Notebook 3: Fusion Experiments

In this notebook, we conduct experiments to answer two questions:

1. How should we combine the outputs of each input modality?
2. How should we predict basic emotions and binary sentiment from complex emotion?

Read on for a more detailed explanation of both questions :)

Initialization and Data Processing

We load a JSON of the sentence-by-sentence Hume predictions on the full MELD dataset. See the previous workbooks for how that JSON is generated and cleaned up.

In [1]:
```
import os
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
```

In [2]:
```
def load_json_to_df(directory):
    '''
    converts JSON output to a PD dataframe
    each JSON contains the predicted emotions for one sentence
    '''
    data = []
    for filename in os.listdir(directory):
        if filename.endswith(".json"):
            file_path = os.path.join(directory, filename)
            with open(file_path, 'r') as file:
                content = json.load(file)
                metadata = content['metadata']

                # sentiment data
                face_emotions = lowercase_keys(content['predicted']['face'])
                prosody_emotions = lowercase_keys(content['predicted']['pros
                lang_emotions = lowercase_keys(content['predicted']['lang'])

                data.append({
                    'dialogue_id': metadata['dialogue_id'],
                    'time_start': metadata['time_start'],
                    'time_end': metadata['time_end'],
                    'speaker': metadata['speaker'],
                    'emotion': metadata['emotion'],
                    'sentiment': metadata['sentiment'],
                    'text_content': metadata['text_content'],
                    'file_name': metadata['file_name'],
```

```
                    'face': face_emotions,
                    'prosody': prosody_emotions,
                    'lang': lang_emotions
                })

    return pd.DataFrame(data)
```

In [3]:
```
def lowercase_keys(dictionary):
    '''
    changes all (String) keys in a dictionary to be fully lower case
    '''
    return {key.lower(): value for key, value in dictionary.items()}
```

In [4]:
```
dataset_directory = './dataset/outputs/merged_all'
df = load_json_to_df(dataset_directory)
```

In [5]:
```
def get_emotion_scores(df):
    '''
    adds a column for each Hume/complex emotion, which contains a list of fo
    '''
    df_with_emotions = df.copy()
    for emotion in all_emotions:
        df_with_emotions[emotion] = df.apply(lambda row: [
            row['face'].get(emotion, None),
            row['prosody'].get(emotion, None),
            row['lang'].get(emotion, None)
        ], axis=1)

    return df_with_emotions
```

Exploring Our Emotion Data

Hume has three models that predict emotion based on different modality. One model predicts based on language (the words spoken), another based on prosody (tone of voice, pauses, and vocables), and the last on facial expression. The prosody and face models output 48 emotions. The language model outputs 53 emotions. The five additional emotions output by the language model are {'Annoyance', 'Disapproval', 'Enthusiasm', 'Gratitude', 'Sarcasm'}.

Critically, the MELD dataset contains many sentences labeled with the 7 basic emotions (anger, sadness, fear, joy, surprise, disgust, and neutral). The Hume outputs are *not* the same as the MELD dataset labels. Going forward, we will refer to the Hume outputs as "complex emotions" and the MELD dataset labels as "basic emotions." As you'll see below, a key part of our work is reducing predictions about "complex emotions" to predictions about "basic emotions."

In [6]:
```
lang_emotions = set(df['lang'][0].keys())
prosody_emotions = set(df['prosody'][0].keys())
face_emotions = set(df['face'][0].keys())
```

```
print("Number of outputted emotions in lang, prosody, and face models:", len
print("Emotions outputted by language model but not prosody model:", lang_em
print("Emotions outputted by prosody model but not face model:", prosody_emo

all_emotions = sorted(list(lang_emotions))
print('All emotions:', all_emotions)
```

```
Number of outputted emotions in lang, prosody, and face models: 53 48 48
Emotions outputted by language model but not prosody model: {'sarcasm', 'ann
oyance', 'enthusiasm', 'gratitude', 'disapproval'}
Emotions outputted by prosody model but not face model: set()
All emotions: ['admiration', 'adoration', 'aesthetic appreciation', 'amuseme
nt', 'anger', 'annoyance', 'anxiety', 'awe', 'awkwardness', 'boredom', 'calm
ness', 'concentration', 'confusion', 'contemplation', 'contempt', 'contentme
nt', 'craving', 'desire', 'determination', 'disappointment', 'disapproval',
'disgust', 'distress', 'doubt', 'ecstasy', 'embarrassment', 'empathic pain',
'enthusiasm', 'entrancement', 'envy', 'excitement', 'fear', 'gratitude', 'gu
ilt', 'horror', 'interest', 'joy', 'love', 'nostalgia', 'pain', 'pride', 're
alization', 'relief', 'romance', 'sadness', 'sarcasm', 'satisfaction', 'sham
e', 'surprise (negative)', 'surprise (positive)', 'sympathy', 'tiredness',
'triumph']
```

Emotion Maps

Below, we create maps from the output of Hume AI (up to 53 emotions) to the seven basic emotions in our dataset. We also map the Hume AI emotions to positive/negative sentiment. Annotations below are manually created. Interesting future exploration could involve mapping each Hume/complex emotion to a weighted sum of the 7 basic emotions, potentially through training an ML model.

