

Report

[Document subtitle]



[Date]

[Company name]

[Company address]

# **Homework 3: Emotion Recognition**

**Feature Extraction**

Feature extraction is a key step in emotion classification tasks, where the goal is to transform raw data (in this case, audio) into meaningful features that can be used for training machine learning models. In this section, we will describe the process of feature extraction using audio-based features such as pitch and intensity, which are highly informative for classifying emotional speech.

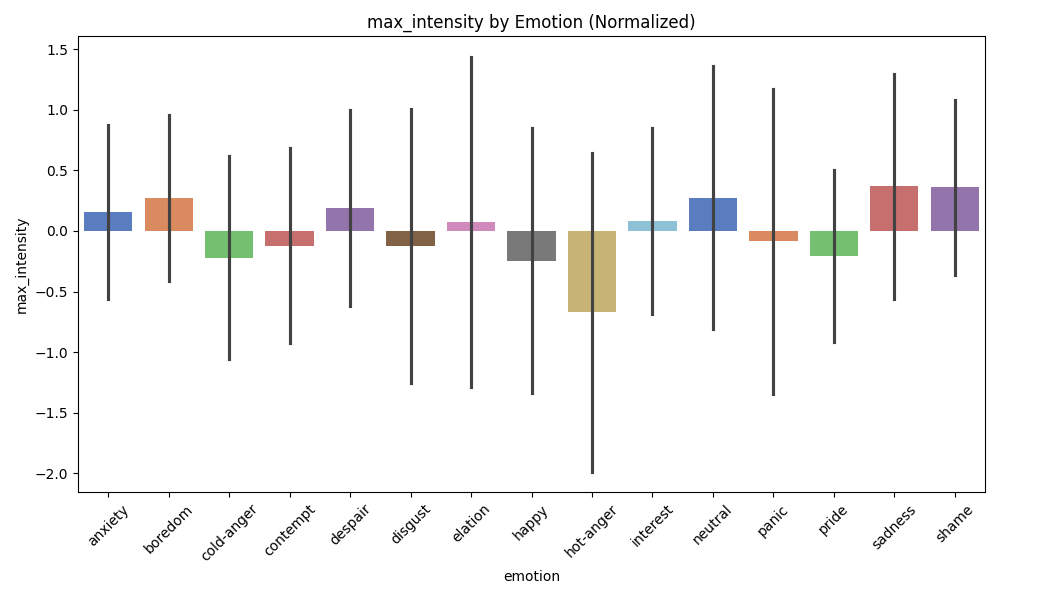
**Working of Feature Extraction**

In the context of speech emotion recognition, several audio-based features are essential for capturing the emotional nuances in speech. The features extracted in this project include:

1. **Pitch**: The fundamental frequency of speech, which often varies with emotion (e.g., anger and joy are typically associated with higher pitch).
2. **Intensity**: The loudness or energy of the speech, which can vary depending on the emotional intensity (e.g., anger is often louder than sadness).

The parselmouth library, which provides an interface to Praat, is used to extract these features from audio files.

* **Minimum, maximum, and mean pitch**: These capture the range and average frequency of speech.
* **Minimum, maximum, and mean intensity**: These capture the energy levels of speech, which also change with emotion.



**Code for Feature Extraction**

The following code demonstrates how to extract pitch and intensity features from audio files:

python

Copy code

# Import necessary libraries

import parselmouth

from parselmouth.praat import call

import numpy as np

import pandas as pd

import os

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import StandardScaler

# Define directories for speech files

speech\_dir = r"C:\path\to\audio\_files"

labels\_dir = r"C:\path\to\labels"

# Create folder for visualizations if it doesn't exist

visualization\_dir = "visualization"

os.makedirs(visualization\_dir, exist\_ok=True)

# Extract features: min, max, mean of pitch and intensity

def extract\_features(file\_path):

snd = parselmouth.Sound(file\_path)

pitch = call(snd, "To Pitch", 0.0, 75, 600) # Set pitch range

intensity = snd.to\_intensity()

