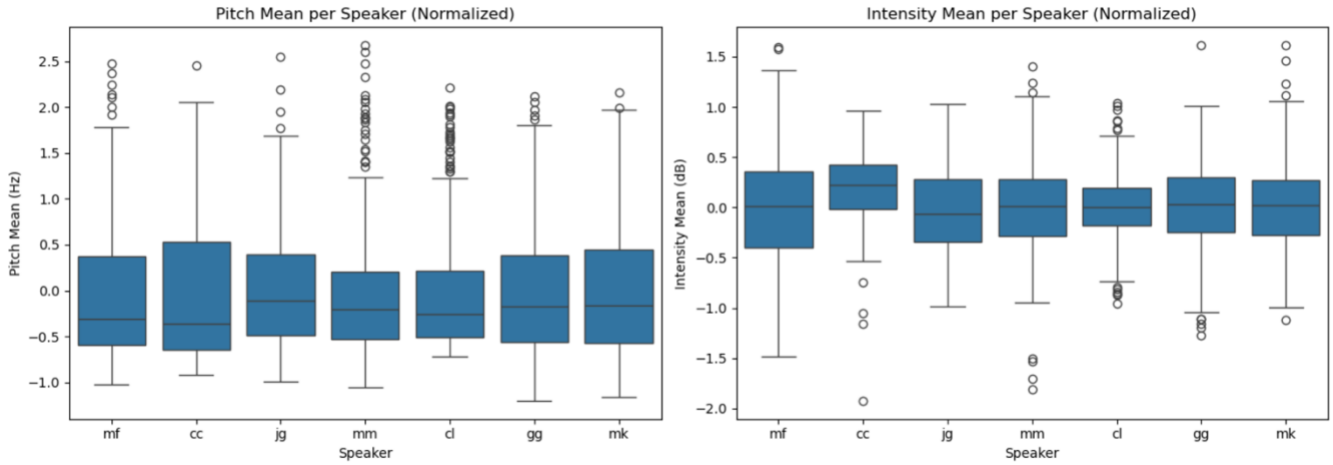
	filename	speaker	emotion	pitch_array_norm	intensity_array_norm	pitch_min_norm	intensity_mean_norm	intensity_max_norm	intensity_min_norm	pitch_mean_norm
0	mf_001_interest_2590.84_Eight-hundred-two.wav	mf	interest	[-0.5169949201976874, -0.5254057886932291, -0....	[0.4956785335088538, 0.5045186675428964, 0.483...	-0.642384	-0.636064	0.589701	-2.198216	1.218336
1	mf_001_anxiety_1171.28_Six-hundred-one.wav	mf	anxiety	[-0.35125440358855503, -0.38197815098881016, -...	[-1.4143703207980474, -1.211156667610397, -0.9...	-0.525131	0.123570	0.788088	-1.414370	-0.324859
2	cc_001_pride_2501.34_March-twenty-fifth.wav	cc	pride	[-0.4172927678640202, -0.43089334123807016, -0...	[-1.3473246724306385, -0.5414217002844751, -0....	-0.905147	-0.227214	0.851753	-1.460052	-0.275396
3	cc_001_boredom_2278.62_Six-hundred-six.wav	cc	boredom	[-0.7025697557919711, -0.6952429744278518, -0....	[-1.6651677702583898, -1.8010341996447055, -1....	-0.953341	-0.214341	0.625338	-1.807034	-0.518827
4	jg_001_panic_443.70_Fifty-seven.wav	jg	panic	[-0.2529778638224491, -0.39492272407209283, -0...	[-1.7451857893890037, -1.6150249530038514, -1....	-1.200306	-0.704168	0.588718	-1.803703	-0.860294

Below is a table summarizing the descriptive statistics (describe()) of the normalized features. Compared to the raw feature statistics, the normalized feature clearly exhibits reduced variation, confirming that the differences across features have been scaled down effectively.

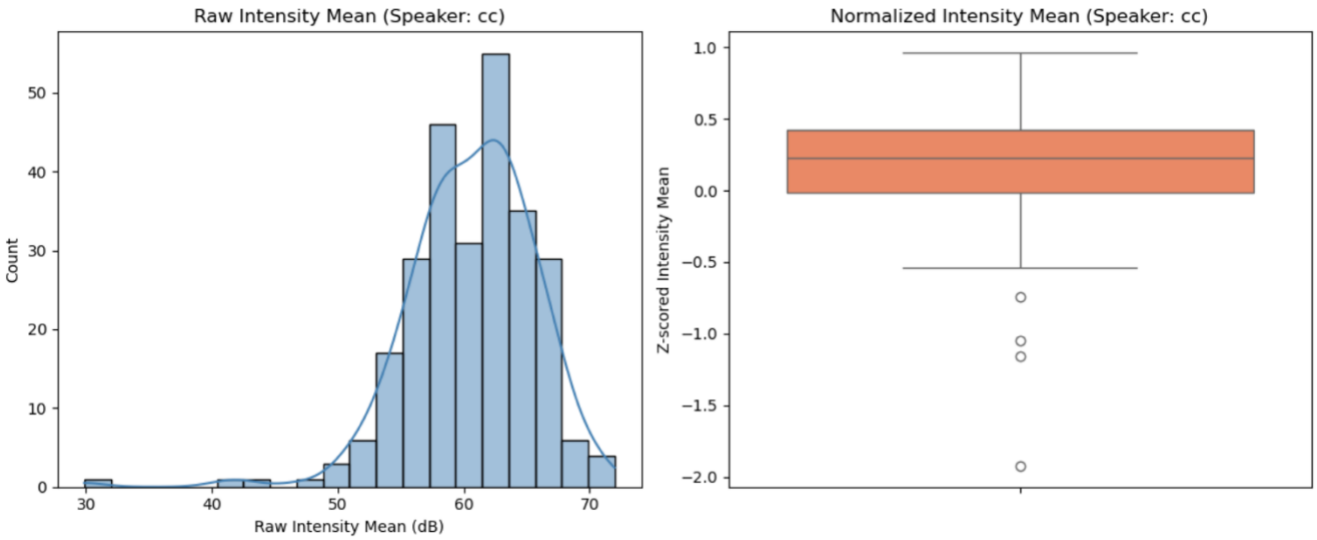
	pitch_min_norm	pitch_max_norm	pitch_mean_norm	intensity_min_norm	intensity_max_norm	intensity_mean_norm
count	2324.000000	2324.000000	2324.000000	2324.000000	2324.000000	2324.000000
mean	-0.834217	1.481870	-0.014563	-1.758494	1.310917	0.026545
std	0.538627	1.459062	0.713928	0.544625	0.492893	0.441493
min	-2.143777	-0.904523	-1.206272	-3.855171	-0.417897	-1.926496
25%	-1.109512	0.137936	-0.553339	-2.123198	1.002277	-0.245531
50%	-0.889233	1.303095	-0.206369	-1.687693	1.308812	0.034535
75%	-0.669941	2.736967	0.363948	-1.394999	1.647066	0.301637
max	2.176417	4.672103	2.678574	0.362883	2.877572	1.615998

After applying normalization, I visualized the speaker-wise distributions of mean pitch and mean intensity using box plots. In the raw feature distribution, mean pitch varied significantly across speakers. However, after normalization, the median pitch of almost every speaker aligned closely around zero. The interquartile range (25%-75%) became more consistent across speakers, and while some outliers remained, the distributions were regularized without major distortion. This demonstrates that z-score normalization effectively transformed speaker-specific pitch ranges into a shared scale. This adjustment allows the model to focus more on emotion-related variation rather than speaker-specific pitch characteristics. However, due to the presence of visible outliers, it may be beneficial to apply additional outlier removal after normalization when using this feature for classification. Similarly, although intensity showed less speaker-to-speaker variation compared to pitch in the raw features, mean intensity also becomes more balanced after normalization. The box plot of normalized intensity means shows that the median for each speaker is now close to zero, and the variation is significantly reduced. This suggests that while intensity is less speaker-sensitive than pitch, normalization still improved its consistency

across speakers.