In [87]:
```
BASIC_TO_COMPLEX = {
  'anger': ['anger', 'annoyance', 'disapproval'],
  'fear': ['anxiety', 'doubt', 'fear', 'horror'],
  'joy': ['admiration', 'adoration', 'amusement', 'contentment', 'desire', '
  'sadness': ['disappointment', 'distress', 'empathic pain', 'guilt', 'nosta
  'surprise': ['awe', 'confusion', 'realization', 'surprise (negative)', 'su
  'disgust': ['contempt', 'disgust', 'envy', 'sarcasm'],
  'neutral': ['aesthetic appreciation', 'awkwardness', 'boredom', 'calmness'
}
```

In [88]:
```
SENTIMENT_TO_EMOTION = {
    'positive': [
        'admiration', 'adoration', 'aesthetic appreciation', 'amusement',
        'awe', 'calmness', 'contentment', 'craving', 'desire',
        'determination', 'ecstasy', 'enthusiasm', 'entrancement',
        'excitement', 'gratitude', 'interest', 'joy', 'love', 'nostalgia',
        'pride', 'realization', 'relief', 'romance', 'satisfaction',
        'surprise (positive)', 'triumph'
    ],
    'negative': [
        'anger', 'annoyance', 'anxiety', 'awkwardness', 'boredom',
        'confusion', 'contempt', 'disappointment', 'disapproval',
        'disgust', 'distress', 'doubt', 'embarrassment', 'empathic pain',
        'envy', 'fear', 'guilt', 'horror', 'pain', 'sadness',
```

```
        'sarcasm', 'shame', 'surprise (negative)', 'tiredness'
    ],
    'neutral': [
        'concentration', 'contemplation', 'sympathy'
    ]
}
```

Methods To Fuse Modalities

Hume has a facial expression model (predicts based on frame capture of facial expressions in a video), a prosody model (predicts based on signal waveform), and language model (predicts based off words spoken). Each model outputs predictions for up to 53 emotions based on the given input modality. We experiment with two methods to combine the outputs from different modalities to obtain a single number representing the intensity of each emotion.

1. The first method is a simple sum. To get the intensity of an emotion, this function sums the intensity predicated for each modality. In other words: `awe_intensity = face_awe_intensity + prosody_awe_intensity + lang_awe_intensity`
2. The second method is a relative sum. To get the intensity of an emotion, this function takes sums the intensity predicated for each modality, weighted by the predictive accuracy of that modality alone. In other words: `awe_intensity = face_awe_intensity * relative accuracy of face-only prediction + ... (the same for prosody and language)`

Method 1: Simple Sum

In [9]:
```
def get_simple_sum(df):
    '''
    to get the intensity of an emotion, this function sums the intensity pre
    awe_intensity = face_awe_intensity + prosody_awe_intensity + lang_awe_in
    '''
    simple_sum_df = df.copy()
    for emotion in all_emotions:
        simple_sum_df[emotion] = df[emotion].apply(lambda x: sum([i for i in
    return simple_sum_df
```

Method 2: Relative Sum

We calculate the extent to which each individual modality predicts the final emotion, and use the relative accuracy of each modality to weight the final sum.

2a: Accuracy of Individual Modalities

First, we test each individual modality to see how predictive it is of the final emotion.

In [10]:
```
def get_predictions(df, mapping, modality):
    '''
    predicts which basic emotion dominates by taking the mean of all complex
    '''
    intensities = pd.DataFrame()
    basic_emotions = sorted(list(mapping.keys()))

    for basic_emotion in basic_emotions:
        # this is a disgusting list comprehension that came from flattening
        complex_emotions = pd.concat([df[modality].apply(lambda row: row.get

        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            intensities[basic_emotion] = np.nanmean(complex_emotions, axis=1

    # drop all rows that have all nan values
    intensities = intensities.dropna(how='all')

    y_pred = intensities.idxmax(axis=1)
    return y_pred
```

In [11]:
```
def get_accuracy(df, target, modality):
    '''
    reports the accuracy of the approach above compared to ground truth
    @param df with a column `y_pred` and a column with the name `target`
    @returns the percent of rows that have identical `y_pred` and `target` v
    '''
    if target == 'emotion':
        mapping = BASIC_TO_COMPLEX
    elif target == 'sentiment':
        mapping = SENTIMENT_TO_EMOTION
    else:
        raise Exception('Invalid target')

    y_pred = get_predictions(df, mapping, modality)
    comparable_columns = df[target].loc[y_pred.index]
    total_sentences = len(comparable_columns)
    accuracy = np.sum(y_pred == comparable_columns) / total_sentences
    return accuracy
```

In [12]:
```
def calc_relative_weights(df, show_results=False, return_results=False):
    # run on all combinations of emotions and modalities
    targets = ['emotion', 'sentiment']
    modalities = ['face','prosody','lang']

    results = pd.DataFrame(index=targets, columns=modalities)

    for target in targets:
        for modality in modalities:
            accuracy = get_accuracy(df, target, modality)
            results.at[target, modality] = accuracy
    results = results.astype(float)

    # calculate and return relative weights
    relative_weights = results.apply(lambda row: row / np.sum(row), axis=1)
```

```
    if show_results:
        print("Accuracy by Modality")
        display(results)
        print("Relative Modality Weights")
        display(relative_weights)

    if return_results:
        return relative_weights, results
    return relative_weights
```