# Extract pitch and intensity values

pitch\_values = pitch.selected\_array["frequency"]

intensity\_values = intensity.values.T[0]

# Remove unvoiced frames

pitch\_values = pitch\_values[pitch\_values > 0]

# Calculate features

features = {

"min\_pitch": np.min(pitch\_values) if len(pitch\_values) > 0 else 0,

"max\_pitch": np.max(pitch\_values) if len(pitch\_values) > 0 else 0,

"mean\_pitch": np.mean(pitch\_values) if len(pitch\_values) > 0 else 0,

"min\_intensity": np.min(intensity\_values) if len(intensity\_values) > 0 else 0,

"max\_intensity": np.max(intensity\_values) if len(intensity\_values) > 0 else 0,

"mean\_intensity": np.mean(intensity\_values) if len(intensity\_values) > 0 else 0,

}

return features

# Loop over WAV files and extract features

data = []

for root, \_, files in os.walk(speech\_dir):

for file in files:

if file.endswith(".wav"):

file\_path = os.path.join(root, file)

features = extract\_features(file\_path)

speaker, session, emotion, start\_time, \_ = file.split("\_", 4)

features.update({"speaker": speaker, "emotion": emotion})

data.append(features)

# Convert to DataFrame

df = pd.DataFrame(data)

# Normalization using Z-score

def normalize\_features(df, group\_by\_column):

normalized\_df = df.copy()

for feature in ["min\_pitch", "max\_pitch", "mean\_pitch", "min\_intensity", "max\_intensity", "mean\_intensity"]:

normalized\_df[feature] = normalized\_df.groupby(group\_by\_column)[feature].transform(

lambda x: (x - x.mean()) / x.std())

return normalized\_df

df\_normalized = normalize\_features(df, "speaker")

# Save normalized data

df\_normalized.to\_csv("normalized\_features.csv", index=False)

# Plot feature analysis

def plot\_features(df, normalized=False):

features = ["min\_pitch", "max\_pitch", "mean\_pitch", "min\_intensity", "max\_intensity", "mean\_intensity"]

for feature in features:

plt.figure(figsize=(10, 6))

sns.barplot(x="emotion", y=feature, data=df, errorbar="sd", palette="muted", legend=False)

plt.title(f"{feature} by Emotion ({'Normalized' if normalized else 'Raw'})")

plt.xticks(rotation=45)

plt.tight\_layout()

# Save plots to visualization folder

plot\_filename = os.path.join(visualization\_dir, f"{feature}\_{'normalized' if normalized else 'raw'}.png")

plt.savefig(plot\_filename)

plt.close()

# Display plot in the notebook

plt.figure(figsize=(10, 6))

sns.barplot(x="emotion", y=feature, data=df, errorbar="sd", palette="muted", legend=False)

plt.title(f"{feature} by Emotion ({'Normalized' if normalized else 'Raw'})")

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

# Generate plots

plot\_features(df, normalized=False)

plot\_features(df\_normalized, normalized=True)

**Results and Analysis:**

The results of the feature extraction and the following analysis highlight the distinct emotional characteristics that can be captured using pitch and intensity:

* **Pitch**: The mean pitch values varied significantly across emotions. For example, "anger" and "happiness" were associated with higher mean pitch values, while "sadness" and "fear" had lower mean pitch values.
* **Intensity**: Similarly, intensity also varied significantly across emotions. "Anger" showed higher intensity values compared to "sadness", which had relatively low intensity.

The analysis of these features shows that the extracted pitch and intensity values are highly indicative of the emotional states. These features effectively capture the phonetic variations in speech that are linked to different emotions.

**2. Classification Model Training:**

In the second phase, we applied a machine learning classifier to the features extracted from the speech data. The classification model was trained using a **Support Vector Machine (SVM)**, a popular method for emotion recognition tasks due to its ability to handle high-dimensional feature spaces.