That said, I noticed that the normalized intensity distribution for speaker “cc” was slightly shifted in the positive direction. To further investigate this, I visualized both the raw and normalized intensity distributions of speaker “cc” using a histogram and a box plot. As shown in the left-side histogram of raw intensity mean, most values are concentrated around 60 dB and follow an approximately normal distribution. However, the left-skewed tail indicates negative skewness. On the right-side box plot, the normalized intensity mean for speaker “cc” has its median near zero and an interquartile range around 0.5, showing that the scaling was applied properly. While a few outliers exist below -1.5, they are within the expected range and do not indicate a normalization failure. Therefore, I found that z-score normalization worked well even for speaker “cc”.



c. Plots of the mean and standard deviation of each feature for all of the 15 emotion classes, also report and discuss at least five interesting observations.

i. Emotion-Wise Discriminative Power: Overall, pitch features exhibit greater discriminative potential across emotions than intensity features. A feature is informative for classification when the mean differs notably across emotion classes while within-class variance remains small. In this regard, pitch_mean_norm stands out with large inter-emotion gaps and moderate error bars, indicating clear separability. Conversely, intensity_min_raw and intensity_mean_raw show minimal mean differences and relatively small variance across emotions, suggesting weak discriminative power.

ii. Raw vs Normalized Features: Features like pitch_mean, pitch_max, and intensity_max preserve class-wise separability even after normalization, making them robust for emotion classification. Although they show high inter-class differences in the raw form, the variance is also large. Z-score normalization effectively reduces within-class variability, yielding tighter distributions and better-defined decision boundaries. In contrast, intensity_min shows little to no separation both before and after normalization, indicating a weak feature for classification regardless of preprocessing. Interestingly, intensity_mean, initially uninformative, becomes more separable post-normalization as class-wise means diverge while error bars remain consistent.

iii. Elation shows enhanced separability after normalization: Among all emotions, elation benefits most from normalization. Its pitch and intensity values in the raw features are unremarkable; however, after normalization, pitch_mean_norm shows the highest mean for elation with relatively small error bars, indicating strong inter-class separation and improved within-class compactness. This suggests that normalization can reveal emotion-specific patterns that were obscured in the raw space.

iv. Emotions with ambiguous patterns: Cold-anger, boredom, and neutral consistently exhibit unclear or overlapping distributions. Cold-anger maintains mid-range values across all features with moderate error bars, making it difficult to distinguish. Boredom shows average pitch and intensity, again with little discriminative signal. Neutral, representing a baseline state, clusters around the center of most feature distributions with moderate spread, leading to frequent overlap with other emotions. These characteristics suggest lower classification accuracy due to fuzzy decision boundaries.

v. Different features highlight specific emotions: While some emotions exhibit overlapping patterns, some emotions display distinctive characteristics in particular features, providing helpful cues for classification. For example, panic shows a clearly elevated mean in pitch_max_norm. Although the error bar is large, the mean is distinctly higher than that of other emotions, likely reflecting the sudden high-pitch spikes typical of panic. Similarly, disgust is most prominent in intensity_max_norm, where it also shows a high mean and relatively large variance. Despite the variance, the distribution is sufficiently shifted upward to maintain separation from other emotions. On the other hand, in pitch_mean_norm, disgust falls closer to the lower-middle range with moderate variance, offering less discriminative power. These examples show that while not all acoustic features provide clear class separation, certain features can still serve as effective indicators for specific emotions.

2. Classification Experiments

a. Extract a set of acoustic-prosodic features using the openSMILE toolkit.

To use the openSMILE toolkit, I first created a 'feature' folder in the parent directory to store the extracted openSMILE features. Then, I cloned the openSMILE repository from GitHub and built openSMILE within the created 'opensmile' directory to enable execution. After successfully building openSMILE, I used the following command to extract acoustic-prosodic features from the .wav files in the 'hw3_speech_files' directory.

```
[ 98%] Building CXX object progsrc/smileapi/CMakeFiles/SMILEapi.dir/SMILEapi.cpp
.o
[ 98%] Building CXX object progsrc/smilextract/CMakeFiles/SMILExtract.dir/SMILEx
tract.cpp.o
[ 99%] Linking CXX executable SMILExtract
[ 99%] Built target SMILExtract
[100%] Linking CXX shared library libSMILEapi.dylib
[100%] Built target SMILEapi
(base)  eesun  ~/CODE/Advanced_Spoken_Language/HW3/opensmile  master
 find ../hw3_speech_files/ -type f -name "*.wav" -exec sh -c 'for file; do base
=$(basename "$file" ".wav"); ./build/progsrc/smilextract/SMILExtract -l 1 -C ./c
onfig/is09-13/IS09_emotion.conf -I $file -csvoutput "../features/${base}.csv"; d
one' sh {} +
```

After extraction, I confirmed that each resulting CSV file used a ; separator and contained 386 columns. Among these columns, 'name' and 'frameTime' were not actual acoustic-prosodic features (consisting only of "unknown" and 0.0 values, respectively), so I dropped these columns. Additionally, I extracted the speaker, emotion, and content information embedded in the filename and combined them with the openSMILE features, resulting in a single DataFrame with a shape of (2324, 387).

```
print(df.shape)
```

```
(2324, 390)
```

```
df.head()
```

	name	frameTime	pcm_RMSenergy_sma_max	pcm_RMSenergy_sma_min	pcm_RMSenergy_sma_range	p
0	'unknown'	0.0	0.010760	0.000064	0.010696	
1	'unknown'	0.0	0.076285	0.000034	0.076251	
2	'unknown'	0.0	0.020006	0.000010	0.019996	
3	'unknown'	0.0	0.015069	0.000043	0.015026	
4	'unknown'	0.0	0.016625	0.000111	0.016514	

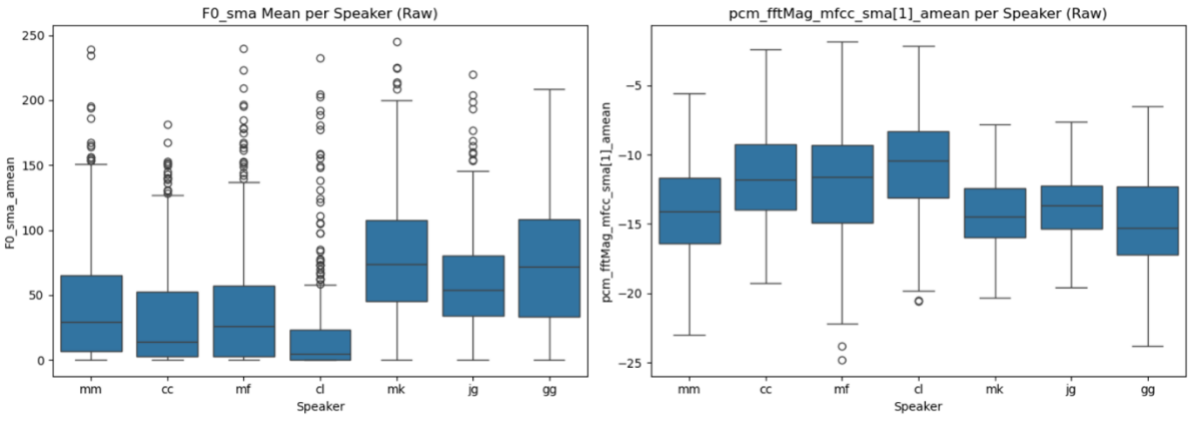
5 rows × 390 columns

In addition, I analyzed the structure of the openSMILE IS09_emotion feature set. The IS09_emotion configuration in openSMILE consists of 16 Low-Level Descriptors (LLDs), 2 types of smoothing/delta operations, and 12 types of statistical functionals. Thus, the total number of features can be calculated as: 16 x 2 x 12 = 384 features. The naming structure of the features is organized using underscores(_), separating the base descriptor, smoothing/delta type, and functional applied. A detailed breakdown of the feature organization is illustrated in the figures below.

	feature	smoothing	functionals
0	pcm_RMSenergy	sma	[max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis]
1	pcm_fftMag_mfcc	sma	[max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis, max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis, max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis, max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis, max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis, max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis, max, min, range, maxPos, ...]
2	pcm_zcr	sma	[max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis]
3	voiceProb	sma	[max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis]
4	F0	sma	[max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis]
5	pcm_RMSenergy	sma_de	[max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis]
6	pcm_fftMag_mfcc	sma_de	[max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis, max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis, max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis, max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis, max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis, max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis, max, min, range, maxPos, ...]
7	pcm_zcr	sma_de	[max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis]
8	voiceProb	sma_de	[max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis]
9	F0	sma_de	[max, min, range, maxPos, minPos, amean, linregc1, linregc2, linregerrQ, stddev, skewness, kurtosis]

b. Normalize your extracted features (as in Part 1. Feature Analysis)

For Part 2, I performed z-score normalization across individual speakers, following the same approach used in Part 1. However, since openSMILE extracts a large number of features, I first analyzed speaker-level distribution by plotting boxplots for the mean values of F0 and one selected MFCC channel for each speaker.



As shown in the figure, the distribution of F0 varies significantly across speakers, and the range (height) of the boxplots also differs noticeably among speakers. Although the MFCC coefficients do not show as extreme differences as F0, there are still noticeable speaker-dependent variations. This supports the fact that speech characteristics are inherently different across speakers, and without normalization, the model may end up learning speaker identity rather than emotional content. Thus, speaker-specific normalization is essential to reduce speaker bias.

Unlike the feature extraction process in Part 1 (using Praat and Parselmouth), the openSMILE features extracted here consist of 384 statistical functional values per utterance. Since these are aggregated statistics rather than raw feature arrays, it was not possible to perform normalization exactly the same way as in Part 1 (i.e., computing mean and standard deviation per frame and per speaker). Instead, I implemented a new normalization method, which is speaker-specific normalization. The normalization process is as follows: For each speaker, I computed the mean and standard deviation of each feature across their utterances, excluding zero and NaN values. Then, I applied z-score normalization using the formula: z = (x-mean)/std, where x is the feature value, and "mean" and "std" are the mean and standard deviation for the corresponding speaker.