In [13]:
```
unfused_data = get_emotion_scores(df)
weights, results = calc_relative_weights(unfused_data, show_results=True, re
```

Accuracy by Modality

	face	prosody	lang
emotion	0.223697	0.280374	0.357610
sentiment	0.447393	0.441121	0.511671

Relative Modality Weights

	face	prosody	lang
emotion	0.259605	0.325380	0.415014
sentiment	0.319524	0.315045	0.365431

In [14]:
```
# Older matplotlib implementation that still had nice results

# def display_heatmap(results, title='Accuracy', xlabel='Modalities', ylabel
#     indices = results.index.tolist()
#     columns = results.columns.tolist()

#     # plot heatmap
#     fig, ax = plt.subplots()
#     heatmap = ax.imshow(results, cmap='viridis', interpolation='nearest')

#     ax.set_xticks(np.arange(len(columns)))
#     ax.set_yticks(np.arange(len(indices)))
#     ax.set_xticklabels(columns)
#     ax.set_yticklabels(indices)
#     plt.setp(ax.get_xticklabels(), ha="right", rotation_mode="anchor")
#     plt.colorbar(heatmap)

#     ax.set_title(title)
#     plt.xlabel(xlabel)
#     plt.ylabel(ylabel)

#     if savetitle:
#         output_path = os.path.join('plots', savetitle + '.png')
#         plt.savefig(output_path)

#     plt.show()
```

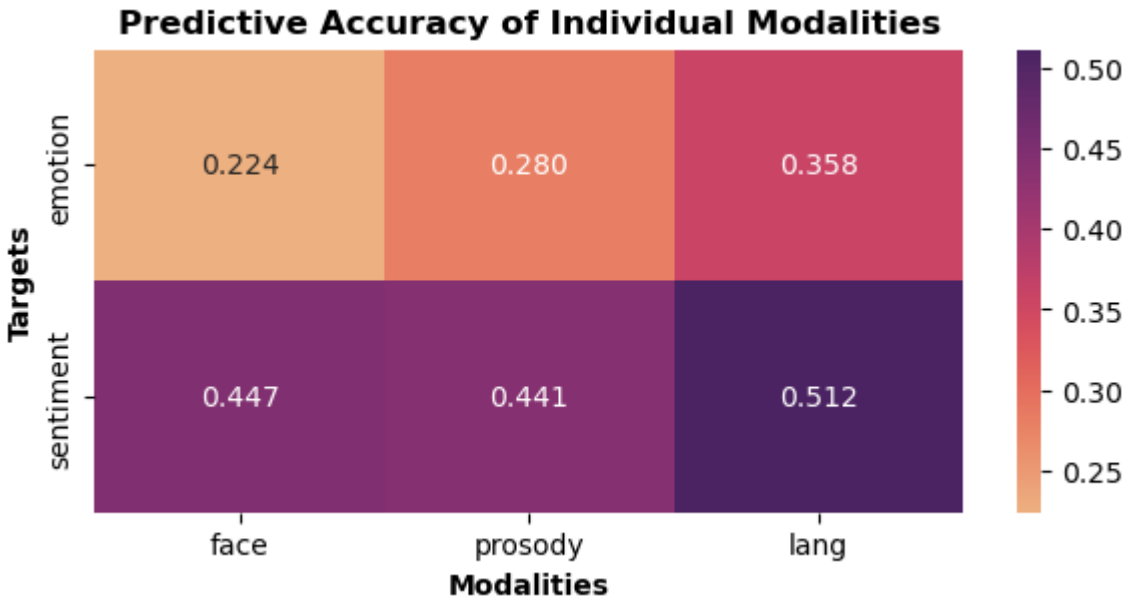
In [56]:
```
def display_heatmap(results, title='Accuracy', xlabel='Modalities', ylabel='

    plt.figure(figsize=dim)
    heatmap = sns.heatmap(results, annot=True, cmap=cmap, cbar=True, fmt=".3
    heatmap.axes.set_title(title, weight='bold')
    plt.xlabel(xlabel, weight='bold')
    plt.ylabel(ylabel, weight='bold')

    if savetitle:
        output_path = os.path.join('plots', f"sns-{savetitle}.png")
        plt.savefig(output_path, dpi=400, bbox_inches='tight')

    plt.show()
    plt.close()
```

In [57]:
```
display_heatmap(results, title='Predictive Accuracy of Individual Modalities
```



2b: Fusion

And now let's put those relative weights to use!

In [17]:
```
def get_relative_weights(df, target):
    '''
    to get the intensity of an emotion, this function takes sums the intensi
    awe_intensity = face_awe_intensity * relative accuracy of face-only pred
    '''
    weights_df = df.copy()
    # df with rows of form [face_weight, prosody_weight, lang_weight]
    weights = calc_relative_weights(df)
    target_weights = list(weights.loc[target])
    for emotion in all_emotions:
        weights_df[emotion] = df[emotion].apply(lambda row: sum([modality *
    return weights_df
```

Fused Data

You can see the results below for unfused data, the simple sum, and the relative sum.