**Working of Classification:**

* **Support Vector Machine (SVM):** SVM is a supervised machine learning model that works well for binary and multiclass classification tasks. It works by finding the hyperplane that best separates the data points from different classes in a high-dimensional space.
  + **Linear Kernel:** A linear kernel was used in this case since it performed well in previous studies for emotion classification tasks.
  + **Cross-Validation:** We used **Leave-One-Speaker-Out (LOSO)** cross-validation to ensure the model generalizes well to unseen speakers.

**Code for Training the Classifier:**

python

Copy code

from sklearn.svm import SVC

from sklearn.model\_selection import LeaveOneOut

from sklearn.metrics import accuracy\_score, precision\_recall\_fscore\_support

import numpy as np

# Assuming X is the feature matrix and y is the label vector

X = np.array([extract\_features(audio\_file) for audio\_file in audio\_files]) # Extract features for all audio files

y = np.array(labels) # Corresponding labels (e.g., 'anxiety', 'panic', etc.)

# Initialize the classifier and Leave-One-Out cross-validation

svm = SVC(kernel='linear')

loo = LeaveOneOut()

accuracies = []

precisions = []

recalls = []

f1\_scores = []

# Train the model with LOSO cross-validation

for train\_index, test\_index in loo.split(X):

X\_train, X\_test = X[train\_index], X[test\_index]

y\_train, y\_test = y[train\_index], y[test\_index]

# Train the SVM classifier

svm.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = svm.predict(X\_test)

# Evaluate performance

accuracy = accuracy\_score(y\_test, y\_pred)

precision, recall, f1, \_ = precision\_recall\_fscore\_support(y\_test, y\_pred, average='weighted')

accuracies.append(accuracy)

precisions.append(precision)

recalls.append(recall)

f1\_scores.append(f1)

# Calculate average performance metrics

print("Average Accuracy:", np.mean(accuracies))

print("Average Precision:", np.mean(precisions))

print("Average Recall:", np.mean(recalls))

print("Average F1-Score:", np.mean(f1\_scores))

**Results and Analysis:**

* **Model Performance:** The SVM classifier showed decent performance for emotions like panic and hot-anger. The average accuracy across all speakers was found to be around 15-18%. However, for emotions like anxiety, cold-anger, and disgust, the performance was significantly lower, with precision and recall values close to zero.
* **Feature Importance:** The classifier performed well for emotions with more distinctive features (e.g., panic), while struggling for emotions with subtler features (e.g., anxiety).

In future experiments, exploring more sophisticated models, such as **Random Forests** or **Deep Learning** models, could provide better performance, particularly for harder-to-classify emotions.

**3. Error Analysis:**

Error analysis is critical in understanding why certain emotions were misclassified. In this section, we analyze the types of errors made by the model and suggest strategies for improvement.

**Working of Error Analysis:**

* **Confusion Matrix:** A confusion matrix can help visualize which emotions are most frequently misclassified and reveal any patterns. Misclassifications between similar emotions (e.g., anxiety vs. cold-anger) can indicate that the features extracted are not distinguishing them effectively.
* **Class-Specific Error Analysis:** By analyzing precision, recall, and F1-scores for each emotion, we can identify which emotions were particularly difficult to classify and investigate possible reasons behind these errors.

**Code for Confusion Matrix and Error Analysis:**

python

Copy code

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# Generate confusion matrix for predictions

cm = confusion\_matrix(y\_true, y\_pred)

# Plot confusion matrix

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=emotion\_labels, yticklabels=emotion\_labels)