Before performing normalization, I inspected the feature distributions to check for missing (NaN) and zero values. Although no NaN values were found across the features, a large number of features contained zeros, as shown in the figure below.

Columns with zero values:

pcm_RMSenergy_sma_maxPos	1
pcm_RMSenergy_sma_minPos	324
pcm_fftMag_mfcc_sma[1]_maxPos	17
pcm_fftMag_mfcc_sma[1]_minPos	24
pcm_fftMag_mfcc_sma[2]_maxPos	31
pcm_fftMag_mfcc_sma[2]_minPos	8
pcm_fftMag_mfcc_sma[3]_maxPos	28
pcm_fftMag_mfcc_sma[3]_minPos	6
pcm_fftMag_mfcc_sma[4]_maxPos	90
pcm_fftMag_mfcc_sma[4]_minPos	6
pcm_fftMag_mfcc_sma[5]_maxPos	109
pcm_fftMag_mfcc_sma[5]_minPos	4
pcm_fftMag_mfcc_sma[6]_maxPos	111
pcm_fftMag_mfcc_sma[7]_maxPos	79
pcm_fftMag_mfcc_sma[7]_minPos	2
pcm_fftMag_mfcc_sma[8]_maxPos	54
pcm_fftMag_mfcc_sma[8]_minPos	5
pcm_fftMag_mfcc_sma[9]_maxPos	71
pcm_fftMag_mfcc_sma[9]_minPos	3
pcm_fftMag_mfcc_sma[10]_maxPos	59
pcm_fftMag_mfcc_sma[10]_minPos	3
pcm_fftMag_mfcc_sma[11]_maxPos	64
pcm_fftMag_mfcc_sma[11]_minPos	11
pcm_fftMag_mfcc_sma[12]_maxPos	50
pcm_fftMag_mfcc_sma[12]_minPos	9
pcm_zcr_sma_min	159
pcm_zcr_sma_maxPos	16
pcm_zcr_sma_minPos	193
voiceProb_sma_maxPos	1
voiceProb_sma_minPos	134
F0_sma_max	363
F0_sma_min	2324
F0_sma_range	363
F0_sma_maxPos	363
F0_sma_minPos	2316
F0_sma_amean	363
F0_sma_linregc1	363
F0_sma_linregc2	363
F0_sma_linregerrQ	363
F0_sma_stddev	363
F0_sma_skewness	363
F0_sma_kurtosis	363
pcm_RMSenergy_sma_de_maxPos	3
pcm_fftMag_mfcc_sma_de[2]_minPos	1
F0_sma_de_max	363
F0_sma_de_min	364
F0_sma_de_range	363
F0_sma_de_maxPos	363
F0_sma_de_minPos	364
F0_sma_de_amean	436
F0_sma_de_linregc1	363
F0_sma_de_linregc2	363
F0_sma_de_linregerrQ	363
F0_sma_de_stddev	363
F0_sma_de_skewness	363
F0_sma_de_kurtosis	363

In particular, features like F0_sma_min and F0_sma_minPos consisted of more than 90% zeros across the 2324 wav files. Since these features are dominated by zero values, they likely carry very little useful information and could instead introduce noise during model training. Therefore, I decided to drop such features before performing speaker-wise normalization.

The steps for speaker-wise z-score normalization performed in Part 2 are summarized as follows:

i. For each speaker and each feature, compute the mean and standard deviation while excluding values that are either 0 or NaN.

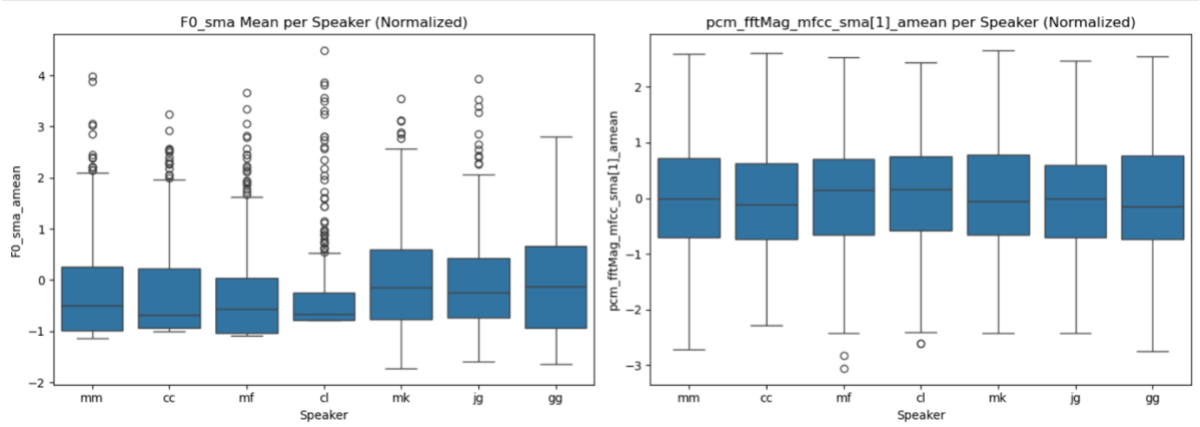
ii. Normalize each feature value using the formula: (x-mean)/std, where x is the feature value, and mean and std are calculated in step.

```
def normalize_feature_by_speaker(df, feature_col):
    normalized = []

    for speaker, group in df.groupby('speaker'):
        values = group[feature_col]

        # Exclude 0 and NaN to compute mean and std
        valid_values = values[(values != 0) & (~values.isna())]

        if len(valid_values) == 0:
            mean = 0
            std = 1
        else:
            mean = valid_values.mean()
            std = valid_values.std()

        # Normalize
        normalized_values = values.apply(
            lambda x: (x - mean) / std if (std != 0) else x
        )

        normalized.append(normalized_values)

    normalized_series = pd.concat(normalized)
    normalized_series = normalized_series.sort_index()  # restore original order
    return normalized_series
```

iii. Apply this normalization procedure to all features individually.

```
feature_columns = [col for col in df.columns if col not in ['speaker', 'emotion', 'content']]

normalized_df = df[['speaker', 'emotion', 'content']].copy()

normalized_features = []

for feature in feature_columns:
    normalized_feature = normalize_feature_by_speaker(df, feature)
    normalized_features.append(normalized_feature)

normalized_features_df = pd.concat(normalized_features, axis=1)
normalized_df = pd.concat([normalized_df, normalized_features_df], axis=1)
```

After applying z-score normalization across individual speakers, I visualized the speaker-level distributions using boxplots. The resulting boxplots show that: For F0, the differences in mean values across speakers have been largely eliminated. The median of each speaker is now approximately centered around 0, and the IQR (interquartile range) has become more consistent across speakers. Similarly, for MFCC features, the median values and overall distributions have become much more uniform across speakers. Thus, the speaker-wise normalization effectively reduced speaker variability, making the data more appropriate for emotion recognition rather than speaker identification.



c. Classification: Leave-one-speaker-out cross-validation to predict emotion classes.

Before performing classification, the class labels, which were initially in string format, needed to be encoded into numeric values. To achieve this, I used scikit-learn's LabelEncoder to encode the emotion labels into integers.

This task involves predicting 15 emotional states across seven different speakers, making it inherently complex and potentially more suitable for non-linear classification models rather than simple linear models. Therefore, to establish a baseline, I first trained plain, untuned models using Random Forest and Support Vector Machine (SVM) with a non-linear RBF kernel. In this setup, I used leave-one-speaker-out cross-validation: for each evaluation, one speaker was held out as the test set while the remaining speakers were used to train the model. Since I employed leave-one-speaker-out cross-validation, no separate train/validation/test split was necessary.