In [18]:
```
unfused_data = get_emotion_scores(df)
unfused_data[all_emotions].head()
```

Out[18]:

	admiration	adoration	aesthetic appreciation	
0	[0.1813952475786209, 0.013488777913153172, 0.0...	[0.17787836492061615, 0.00977570191025734, 0.0...	[0.09267013520002365, 0.010443544946610928, 0....	[0.3887 0.19728
1	[0.2766248881816864, 0.15509642660617828, 0.13...	[0.284834623336792, 0.13949357345700264, 0.124...	[0.13193348050117493, 0.08952821232378483, 0.1...	[0.4468 0.09358
2	[0.05406120419502258, 0.05932471761479974, 0.0...	[0.04063199833035469, 0.04072318458929658, 0.0...	[0.038426585495471954, 0.007938217604532838, 0...	[0.09366 0.18757
3	[0.0761360302567482, 0.02562095010653138, 0.01...	[0.04687286168336868, 0.024671880481764673, 0....	[0.051195625215768814, 0.018752328492701052, 0...	[0.10179 0.08181
4	[0.12609320878982544, 0.026591897010803223, 0....	[0.14971742033958435, 0.01609588973224163, 0.0...	[0.06984098255634308, 0.021242598071694374, 0....	[0.29830 0.11833

5 rows × 53 columns

In [19]:
```
simple_sum_df = get_simple_sum(unfused_data)
simple_sum_df[all_emotions].head()
```

Out[19]:

	admiration	adoration	aesthetic appreciation	amusement	anger	annoyance	anxiety	
0	0.201632	0.190222	0.107997	0.594856	0.154327	0.069127	0.236938	
1	0.571381	0.548424	0.330574	0.566007	0.146984	0.009968	0.121462	
2	0.119949	0.090469	0.048201	0.305937	0.495151	0.418941	0.313456	(
3	0.116525	0.085260	0.088551	0.306416	0.184289	0.268749	0.268937	(
4	0.156628	0.168514	0.102729	0.433415	0.138296	0.108364	0.214304	

5 rows × 53 columns

In [20]:
```
weighted_df = get_relative_weights(unfused_data, 'emotion')
weighted_df[all_emotions]
```

Out[20]:

	admiration	adoration	aesthetic appreciation	amusement	anger	annoyance	anxiet
0	0.054281	0.050425	0.029482	0.168770	0.049513	0.028689	0.07083
1	0.180239	0.170835	0.108665	0.157073	0.046078	0.004137	0.03500
2	0.036062	0.027581	0.013321	0.095599	0.174560	0.173867	0.0917
3	0.034231	0.025888	0.027113	0.104013	0.055265	0.111535	0.0748
4	0.043023	0.045226	0.029876	0.122907	0.041460	0.044972	0.06017
...	...	...	...	...	...	...	
1066	0.068599	0.086547	0.035237	0.078073	0.045804	0.038687	0.0772
1067	0.074282	0.063510	0.033490	0.168561	0.073576	0.013366	0.04936
1068	0.053845	0.057761	0.033160	0.222782	0.150564	0.254453	0.07358
1069	0.033458	0.031642	0.023443	0.072632	0.029218	0.056302	0.10923
1070	0.024902	0.017666	0.020236	0.076961	0.085007	0.166288	0.13322

1071 rows × 53 columns

In [21]:
```
weighted_df = get_relative_weights(unfused_data, 'sentiment')
weighted_df[all_emotions]
```

Out[21]:

	admiration	adoration	aesthetic appreciation	amusement	anger	annoyance	anxiet
0	0.064676	0.060855	0.034685	0.189591	0.049230	0.025261	0.07668
1	0.188287	0.180307	0.110234	0.181609	0.046477	0.003643	0.03867
2	0.038362	0.029143	0.015450	0.098048	0.167462	0.153094	0.10144
3	0.037796	0.027762	0.029064	0.103178	0.060312	0.098209	0.08614
4	0.050108	0.053896	0.033264	0.138727	0.044055	0.039599	0.06937
...	...	...	...	...	...	...	
1066	0.072168	0.086022	0.037989	0.084383	0.049213	0.034065	0.08388
1067	0.081815	0.070732	0.038939	0.195860	0.073604	0.011769	0.05199
1068	0.065275	0.070013	0.040000	0.248654	0.141965	0.224052	0.07493
1069	0.040162	0.038167	0.027344	0.080787	0.031445	0.049575	0.11509
1070	0.029137	0.020449	0.023482	0.079177	0.086593	0.146421	0.14006

1071 rows × 53 columns

Predicting the Base Emotion

Now we have an intensity score for each complex emotion. How do we combine the complex emotions to predict the base emotion? We try two approaches.

1. **Group Average:** The first approach maps the complex emotions into label groups (each complex emotion is assigned one basic emotion and one sentiment -- either 'positive' or 'negative'). We average the intensities of all the complex emotions that correspond to a given label in order to get the average intensity of that label. In other words, to get the average 'anger' intensity, we average the intensities of the complex emotions that correspond to anger: 'anger', 'annoyance', and 'disapproval'.
2. **Classifier:** Our second approach is to train a classifier (a small neural network) that learns the relationship between the complex emotions and each label.