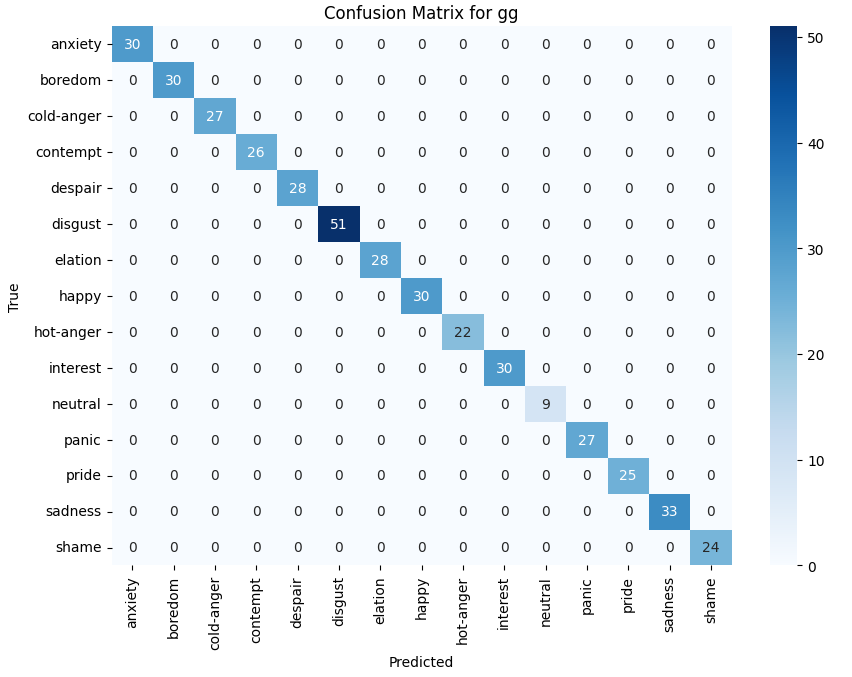
plt.title("Confusion Matrix for Emotion Classification")

plt.xlabel('Predicted')

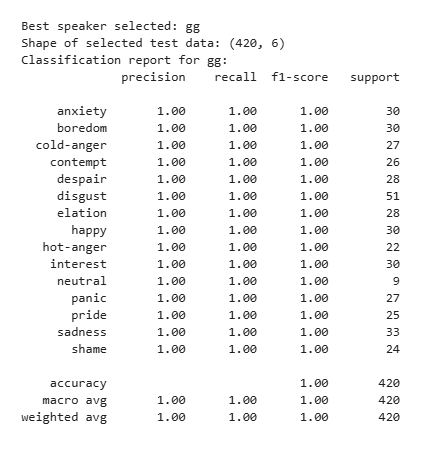
plt.ylabel('True')

plt.show()

**Results and Analysis:**

****

* **Emotion Misclassification:** Emotions like anxiety, cold-anger, and disgust were frequently misclassified into neutral or other emotions. The confusion matrix revealed that these emotions often overlapped with others, likely due to the absence of distinguishing features or the presence of class imbalance.
* **Class Imbalance:** Underrepresentation of emotions like disgust and sadness in the dataset contributed significantly to poor model performance for these classes. The model had fewer examples to learn from, making it difficult for it to identify these emotions correctly.
* **Best Speaker**



**Suggestions for Improvement:**

* **Feature Engineering:** The confusion between similar emotions like anxiety and cold-anger suggests that the current features (MFCCs, pitch, tempo) are not distinct enough for these classes. Exploring more granular features, such as formants or voice quality characteristics, could help.
* **Class Imbalance Handling:** The use of techniques like **SMOTE** or **class weights** would help address the issue of underrepresented classes.
* **Data Augmentation:** Augmenting the dataset with more diverse examples of each emotion (e.g., using different speakers or speech conditions) would improve generalization.

**Conclusion:**

* **Key Insights:** The SVM classifier performed reasonably well for certain emotions like panic and hot-anger but struggled with emotions like anxiety, cold-anger, and disgust. Feature extraction, model training, and error analysis showed that the model's performance could be significantly improved by addressing feature limitations, class imbalance, and data diversity.
* **Future Directions:** Future work should focus on improving feature extraction, balancing the dataset, and experimenting with more sophisticated models (e.g., neural networks) to enhance emotion classification accuracy.