Without any hyperparameter tuning (i.e., using the plain default models), Random Forest achieved an aggregated accuracy of 0.2048 and an aggregated weighted F1 score of 0.1995. In comparison, SVM achieved an aggregated accuracy of 0.2590 and an aggregated weighted F1 score of 0.2546.

```
# baseline without using additional features as well as preprocessing

def leave_one_speaker_out_cv_v2(df, feature_columns, label_column='emotion', speaker_column='speaker'):
    speakers = df[speaker_column].unique()
    results = defaultdict(list)

    for speaker in speakers:
        # Train/Test split
        train_df = df[df[speaker_column] != speaker]
        test_df = df[df[speaker_column] == speaker]

        X_train = train_df[feature_columns]
        y_train = train_df[label_column]
        X_test = test_df[feature_columns]
        y_test = test_df[label_column]

        n_samples = len(y_test)
```

```
        # Model 1: Random Forest
        rf_model = RandomForestClassifier(random_state=42)
        rf_model.fit(X_train, y_train)
        y_pred_rf = rf_model.predict(X_test)
        acc_rf = accuracy_score(y_test, y_pred_rf)
        f1_rf = f1_score(y_test, y_pred_rf, average='weighted')

        # Model 2: SVM
        svm_model = SVC(kernel='rbf', random_state=42, class_weight="balanced")
        svm_model.fit(X_train, y_train)
        y_pred_svm = svm_model.predict(X_test)
        acc_svm = accuracy_score(y_test, y_pred_svm)
        f1_svm = f1_score(y_test, y_pred_svm, average='weighted')

        # Save results
        results['speaker'].append(speaker)
        results['n_samples'].append(n_samples)

        results['rf_accuracy'].append(acc_rf)
        results['rf_f1'].append(f1_rf)

        results['svm_accuracy'].append(acc_svm)
        results['svm_f1'].append(f1_svm)
```

```
        # Print classification reports
        print(f"\n==== Speaker {speaker} (Random Forest) ====")
        print(classification_report(y_test, y_pred_rf, zero_division=0))

        print(f"\n==== Speaker {speaker} (SVM) ====")
        print(classification_report(y_test, y_pred_svm, zero_division=0))

    # results
    results_df = pd.DataFrame(results)

    # Aggregated metrics
    total_samples = results_df['n_samples'].sum()

    aggregated_rf_accuracy = np.sum(results_df['rf_accuracy'] * results_df['n_samples']) / total_samples
    aggregated_rf_f1 = np.sum(results_df['rf_f1'] * results_df['n_samples']) / total_samples

    aggregated_svm_accuracy = np.sum(results_df['svm_accuracy'] * results_df['n_samples']) / total_samples
    aggregated_svm_f1 = np.sum(results_df['svm_f1'] * results_df['n_samples']) / total_samples

    # Print final results
    print("\n--- Aggregated Results ---")
    print(f"Random Forest Aggregated Accuracy: {aggregated_rf_accuracy:.4f}")
    print(f"Random Forest Aggregated Weighted F1: {aggregated_rf_f1:.4f}")

    print(f"SVM Aggregated Accuracy: {aggregated_svm_accuracy:.4f}")
    print(f"SVM Aggregated Weighted F1: {aggregated_svm_f1:.4f}")

    return results_df
```

To improve the aggregated accuracy and aggregated weighted F1 score of the baseline model, I conducted the following experiments:

i. Outlier Handling

From the earlier boxplot analysis, it was clear that several outliers existed, especially for F0 values (notably for the speaker class “cl”). To further investigate the presence of outliers for each feature, I analyzed values with an absolute z-score greater than 3, as shown in the figure below. Although the outlier proportion for the most affected features was only around 3-4%, I judged that handling outliers was necessary because outliers can distort the mean and standard deviation.

Normally, outlier removal is performed before normalization. However, since the features extracted in Part 2 were not row feature arrays but statistical functionals, and the main objective was to align speaker scales

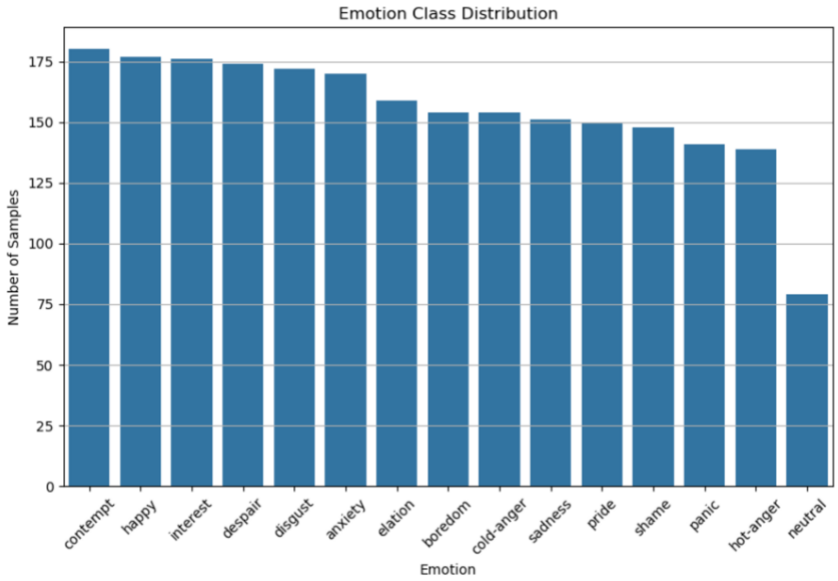
through normalization, I first performed normalization and then handled outliers afterward. Considering the limited data size (2324 utterances across 15 emotion classes), simply removing outliers would risk losing valuable data. Thus, instead of deletion, I applied clipping: I set a z-score threshold of 3, and any value beyond this threshold was clipped to +3 or -3 accordingly, preserving the data while mitigating the outlier effect.

	outlier_count	total_count	outlier_ratio (%)
F0_sma_de_range	89.0	2324.0	3.83
F0_sma_de_min	89.0	2324.0	3.83
pcm_RMSenergy_sma_min	53.0	2324.0	2.28
pcm_RMSenergy_sma_linregc2	50.0	2324.0	2.15
pcm_zcr_sma_de_amean	49.0	2324.0	2.11
...	...	...	...
voiceProb_sma_stddev	2.0	2324.0	0.09
pcm_fftMag_mfcc_sma[3]_max	2.0	2324.0	0.09
pcm_fftMag_mfcc_sma_de[1]_max	2.0	2324.0	0.09
pcm_fftMag_mfcc_sma[1]_amean	1.0	2324.0	0.04
pcm_zcr_sma_stddev	1.0	2324.0	0.04

```python
# clipping outlier
def clip_outliers_after_normalization(df, feature_columns, threshold=3):
    df_clipped = df.copy()

    df_clipped[feature_columns] = df_clipped[feature_columns].clip(lower=-threshold, upper=threshold)

    return df_clipped
```

ii. Class imbalance

Class imbalance can cause models to become biased toward majority classes, leading to poor performance on minority classes and degraded weighted F1 scores. To analyze this issue, I visualized the class distribution using a count plot.



The analysis revealed that the most frequent class, contempt, had 180 samples, while the least frequent class, neutral, had only 79 samples. Most other classes had approximately 140-180 samples, suggesting moderate balance except for neutral. Therefore, during leave-one-speaker-out cross-validation, I applied SMOTE oversampling to balance the training set for each speaker split.

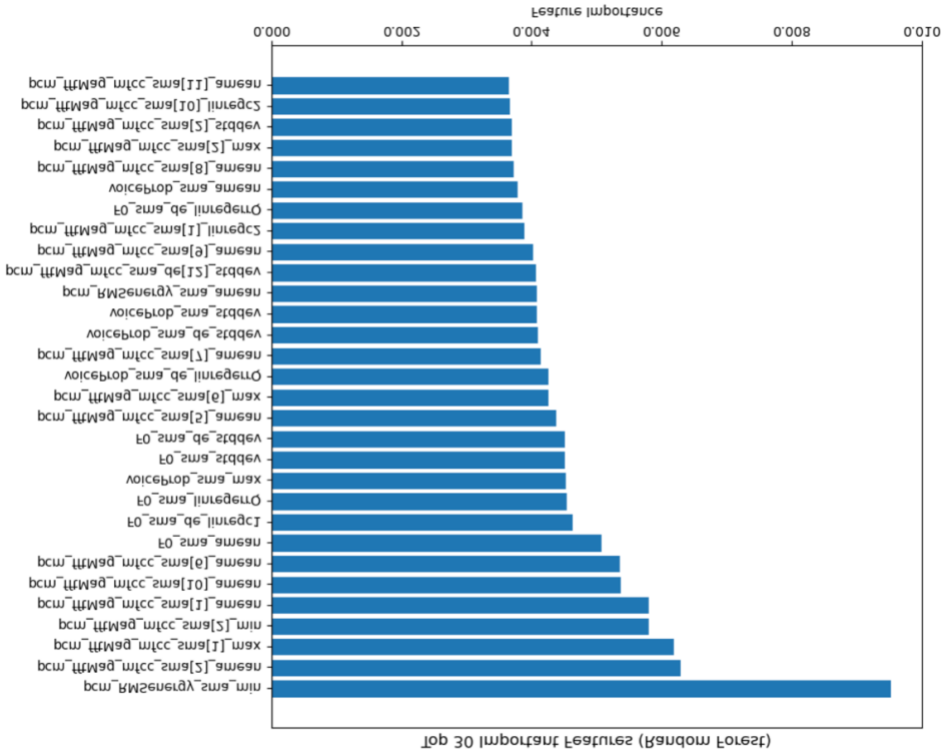
```python
def leave_one_speaker_out_cv_v2_smote(df, feature_columns, label_column='emotion', speaker_column
                                      target_size_per_class=None, smote_k_neighbors=5):
    speakers = df[speaker_column].unique()
    results = defaultdict(list)

    for speaker in speakers:
        # Train/Test split
        train_df = df[df[speaker_column] != speaker]
        test_df = df[df[speaker_column] == speaker]

        X_train = train_df[feature_columns]
        y_train = train_df[label_column]
        X_test = test_df[feature_columns]
        y_test = test_df[label_column]

        n_samples = len(y_test)

        # ===== SMOTE oversampling (Train set only) =====
        smote = SMOTE(k_neighbors=smote_k_neighbors, random_state=42)
        X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
        # ===== End oversampling =====

        # Model: SVM
        svm_model = SVC(kernel='rbf', random_state=42, class_weight="balanced")
        svm_model.fit(X_train_resampled, y_train_resampled)
        y_pred = svm_model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average='weighted')

        # Save results
        results['speaker'].append(speaker)
        results['n_samples'].append(n_samples)
        results['accuracy'].append(acc)
        results['f1'].append(f1)
```

```python
    # results
    results_df = pd.DataFrame(results)
    total_samples = results_df['n_samples'].sum()

    aggregated_accuracy = (results_df['accuracy'] * results_df['n_samples']).sum() / total_samples
    aggregated_f1 = (results_df['f1'] * results_df['n_samples']).sum() / total_samples

    # Print final results
    print("\n--- Aggregated Results ---")
    print(f"SVM Aggregated Accuracy: {aggregated_accuracy:.4f}")
    print(f"SVM Aggregated Weighted F1: {aggregated_f1:.4f}")

    return aggregated_accuracy, aggregated_f1
```