Predicting by Group Average

In [22]:
```
def pred_highest_intensity(df, target, show=True):

    # get mapping based on prediction target
    if target == 'emotion':
        mapping = BASIC_TO_COMPLEX
    elif target == 'sentiment':
        mapping = SENTIMENT_TO_EMOTION
    else:
        raise Exception('Invalid target')

    labels = sorted(list(mapping.keys()))
    intensities = pd.DataFrame()
    individual_emotions = pd.DataFrame()

    for label in labels:
        scores = pd.concat([df[complex_emotion] for complex_emotion in mappi
        individual_emotions = pd.concat([individual_emotions, scores], axis=

        # Suppress runtime warnings for mean of empty slice
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            intensities[label] = np.nanmean(scores, axis=1)

    if show:
        print('Intensity of Complex Emotions (Summed Across All Modalities)'
        display(individual_emotions.head())
        print('Intensity of Each Label (Averaged Across All Complex Emotions
        display(intensities.head())
    return intensities.idxmax(axis=1)
```

In [23]:
```
pred_simple = pred_highest_intensity(simple_sum_df, 'sentiment')
```

```
Intensity of Complex Emotions (Summed Across All Modalities)
```

	anger	annoyance	anxiety	awkwardness	boredom	confusion	contempt	disa
0	0.154327	0.069127	0.236938	0.465328	0.163396	0.880139	0.177269	
1	0.146984	0.009968	0.121462	0.338550	0.300334	0.254876	0.233129	
2	0.495151	0.418941	0.313456	0.342286	0.222464	1.288661	0.316636	
3	0.184289	0.268749	0.268937	0.443272	0.304051	0.907013	0.311960	
4	0.138296	0.108364	0.214304	0.271059	0.247492	0.381291	0.239528	

5 rows × 53 columns

```
Intensity of Each Label (Averaged Across All Complex Emotions Corresponding
to that Label)
```

	negative	neutral	positive
0	0.184877	0.339033	0.230474
1	0.158199	0.244558	0.350006
2	0.337878	0.205489	0.125146
3	0.287748	0.274968	0.137454
4	0.197643	0.460091	0.219401

In [24]:
```
pred_weights = pred_highest_intensity(weighted_df, 'sentiment')
```

```
Intensity of Complex Emotions (Summed Across All Modalities)
```

	anger	annoyance	anxiety	awkwardness	boredom	confusion	contempt	disa
0	0.049230	0.025261	0.076688	0.149679	0.053169	0.297665	0.057515	
1	0.046477	0.003643	0.038674	0.108054	0.096065	0.081518	0.074813	
2	0.167462	0.153094	0.101444	0.111333	0.071444	0.431062	0.104218	
3	0.060312	0.098209	0.086143	0.147238	0.097940	0.296983	0.103358	
4	0.044055	0.039599	0.069371	0.087467	0.079854	0.124077	0.078144	

5 rows × 53 columns

```
Intensity of Each Label (Averaged Across All Complex Emotions Corresponding
to that Label)
```

	negative	neutral	positive
0	0.060677	0.112263	0.074775
1	0.050442	0.078400	0.115695
2	0.111620	0.066815	0.040610
3	0.094770	0.089196	0.045037
4	0.064171	0.154991	0.071579

In [25]:
```
def score(df, target):
    pred = pred_highest_intensity(df, target, show=False)
    return np.nanmean(df[target] == pred)
```

Prediction by Group Average: Simple Sum vs. Weighted Sum Accuracy Comparison

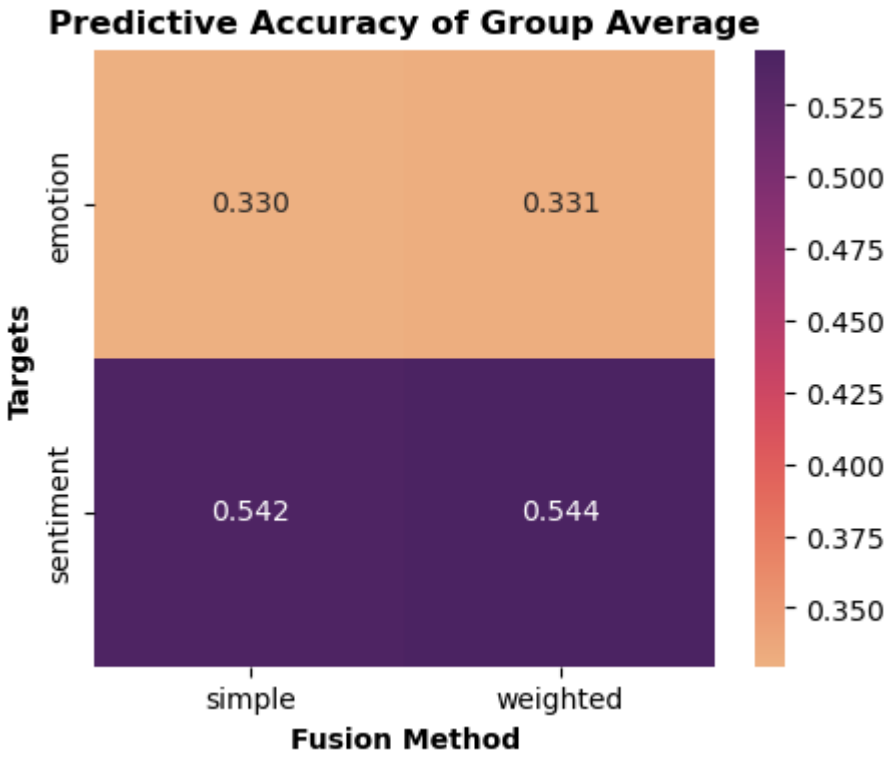
In [26]:
```
simple_emotion = score(simple_sum_df, 'emotion')
simple_sentiment = score(simple_sum_df, 'sentiment')
weighted_emotion = score(weighted_df, 'emotion')
weighted_sentiment = score(weighted_df, 'sentiment')
```

In [27]:
```
data = {
    'simple': [simple_emotion, simple_sentiment],
    'weighted': [weighted_emotion, weighted_sentiment]
}
index = ['emotion', 'sentiment']
fusion_results = pd.DataFrame(data, index=index)
fusion_results
```

Out[27]:

	simple	weighted
emotion	0.329599	0.331466
sentiment	0.541550	0.544351

In [58]:
```
display_heatmap(fusion_results, 'Predictive Accuracy of Group Average', xlab
```



In [29]:
```
print('Accuracy at predicting emotion:', round(simple_emotion * 100, 2))
print('Accuracy at predicting sentiment:', round(simple_sentiment * 100, 2))
```

```
Accuracy at predicting emotion: 32.96
Accuracy at predicting sentiment: 54.15
```

In [30]:
```
print('Accuracy at predicting emotion:', round(weighted_emotion * 100, 2))
print('Accuracy at predicting sentiment:', round(weighted_sentiment * 100, 2
```

```
Accuracy at predicting emotion: 33.15
Accuracy at predicting sentiment: 54.44
```

Observations: Odd that sentiment prediction is identical (a quick look through the data shows that every prediction is the same for all sentences).

Predicting by Classifier

Let's try training a small neural network that takes in the simple sum complex emotions and predicts the final emotion.

In [31]:
```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
```

In [32]:
```
inputs = simple_sum_df[all_emotions]
inputs.head()
```

Out[32]:

	admiration	adoration	aesthetic appreciation	amusement	anger	annoyance	anxiety
0	0.201632	0.190222	0.107997	0.594856	0.154327	0.069127	0.236938
1	0.571381	0.548424	0.330574	0.566007	0.146984	0.009968	0.121462
2	0.119949	0.090469	0.048201	0.305937	0.495151	0.418941	0.313456
3	0.116525	0.085260	0.088551	0.306416	0.184289	0.268749	0.268937
4	0.156628	0.168514	0.102729	0.433415	0.138296	0.108364	0.214304

5 rows × 53 columns

In [33]:
```
labels = simple_sum_df[['emotion', 'sentiment']]
labels.head()
```

Out[33]:

	emotion	sentiment
0	neutral	neutral
1	joy	positive
2	surprise	negative
3	sadness	negative
4	joy	positive

In [65]:
```
def train_model(df, target, return_pred=False):
    '''
    @param target is either 'emotion' or 'sentiment'
    '''
    inputs = df[all_emotions]
    X_train, X_test, y_train, y_test = train_test_split(inputs, labels[targe
    model = MLPClassifier(random_state=42, max_iter=1000).fit(X_train, y_tra
    accuracy = model.score(X_test, y_test)
    display_acc = round(accuracy * 100, 2)
    print(f"Accuracy at predicting {target} is {display_acc}%")
    if return_pred:
        return accuracy, model.predict(X_test)
    return accuracy
```

In [66]:
```
simple_emotion = train_model(simple_sum_df, 'emotion')
simple_sentiment = train_model(simple_sum_df, 'sentiment')
weighted_emotion = train_model(weighted_df, 'emotion')
weighted_sentiment = train_model(weighted_df, 'sentiment')
```

```
/Users/selenazhang/miniconda3/envs/mui/lib/python3.12/site-packages/sklearn/
neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (1000) reached and the optimization hasn't con
verged yet.
  warnings.warn(
Accuracy at predicting emotion is 43.26%
```