iii. Feature Selection

Since the dataset originally contained 384 openSMILE features, which represent a high-dimensional space relative to the small dataset size, it was highly susceptible to overfitting and multicollinearity. Therefore, selecting the most important features was essential.

To perform feature selection, I used a random forest model to rank the feature importance and extracted the top-k features accordingly. I first visualized the feature selection results for the top 30 features and experimented by varying the number of selected features, k.



The top 220 features among the 384 openSMILE features yielded the best model performance. After applying outlier handling (clipping) and class balancing (oversampling), the model achieved an aggregated average accuracy of 0.2840 and an aggregated weighted F1 score of 0.2788. To further improve the model's performance, I performed further experiments using both top_features_220 and top_features_200.

iv. Extract Additional Features

To capture emotions from short utterances, I extracted additional features representing prosodic dynamics (pitch and intensity variation), voice quality (jitter, shimmer, and HNR), and articulation clarity (ASR confidence score), along with two text-based features derived from the "content". In total, I extracted eight speech-based features and two text-based features, resulting in ten additional features overall.

Regarding speech-based additional features, based on the error bar analysis from Task 1, I had already confirmed that pitch mean, pitch max, pitch min, and intensity mean were effective for emotion classification. Therefore, I selected these four features (already normalized using speaker-wise z-score normalization in Task 1) into the "merged_df" dataset.

Beyond pitch and intensity, additional features were extracted as follows: Jitter and Shimmer capture instability in pitch and energy, respectively, which are often associated with emotions such as anxiety, nervousness, and sadness. HNR measures the amount of noise in the voice, which can vary depending on emotional states like anger, fatigue, or sadness. ASR confidence score reflects how clearly a spoken utterance can be transcribed to text, serving as an indicator of articulation clarity, which can reveal nervousness or a lack of

confidence.

Using the Praat softwre and the Parselmouth library (as used in HW1), I extracted the jitter, shimmer, and mean values of HNR. For the ASR confidence score, I used speech_recognition library to perform speech-to-text conversion with the Google Speech Recognition API and extracted the confidence score. After extraction, I observed that approximately 5% of the ASR confidence scores were missing (NaN). Given the short utterance nature of the dataset, I filled the missing values by replacing them with the overall mean.

```
jitter: 0 / 2324 (0.00% missing)
shimmer: 0 / 2324 (0.00% missing)
hnr: 0 / 2324 (0.00% missing)
asr_confidence: 118 / 2324 (5.08% missing)
```

Additionally, I considered extracting simple text-based features. The utterances in this dataset, lasting only 1-2 seconds, consist mainly of non-semantic content (e.g., “three-hundred-nine”, “one-thousand-three”), meaning that TF-IDF or semantic embeddings would not be appropriate. Instead, I focused on analyzing differences in reading style, such as utterance length or number of words. Specifically, I compute a content ratio by dividing the word_count by the content_length to reflect utterance compactness. Also, I created an interaction feature, pitch variation, calculated as pitch_mean_norm x content_length, to capture how pitch dynamics vary depending on the length of the utterance.

Regarding normalization, the pitch and intensity features from Task 1 were already normalized. Therefore, I normalized jitter, shimmer, HNR, ASR confidence, and pitch variation by using speaker-wise z-score normalization as applied in Part 2. Descriptive statistics prior to normalization indicated that HNR and pitch variation especially exhibited wide value ranges, justifying normalization.

pitch_min_norm	pitch_max_norm	pitch_mean_norm	intensity_mean_norm	jitter	shimmer	hnr	asr_confidence	content_ratio	pitch_variation
2324.000000	2324.000000	2324.000000	2324.000000	2324.000000	2324.000000	2324.000000	2324.000000	2324.000000	2324.000000
-0.834217	1.481870	-0.014563	0.026545	0.022854	0.109806	10.000678	0.660147	0.163908	-0.063299
0.538627	1.459062	0.713928	0.441493	0.009909	0.031856	3.025813	0.284066	0.021211	1.927972
-2.143777	-0.904523	-1.206272	-1.926496	0.005929	0.030318	1.187835	0.001717	0.117647	-3.618815
-1.109512	0.137936	-0.553339	-0.245531	0.015499	0.086730	7.938944	0.433150	0.150000	-1.470704
-0.889233	1.303095	-0.206369	0.034535	0.020734	0.106186	9.835073	0.695405	0.166667	-0.534015
-0.669941	2.736967	0.363948	0.301637	0.028314	0.128475	11.994265	0.931372	0.176471	0.899645
2.176417	4.672103	2.678574	1.615998	0.085474	0.265438	22.125217	0.998513	0.250000	8.991353

After merging these additional features into the dataset, I applied outlier handling through clipping (thresholding z-scores beyond +-3) and conducted oversampling to address class imbalance issues. Finally, I retrained the SVM model using this augmented feature set and evaluated its performance based on the aggregated accuracy and aggregated weighted F1 score.

v. Model Selection and Hyper-parameter tuning

To further validate model choice, I also experimented with other classifiers, including Logistic Regression (a linear model), and ensemble models such as XGBoost and LightGBM. Logistic Regression achieved an aggregated accuracy of 0.2500 and an aggregated weighted F1 score of 0.2501, indicating the simple linear model could struggle with capturing each speaker's patterns. XGBoost achieved an aggregated accuracy of 0.2612 and an aggregated weighted F1 score of 0.2603, similar to that of SVM. LightGBM showed slightly better performance than XGBoost, with an aggregated accuracy of 0.2676 and an aggregated weighted F1 score of 0.2651. However, during LightGBM training, warning messages such as "No further splits with positive gain" were observed, suggesting overfitting risk or convergence issues due to small data size or feature distribution. Given this instability, I determined that LightGBM was not reliable for further feature engineering experiments. As a result, I selected SVM as the base classifier for downstream modeling, as it offered consistently strong and stable performance across speakers and is well-suited for high-dimensional, small-sample scenarios.

```
models = {
    "Logistic Regression": LogisticRegression(solver='lbfgs', max_iter=1000, random_state=42),
    "XGBoost": XGBClassifier(n_estimators=100, max_depth=4, learning_rate=0.1, subsample=0.8, colsample_bytree=0.8,
    "RandomForest": RandomForestClassifier(n_estimators=200, max_depth=20, random_state=42, n_jobs=-1),
    "LightGBM": LGBMClassifier(num_leaves=31, learning_rate=0.05, n_estimators=300, random_state=42)
}

feature_columns = merged_df.columns.difference(['filename', 'speaker', 'emotion', 'content']).tolist()

results = {}
for model_name, model in models.items():
    print(f"\n===== Training {model_name} =====")
    acc, f1 = leave_one_speaker_out_cv_v2_model(clipped_merged_df, feature_columns, model, model_name)
    results[model_name] = (acc, f1)
```

After selecting SVM, I conducted additional fine-tuning experiments by manually adjusting the gamma value (indicating the default "scale" heuristic and fixed values such as 0.01, 0.1, etc.) and sweeping different C values. The hyperparameter C controls the regularization strength; larger C values reduce regularization and may lead to overfitting, while smaller values enforce more margin. The gamma parameter defines the influence range of a single training example in the RBF kernel; lower gamma implies a wider influence and smoother decision boundaries, while higher gamma focuses more narrowly on each point. Despite these tuning efforts, the default configuration (C=1.0, gamma='scale', kernel='rbf') consistently yielded the best performance. Therefore, I finalized the use of SVM with its default setting for subsequent experiments and feature engineering.

```python
def calculate_gamma_scale(X):
    n_features = X.shape[1]
    variance = np.var(X)
    gamma = 1.0 / (n_features * variance)
    return gamma
```

```python
feature_columns = merged_df.columns.difference(['filename', 'speaker', 'emotion', 'content'])
X = merged_df[feature_columns].values
gamma_scale = calculate_gamma_scale(X)

print(f"Calculated gamma (scale): {gamma_scale:}")
```

```
Calculated gamma (scale): 0.004629294535340114
```

```python
c_list = [1, 10, 100]
gamma_list = [gamma_scale * 0.5, gamma_scale, gamma_scale * 2.0, 0.1, 0.01, 0.005, 0.001, 0.003]

best_f1 = 0
best_params = (None, None)

for c in c_list:
    for gamma in gamma_list:
        leave_one_speaker_out_cv_with_params(merged_df, feature_columns, C=c, gamma=gamma, kernel="rbf")

        print(f"C={c}, gamma={gamma:.6f} => Aggregated Accuracy: {aggregated_accuracy:.4f}, Aggregated F1: {aggrega

        if aggregated_f1 > best_f1:
            best_f1 = aggregated_f1
            best_params = (c, gamma)

print(f"\nBest SVM Params: C={best_params[0]}, gamma={best_params[1]:.6f} (Aggregated F1: {best_f1:.4f})")
```

d. Results of the Best Model: (1) Classification results, (2) Aggregated average accuracy, (3) Aggregated weighted F1 scores.

Following the experimental procedures described in Section (c), including outlier handling, class imbalance correction, feature selection using Random Forest, extraction of additional text and speech features, and SVM hyperparameter tuning, I aimed to find the best-performing model.

At first, I selected 200 features from openSMILE and added 10 additional features (eight speech-based and two text-based) to form a combined feature set of 210 features. Among these, feature selection was performed again using a Random Forest model based on feature importance. By varying the number of selected features (k) and evaluating the aggregated weight F1 scores, I determined that selecting 180 features produced the best overall performance. All features were pre-processed using speaker-wise z-score normalization to eliminate speaker-specific biases. Afterward, I experimented with applying outlier handling (clipping) and oversampling (using SMOTE) separately and together. The experimental results revealed that applying normalization plus outlier clipping, without oversampling, achieved the best performance, outperforming other combinations that included oversampling.

```
com2 # 180 0.3003, 0.2977
```

	k	Aggregated Accuracy	Aggregated F1
0	50	0.266781	0.262909
1	100	0.276678	0.271831
2	150	0.291738	0.286739
3	160	0.285714	0.281797
4	170	0.291738	0.287513
5	180	0.300344	0.297668
6	190	0.289587	0.286815
7	200	0.299053	0.294689

```
com2
```

	k	Aggregated Accuracy	Aggregated F1
0	175	0.291738	0.287952
1	180	0.300344	0.297668
2	185	0.296472	0.293024

```
feature k: 180

merged + clipping + feature 180:: acc 0.3003, f1 0.2976
```

For model selection, I used the Support Vector Machine (SVM) classifier. Through hyperparameter tuning, I explored various values of C, gamma, and kernel types. The best configuration was found to be C = 1.0 (default), gamma = “scale” (default), and kernel = “rbf” (default). The choice of an RBF kernel was particularly appropriate because the task involved classifying 15 emotional classes across seven different speakers based on short utterances. This setting naturally leads to non-linear decision boundaries, which RBF kernels handle better than linear kernels. Additionally, the “scale” gamma value adapts dynamically to the feature variance, making it effective for high-dimensional datasets with normalized features.

Using the 180 selected and normalized features and the best-performing SVM configuration, the final model achieved an **aggregated average accuracy of 0.3003** and an **aggregated weighted F1 score of 0.2977**. Finally, the detailed classification reports for each speaker are provided below:

```
==== Speaker mm (SVM) ====
              precision    recall  f1-score   support

     anxiety       0.57      0.31      0.40        39
     boredom       0.47      0.42      0.44        19
  cold-anger       0.16      0.15      0.15        20
    contempt       0.33      0.21      0.26        19
     despair       0.17      0.22      0.20        18
     disgust       0.32      0.26      0.29        23
     elation       0.27      0.32      0.29        19
       happy       0.39      0.72      0.51        18
   hot-anger       0.60      0.38      0.46        16
    interest       0.25      0.33      0.29        21
     neutral       1.00      0.22      0.36         9
       panic       0.75      0.43      0.55        28
       pride       0.41      0.47      0.44        19
     sadness       0.18      0.24      0.21        17
       shame       0.36      0.76      0.49        17

    accuracy                           0.36       302
   macro avg       0.42      0.36      0.36       302
weighted avg       0.41      0.36      0.36       302
```

```
==== Speaker cc (SVM) ====
              precision    recall  f1-score   support

     anxiety       0.05      0.10      0.07        10
     boredom       0.06      0.13      0.08        15
  cold-anger       0.08      0.07      0.07        15
    contempt       0.34      0.45      0.39        22
     despair       0.07      0.11      0.09         9
     disgust       0.42      0.26      0.32        31
     elation       0.21      0.25      0.23        16
       happy       0.37      0.30      0.33        23
   hot-anger       0.38      0.71      0.50        14
    interest       0.21      0.18      0.19        17
     neutral       0.75      0.33      0.46        18
       panic       0.60      0.50      0.55        18
       pride       0.50      0.22      0.30        23
     sadness       0.15      0.15      0.15        13
       shame       0.29      0.19      0.23        21

    accuracy                           0.28       265
   macro avg       0.30      0.26      0.26       265
weighted avg       0.33      0.28      0.29       265
```

```
==== Speaker mf (SVM) ====
              precision    recall  f1-score   support

     anxiety       0.36      0.45      0.40        22
     boredom       0.32      0.26      0.29        27
  cold-anger       0.08      0.10      0.09        20
    contempt       0.61      0.39      0.47        44
     despair       0.16      0.19      0.17        16
     disgust       0.04      1.00      0.07         1
     elation       0.05      0.04      0.04        26
       happy       0.13      0.09      0.11        23
   hot-anger       0.47      0.43      0.45        21
    interest       0.08      0.05      0.06        19
     neutral       0.67      1.00      0.80        10
       panic       0.37      0.58      0.45        12
      pride        0.06      0.06      0.06        18
     sadness       0.11      0.10      0.11        20
       shame       0.36      0.25      0.29        20

    accuracy                           0.26       299
   macro avg       0.26      0.33      0.26       299
weighted avg       0.28      0.26      0.26       299
```

```
==== Speaker cl (SVM) ====
              precision    recall  f1-score   support

     anxiety       0.23      0.33      0.27        21
     boredom       0.42      0.79      0.55        29
  cold-anger       0.69      0.41      0.51        27
    contempt       0.26      0.24      0.25        25
     despair       0.33      0.21      0.26        29
     disgust       0.27      0.18      0.22        22
     elation       0.27      0.30      0.28        27
       happy       0.33      0.24      0.28        21
   hot-anger       0.63      0.65      0.64        26
    interest       0.33      0.27      0.30        26
     neutral       1.00      0.06      0.11        17
       panic       0.24      0.19      0.21        21
      pride        0.40      0.33      0.36        24
     sadness       0.16      0.15      0.15        27
       shame       0.22      0.46      0.30        26

    accuracy                           0.33       368
   macro avg       0.39      0.32      0.31       368
weighted avg       0.38      0.33      0.32       368
```

```
==== Speaker mk (SVM) ====
              precision    recall  f1-score   support

     anxiety       0.06      0.07      0.06        29
     boredom       0.07      0.05      0.06        20
  cold-anger       0.20      0.35      0.25        23
    contempt       0.22      0.19      0.21        21
     despair       0.35      0.17      0.23        53
     disgust       0.27      0.19      0.22        21
     elation       0.27      0.52      0.36        23
       happy       0.27      0.24      0.25        42
   hot-anger       0.44      0.32      0.37        22
    interest       0.34      0.27      0.30        44
     neutral       0.78      0.88      0.82         8
       panic       0.50      0.57      0.53        21
      pride        0.15      0.17      0.16        23
     sadness       0.14      0.18      0.16        22
       shame       0.32      0.36      0.34        25

    accuracy                           0.26       397
   macro avg       0.29      0.30      0.29       397
weighted avg       0.28      0.26      0.26       397
```

```
==== Speaker jg (SVM) ====
              precision    recall  f1-score   support

     anxiety       0.20      0.16      0.18        19
     boredom       0.36      0.36      0.36        14
  cold-anger       0.22      0.18      0.20        22
    contempt       0.24      0.17      0.20        23
     despair       0.18      0.24      0.20        21
     disgust       0.35      0.30      0.33        23
     elation       0.23      0.15      0.18        20
       happy       0.16      0.15      0.15        20
   hot-anger       0.24      0.33      0.28        18
    interest       0.19      0.26      0.22        19
     neutral       0.00      0.00      0.00         8
       panic       0.11      0.07      0.09        14
      pride        0.09      0.17      0.11        18
     sadness       0.32      0.32      0.32        19
       shame       0.08      0.07      0.07        15

    accuracy                           0.21       273
   macro avg       0.20      0.20      0.19       273
weighted avg       0.21      0.21      0.20       273
```

```
==== Speaker gg (SVM) ====
              precision    recall  f1-score   support

     anxiety       0.45      0.50      0.48        30
     boredom       0.34      0.40      0.37        30
  cold-anger       0.22      0.37      0.28        27
    contempt       0.35      0.35      0.35        26
     despair       0.19      0.14      0.16        28
     disgust       0.62      0.31      0.42        51
     elation       0.66      0.75      0.70        28
       happy       0.28      0.57      0.37        30
   hot-anger       0.74      0.77      0.76        22
    interest       0.21      0.23      0.22        30
     neutral       1.00      0.11      0.20         9
       panic       0.78      0.52      0.62        27
      pride        0.16      0.16      0.16        25
     sadness       0.00      0.00      0.00        33
       shame       0.26      0.29      0.27        24

    accuracy                           0.37       420
   macro avg       0.42      0.37      0.36       420
weighted avg       0.39      0.37      0.36       420
```

```
--- Aggregated Results ---
SVM Aggregated Accuracy: 0.3003
SVM Aggregated Weighted F1: 0.2977
```

```
result_df
```