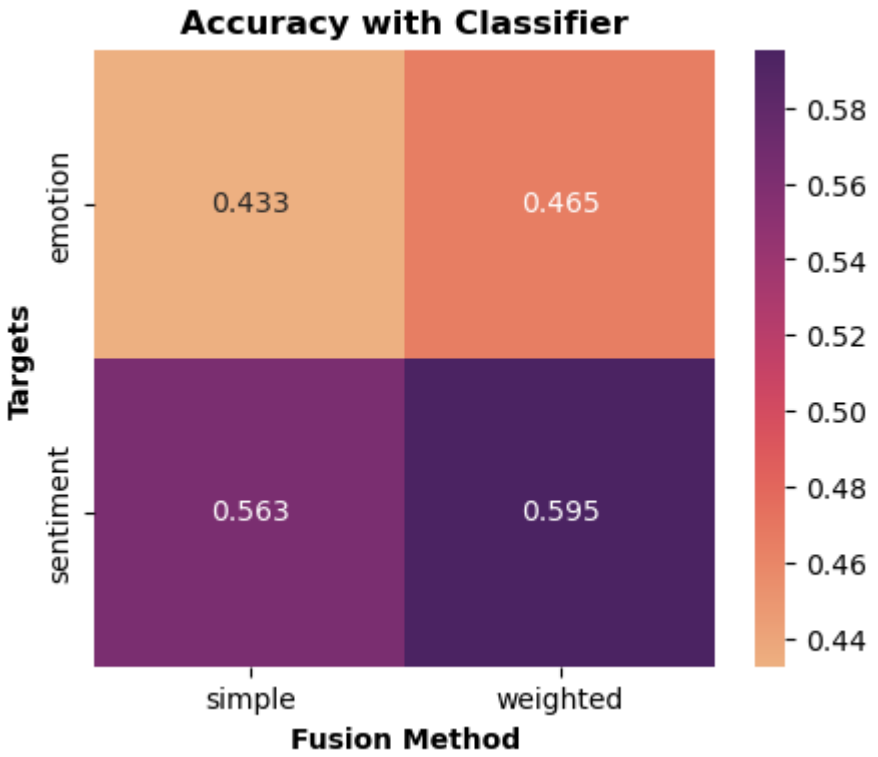
```
/Users/selenazhang/miniconda3/envs/mui/lib/python3.12/site-packages/sklearn/
neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (1000) reached and the optimization hasn't con
verged yet.
  warnings.warn(
Accuracy at predicting sentiment is 56.28%
/Users/selenazhang/miniconda3/envs/mui/lib/python3.12/site-packages/sklearn/
neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (1000) reached and the optimization hasn't con
verged yet.
  warnings.warn(
Accuracy at predicting emotion is 46.51%
Accuracy at predicting sentiment is 59.53%
/Users/selenazhang/miniconda3/envs/mui/lib/python3.12/site-packages/sklearn/
neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (1000) reached and the optimization hasn't con
verged yet.
  warnings.warn(
```

In [36]:
```
clf_data = {
    'simple': [simple_emotion, simple_sentiment],
    'weighted': [weighted_emotion, weighted_sentiment]
}
clf_results = pd.DataFrame(clf_data, index=index)
clf_results
```

Out[36]:

	simple	weighted
emotion	0.432558	0.465116
sentiment	0.562791	0.595349

In [59]:
```
display_heatmap(clf_results, title='Accuracy with Classifier', xlabel='Fusic
```



Observations: The neural network is much better at predicting emotion but less accurate at predicting sentiment (and even worse than random chance). Changing the random state of the train-test split also significantly changes the accuracy, suggesting that the model is probably overfitting and the small sample size of data is skewing results.

Evaluations

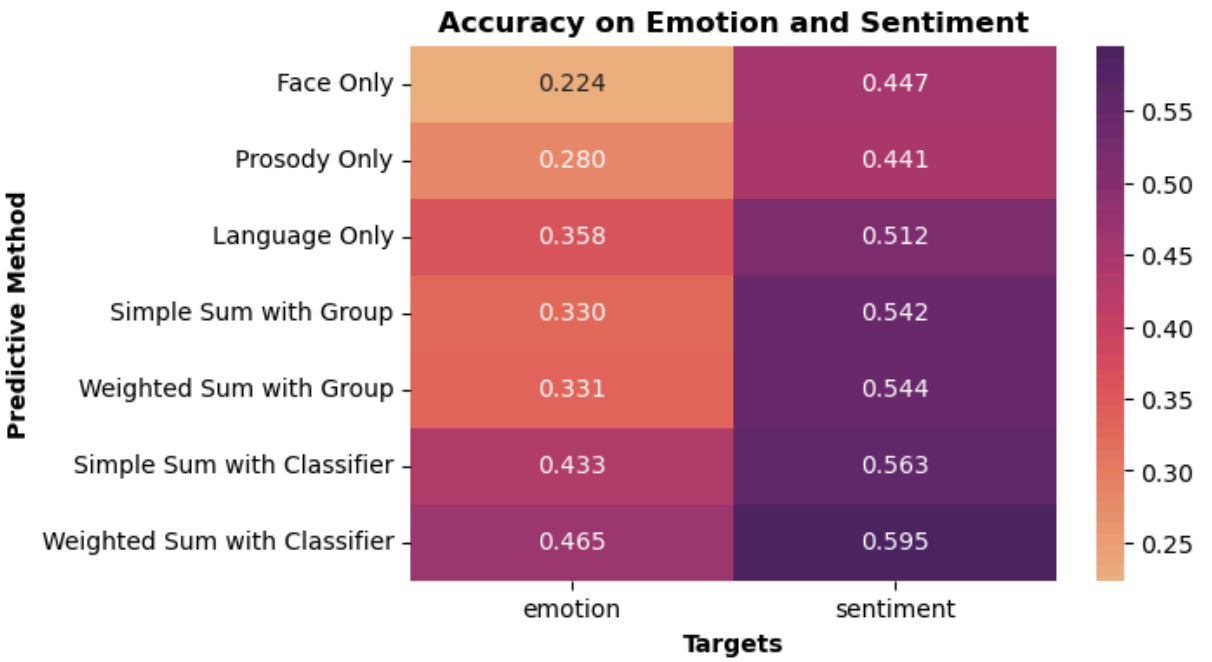
We show that our multimodal approach does better than unimodal approaches. Our approach performs about 10 percentage points worse than the SOTA (14.2% worse). We also evaluate the extent to which our predictions are consistent with each other.