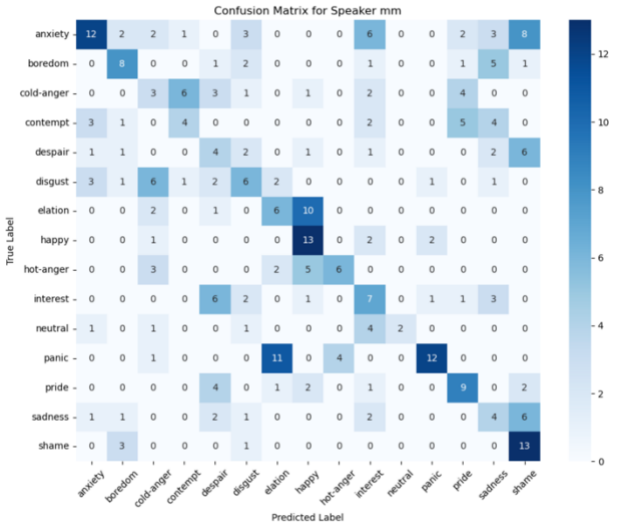
	speaker	n_samples	accuracy	f1
0	mm	302	0.360927	0.360742
1	cc	265	0.275472	0.286293
2	mf	299	0.260870	0.262450
3	cl	368	0.334239	0.322287
4	mk	397	0.264484	0.260877
5	jg	273	0.205128	0.202976
6	gg	420	0.366667	0.359321

Speakers “mm” and “gg” achieved the highest performance, with speaker “mm” reaching an accuracy of 0.3609 and a weighted F1 score of 0.3607, and speaker “gg” achieving an accuracy of 0.3667 and a weighted F1 score of 0.3593. In contrast, speaker “jg” exhibited the lowest performance, with an accuracy of 0.2051 and a weighted F1 score of 0.2030. These results indicate that performance varies significantly across speakers, likely due to individual differences such as speaking styles, pronunciation clarity, or recording conditions.

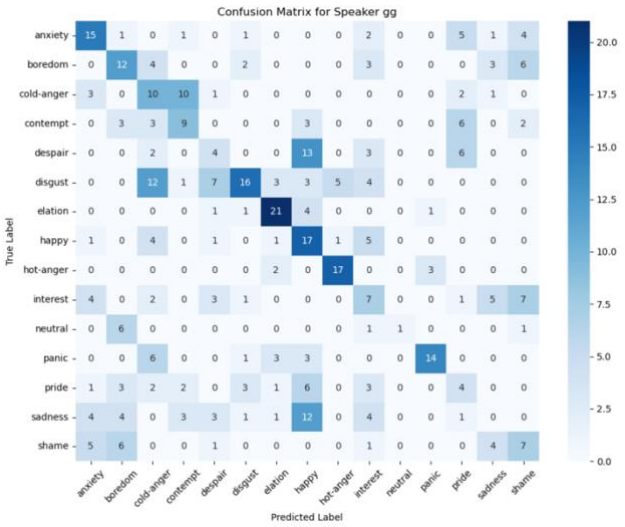
3. Error Analysis

To analyze the best results obtained for one of the seven speakers, the speakers “mm” and “gg” achieved the highest performance with an accuracy of approximately 0.36 and a weighted F1 score of around 0.36. The detailed classification reports and confusion matrices of speakers “mm” and “gg” are shown below:

```
==== Speaker mm (SVM) ====
              precision    recall  f1-score   support

     anxiety       0.57      0.31      0.40        39
     boredom       0.47      0.42      0.44        19
  cold-anger       0.16      0.15      0.15        20
    contempt       0.33      0.21      0.26        19
     despair       0.17      0.22      0.20        18
     disgust       0.32      0.26      0.29        23
     elation       0.27      0.32      0.29        19
       happy       0.39      0.72      0.51        18
   hot-anger       0.60      0.38      0.46        16
    interest       0.25      0.33      0.29        21
     neutral       1.00      0.22      0.36         9
       panic       0.75      0.43      0.55        28
       pride       0.41      0.47      0.44        19
     sadness       0.18      0.24      0.21        17
       shame       0.36      0.76      0.49        17

    accuracy                           0.36       302
   macro avg       0.42      0.36      0.36       302
weighted avg       0.41      0.36      0.36       302
```




```
==== Speaker gg (SVM) ====
              precision    recall  f1-score   support

     anxiety       0.45      0.50      0.48        30
     boredom       0.34      0.40      0.37        30
  cold-anger       0.22      0.37      0.28        27
    contempt       0.35      0.35      0.35        26
     despair       0.19      0.14      0.16        28
     disgust       0.62      0.31      0.42        51
     elation       0.66      0.75      0.70        28
       happy       0.28      0.57      0.37        30
   hot-anger       0.74      0.77      0.76        22
    interest       0.21      0.23      0.22        30
     neutral       1.00      0.11      0.20         9
       panic       0.78      0.52      0.62        27
       pride       0.16      0.16      0.16        25
     sadness       0.00      0.00      0.00        33
       shame       0.26      0.29      0.27        24

    accuracy                           0.37       420
   macro avg       0.42      0.37      0.36       420
weighted avg       0.39      0.37      0.36       420
```



a. What do you observe from the results you got for this speaker overall?

For speaker “mm”, the accuracy and weighted F1 score were both approximately 0.36. While emotions such as neutral and hot-anger showed high precision (1.00 and 0.60, respectively), their recall rates remained relatively low at 0.22 and 0.38. The shame class demonstrated an exceptionally high recall of 0.76, although its precision was moderate at 0.36. Notably, neutral exhibited an exceptionally high precision despite its small support, but it suffered from a very low recall, suggesting that the model was able to correctly identify neutral instances only when it was very confident, but failed to detect most actual neutral samples. According to the confusion matrix, classes such as panic, happy, and shame were relatively well classified, whereas cold-anger, despair, and contempt were often confused with other emotions. From the perspective of precision, recall, and support, the neutral class had a precision of 1.00 but a very small support of 9. Thus, despite the perfect precision, its impact on the overall performance, particularly on the weighted F1 scores, was minimal. Furthermore, the low recall of neutral indicates that the model struggled to recognize most of the neutral instances. In contrast, the panic class, with a precision of 0.75, a recall of 0.43, and a support of 28, contributed substantially to improving the weighted average precision, owing to its relatively large number of samples and strong precision. Similarly, the anxiety, happy, and hot-anger classes also had relatively large support and reasonably high precision and recall, thereby positively impacting the weighted average performance. Conversely, cold-anger, despair, and sadness had moderate support (18-20) but exhibited extremely low precision and recall, leading to degradation in both the macro and weighted averages. Overall, while classes like panic and hot-anger helped maintain a reasonable level of precision, the generally low recall across classes significantly limited the overall performance.

For speaker “gg”, the accuracy and weighted F1 score were approximately 0.37 and 0.36, respectively. Emotions such as hot-anger, panic, and elation showed relatively high precision and recall. In particular, hot-anger achieved a precision of 0.74 and a recall of 0.77, while elation achieved 0.66 and 0.75, respectively, making them the most stable classes. However, the sadness class showed both precision and recall of 0.00, meaning it was not predicted at all. Given that sadness had a substantial support of 33, this failure critically impacted the weighted average performance. The confusion matrix further revealed that predictions were mainly concentrated in elation, hot-anger, and happy, while misclassifications were frequent for sadness, despair, and neutral. From the perspective of precision, recall, and support, both hot-anger and elation had relatively large support and strong performance in both precision and recall, serving as key contributors to boosting the weighted averages. In contrast, although the neutral class had a small support of 9 and achieved a precision of 1.00, its very low recall (0.11) negatively impacted the macro recall. Thus, the strong performances in hot-anger and elation helped maintain the overall averages, but difficulties in predicting emotions such as sadness and neutral severely impaired the performance.