In [38]:
```
results_renamed = results.rename(columns={'face': 'Face Only', 'prosody': 'P
group_renamed = fusion_results.rename(columns={'simple': 'Simple Sum with Gr
clf_renamed = clf_results.rename(columns={'simple': 'Simple Sum with Classif
all_results = pd.concat([results_renamed, group_renamed, clf_renamed], axis=
all_results
```

Out[38]:

	Face Only	Prosody Only	Language Only	Simple Sum with Group	Weighted Sum with Group	Simple Sum with Classifier	Weighted Sum with Classifier
emotion	0.223697	0.280374	0.357610	0.329599	0.331466	0.432558	0.465116
sentiment	0.447393	0.441121	0.511671	0.541550	0.544351	0.562791	0.595349

In [60]:
```
display_heatmap(all_results.T, title='Accuracy on Emotion and Sentiment', xl
```



In [61]:
```
# we omit surprise, which can be either positive or negative
BASIC_TO_SENTIMENT = {
    'anger': 'negative',
    'sadness': 'negative',
    'neutral': 'neutral',
    'joy': 'positive',
    'disgust': 'negative',
    'fear': 'negative',
}
```

In [70]:
```
_, pred_emotion = train_model(simple_sum_df, 'emotion', return_pred=True)
_, pred_sentiment = train_model(simple_sum_df, 'sentiment', return_pred=True
```

```
/Users/selenazhang/miniconda3/envs/mui/lib/python3.12/site-packages/sklearn/
neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (1000) reached and the optimization hasn't con
verged yet.
  warnings.warn(
Accuracy at predicting emotion is 43.26%
Accuracy at predicting sentiment is 56.28%
/Users/selenazhang/miniconda3/envs/mui/lib/python3.12/site-packages/sklearn/
neural_network/_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (1000) reached and the optimization hasn't con
verged yet.
  warnings.warn(
```

In [79]:
```
clf_pred = pd.DataFrame(np.vstack([pred_emotion, pred_sentiment]), index=['e
no_surprise = clf_pred['emotion'] != 'surprise'
clf_no_surprise = clf_pred[no_surprise]
```

In [106…
```
basic_emotions = ['fear', 'disgust', 'sadness', 'anger', 'neutral', 'surpris
emotion_frequencies = [np.sum(clf_pred['emotion'] == emotion) for emotion in
```

```
relative_frequency = [round((freq / np.sum(emotion_frequencies)) * 100, 2) f
freq_data = {
    'emotions': basic_emotions,
    'frequency': emotion_frequencies,
    'relative frequency (%)': relative_frequency
}
freq = pd.DataFrame(freq_data)
freq
```

Out[106…

	emotions	frequency	relative frequency (%)
0	fear	4	1.86
1	disgust	0	0.00
2	sadness	10	4.65
3	anger	29	13.49
4	neutral	126	58.60
5	surprise	15	6.98
6	joy	31	14.42

In [108…

```
clf_no_surprise['predicted sentiment'] = clf_no_surprise['emotion'].apply(la
```

```
/var/folders/dp/jng7b5b51rbbqd9mnzyrqxw80000gn/T/ipykernel_91479/3616678805.
py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  clf_no_surprise['predicted sentiment'] = clf_no_surprise['emotion'].apply
(lambda emotion : BASIC_TO_SENTIMENT[emotion])
```

In [111…

```
np.sum(clf_no_surprise['sentiment'] == clf_no_surprise['predicted sentiment'
```

Out[111… `0.795`

Observations: Our classifier never predicts digust!

In [40]:

```
# def train_no_fusion(inputs, target):
#     '''
#     @param target is either 'emotion' or 'sentiment'
#     '''
#     X_train, X_test, y_train, y_test = train_test_split(inputs, labels[tar
#     model = MLPClassifier(random_state=42, max_iter=1000).fit(X_train, y_t
#     accuracy = model.score(X_test, y_test)
#     display_acc = round(accuracy * 100, 2)
#     print(f"Accuracy at predicting {target} is {display_acc}%")
# inputs = np.array(unfused_data[all_emotions].map(lambda row: np.array(row)
# flattened = inputs.reshape(inputs.shape[0], -1)
# # TODO: process nan better
# train_no_fusion(flattened, 'emotion')
```

Selecting Significant Emotions

We graph the intensity across all the emotions, on all modalities. We can then choose a threshold for when an emotion is 'significant,' and only use 'significant' emotions to predict the final sentiment of the sentence.

In [41]:
```
mean_intensity = simple_sum_df[all_emotions].mean(axis=None)
```

In [42]:
```
THRESHOLD = mean_intensity
```

Note to self: would be good to modularize how modalities are combined (simple vs. relative sum) and then how the basic emotion is predicted (highest intensity vs. neural net)