Both speakers achieved similar accuracy and weighted F1 scores and exhibited a mixture of well-predicted

and confused classes. Specifically, speaker "mm" demonstrated moderate F1 scores across many classes, where classes with relatively high support, such as happy and panic, positively contributed to the overall performance despite inconsistencies across other classes. In contrast, speaker "gg" achieved very strong precision and recall for specific classes like elation and hot-anger, which also had substantial support, helping to sustain the weighted average. However, the complete failure to predict high-support classes such as sadness (precision and recall of 0.00) critically impaired the overall performance for speaker "gg". Both speakers exhibited low recall for neutral; however, due to its small support, the impact on weighted performance was minor. Nevertheless, both sadness and despair were consistently difficult to predict, suggesting that class imbalance and emotional similarity between negative emotions may have negatively influenced the model training and overall performance.

b. And, in more detailed observations, which class(es) were **easiest** to predict? Why do you think they were easy?

For speaker "mm", the easiest classes to predict were happy, panic, and shame. Specifically, happy class has a high recall of 0.72 with relatively stable predictions (13 correct out of 18). This indicates that happy expressions tend to show highly consistent prosodic features, including a higher and more stable pitch, smoother intensity contours, and rhythmic speech patterns. These characteristics would create a strong acoustic signature that makes happy utterances easily distinguishable from other emotions, thereby facilitating more reliable classification by the model. Panic showed a high precision of 0.75 and a moderate recall of 0.43. However, according to the confusion matrix, some confusion occurred where panic instances were misclassified as elation. This suggests that panic and elation can share certain energetic prosodic patterns, such as a high pitch range, rapid pitch variations, and elevated speech intensity. While the energy of panic is often more chaotic or abrupt compared to the structured excitement of elation, the shared high-energy profile may have led the model to occasionally confuse these two emotions. Shame exhibited a very high recall of 0.76 with reasonable classification accuracy (13 out of 17). Shame expressions are often characterized by distinctively lower energy, slower speech tempo, and reduced pitch variability, making them acoustically separable from more dynamic emotions. This relatively consistent and subdued prosodic pattern likely contributed to the model's ability to detect shame reliably, despite moderate precision.

For speaker "gg", the easiest classes to predict were elation and hot-anger. Specifically, elation achieved a high precision of 0.66 and recall of 0.75, with 21 correct predictions. Elation is typically associated with dynamic and highly energetic speech characteristics, including rising pitch trajectories, broad pitch range, and frequent modulation of intensity. These vibrant acoustic patterns form a distinct emotional signature, allowing the model to differentiate elation from other emotions with high confidence. Hot-anger also exhibited very strong performance, achieving a precision of 0.74 and a recall of 0.77. Hot-anger is characterized by abrupt, forceful prosodic changes,

such as sharp increases in volume and pitch, rapid articulation, and an overall harsh vocal tone. These intense and aggressive prosodic cues would provide strong indicators for the model to detect hot-anger instances accurately, minimizing confusion with other emotions.

Comparing the two speakers, it is evident that the easiest classes differed. For speaker “mm”, happy, panic, and shame were easiest to predict, whereas for speaker “gg”, elation and hot-anger were the most reliably predicted classes. This analysis suggests that the variation in easiest-to-predict classes arises not only from the intrinsic acoustic properties of emotions but also from speaker-specific expression patterns, including natural expressiveness, emotional intensity, and vocal modulation characteristics. These findings highlight that variability in how speakers express emotions, rather than consistent patterns, plays a critical role in emotion recognition performance. In particular, the detection of more subtle emotions may be strongly affected by speaker-specific expression styles, indicating that general acoustic features alone may be insufficient for robust recognition. Therefore, speaker-adaptive strategies, such as speaker-wise normalization or model conditioning on speaker characteristics, should be considered to enhance the robustness and generalization of emotion recognition models.

c. Which were the most **difficult**? Why do you think they were difficult?

For speaker “mm”, the most difficult classes to predict were cold-anger, despair, and sadness. Specifically, cold-anger exhibited extremely poor performance, with a precision of 0.16 and a recall of 0.15. According to the confusion matrix, cold-anger instances were widely misclassified into contempt, pride, and despair. Cold-anger inherently lacks explosive energy, showing suppressed and controlled vocal patterns. Confusion with other negative emotions, such as contempt and despair, is understandable due to shared low pitch and tense voice quality. However, misclassification into pride is particularly intriguing; it may result from subtle overlapping prosodic traits, such as controlled articulation and confident, monotonic speech patterns, which pride can sometimes exhibit acoustically. Despair also demonstrated low prediction performance, with a precision of 0.17 and a recall of 0.22, and was most frequently misclassified as shame. Both despair and shame share low vocal energy, reduced prosodic variation, and slower tempo, leading to blurred acoustic boundaries between them. Similarly, sadness achieved a precision of 0.18 and a recall of 0.24, with most errors resulting in misclassification as shame. Sadness exhibits flattened prosodic patterns, such as low pitch, reduced loudness, and slower speaking rate, making fine differentiation particularly difficult.

For speaker “gg”, the most difficult classes to predict were sadness, despair, and pride. Specifically, sadness resulted in complete failure, with both precision and recall at 0.00, and was most commonly misclassified as happy, but also confused with interest, anxiety, and boredom. If sadness was expressed subtly, it may have lacked clear prosodic markers, causing the model to misinterpret it as more neutral or positive emotions. Speaker “gg” may have used slightly higher pitch or livelier rhythms even when expressing sadness, exacerbating this confusion. Despair similarly suffered from low prediction accuracy, with a precision of 0.19 and a recall of 0.14,

and was again misclassified as happy. As with sadness, insufficient depression of acoustic features could have led the model to perceive despair as closer to higher-energy emotions. Pride also exhibited poor classification performance, with a precision and recall both at 0.16, often being misclassified as happy. Pride's prosodic characteristics, such as firm and controlled articulation, may have overlapped with confident aspects of happy utterances. Once again, speaker-specific tendencies, such as increased vocal brightness or pitch modulation, may have amplified this confusion.

Across both speakers, sadness and despair were consistently the most difficult classes to predict. Both emotions are characterized by low-energy, low-pitch, slow, flattened prosodic profiles. Unlike high-arousal emotions such as elation or anger, these low-arousal emotions produce subtle and easily overlapped acoustic patterns, complicating reliable classification. Furthermore, for speaker "gg", emotional expression, particularly negative ones, may have been less exaggerated or differentiated acoustically, further diminishing the model's ability to distinguish sadness and despair from other emotions.

d. Based on this analysis, what ideas do you have to further **improve** your classifier?

First, addressing dataset limitations is crucial. The current dataset contains only 2,324 samples across seven speakers and 15 emotions, which may be insufficient, especially under the leave-one-speaker-out cross-validation setting. Class imbalance, particularly the lack of samples for certain emotions per speaker, could severely impact performances. Thus, collecting more samples per speaker for each emotion, or applying targeted data augmentation strategies, such as pitch shifting, time stretching, or vocal perturbation, could help mitigate imbalance and increase the model's robustness.

Second, performance on subtle negative emotions, characterized by low arousal and minimal prosodic variation, such as sadness and despair, was consistently poor. While full-dataset augmentation could lead to overfitting, emotion-specific augmentation, targeting low-energy emotions, may help. Additionally, introducing specialized subnetworks for low-energy emotions or applying cost-sensitive learning, which increases the penalty for misclassifying sadness and despair, could improve the detection of these difficult classes.

Third, although speaker-wise z-score normalization was applied and shown to be effective, further speaker-adaptive modeling may be explored. Techniques such as incorporating speaker embeddings into the model input or using domain adaptation layers to minimize speaker-style variability could be beneficial. However, caution must be taken; while speaker adaptation can enhance robustness, it risks introducing speaker bias, where the model may rely on speaker identity rather than emotion-specific cues. Techniques such as adversarial learning, which is a discriminator that tries to predict speaker identity while the feature extractor aims to prevent it, could be used to decorrelate speaker identity from emotion prediction.

Fourth, the dataset currently consists of isolated short utterances without meaningful semantic content. Utilizing longer and more emotionally rich speech samples, possibly including emotionally charged sentences or

dialogues, could allow the model to better capture the dynamic unfolding of emotions over time. Additionally, incorporating textual features through text embedding models, such as BERT-based, could provide additional semantic context, particularly helpful for distinguishing subtle or context-dependent emotions.

Finally, understanding the tradeoff between precision and recall is essential from an application perspective. In most emotion recognition applications, recall is often prioritized, as missing true emotion (low recall) can be more critical than occasional misclassification (low precision). However, if emotion recognition triggers downstream actions, such as alerts or interventions, precision must also be carefully managed. Analyzing precision-recall balance for each emotion suggests targeted strategies. For emotions with low precision and high recall, enhancing prosodic feature discrimination is key; for emotions with high precision but low recall, techniques like oversampling, threshold tuning, and loss adjustment could be employed to boost recall without sacrificing too much precision.