## Emotion Recognition using Twitter dataset - Technical Report

### 1. Data Preprocessing

The preprocessing pipeline was carefully designed to handle the unique characteristics of Twitter data while preserving meaningful emotional signals.

#### 1.1 Text Cleaning and Normalization

- Converted text to lowercase while preserving significant uppercase patterns (e.g., "HAPPY" → "uppercase_happy").
- Removed URLs, @mentions, and standardized hashtags.
- Standardized punctuation patterns (e.g., "..." → "....").
- Preserved emphasis indicators like repeated punctuation ("!!", "??").
- Special handling for "LH" variations to maintain consistency.

#### 1.2 Emoji Processing

- Extracted and counted emoji occurrences.
- Converted emojis to readable text format (e.g., "😊" → "emoji_smilingface").
- Included frequency information for repeated emojis (e.g., "emoji_smilingface_3x").

#### 1.3 Stopword and Token Processing

- Removed common English stopwords while preserving emotionally significant words:
  - Kept negation words: "not", "no", "never", "against".
  - Retained intensity indicators: "very", "so", "too", "only".
  - Preserved directional words: "up", "down", "in", "out".
- Skipped very short words (≤ 2 characters) and standalone periods.

#### 1.4 Special Features

- Preserved hashtags as separate features.
- Marked words in all caps (excluding "LH") to capture emphasis.
- Combined processed text with hashtags for final representation.

#### 1.5 Spacy Lemmatization

- Used spaCy's small English model ('en_core_web_sm') for lemmatization.
- Disabled unnecessary components (parser, NER) for efficiency.

#### 1.6 Label Encoding

- Used scikit-learn's LabelEncoder to convert emotion labels to numeric values.
- Encoded 8 emotion classes: anger, anticipation, disgust, fear, joy, sadness, surprise, and trust.

- Distribution showed significant imbalance:
    - joy (35.5%)
    - anticipation (17.1%)
    - trust (14.1%)
    - sadness (13.3%)
    - disgust (9.6%)
    - fear (4.4%)
    - surprise (3.3%)
    - anger (2.7%)

## 2. Feature Engineering

Two main approaches were explored for feature extraction:

### 2.1 TF-IDF Vectorization

- Applied TF-IDF vectorization with 1000 maximum features.
- Captured word importance while accounting for frequency across documents.

### 2.2 BERT Embeddings

- Generated 384-dimensional dense vectors using 'paraphrase-MiniLM-L6-v2' model.
- Resulted in shape (1455563, 384) for training data.

## 3. Model Development and Results

Multiple models were evaluated with different architectures and approaches:

### 3.1 Neural Network Models

The best performing model was a deep neural network with the architecture:

```
- Dense(1024, activation='relu')
- BatchNormalization + Dropout(0.3)
- Dense(512, activation='relu')
- BatchNormalization + Dropout(0.3)
- Dense(256, activation='relu')
- BatchNormalization + Dropout(0.2)
- Dense(128, activation='relu')
```

```
- BatchNormalization + Dropout(0.2)

- Dense(64, activation='relu')

- BatchNormalization + Dropout(0.1)

- Dense(8, activation='softmax')
```

Key features:

- Batch size: 32
- Learning rate: 0.001
- Early stopping with patience=5
- Learning rate reduction on plateau
- Training accuracy: 54.145%
- Public test accuracy: 43.554%

### 3.2 Other Approaches Tried

1. Random Forest + TF-IDF:
    - Training accuracy: 93.29%
    - Test accuracy: 36.54%
    - Showed signs of overfitting
2. Random Forest + BERT:
    - Training accuracy: 99.42%
    - Test accuracy: 32.23%
    - Severe overfitting
3. Neural Network + TF-IDF:
    - Training accuracy: 52.00%
    - Test accuracy: 38.51%
4. Simpler Neural Networks:
    - Various architectures with fewer layers
    - Generally performed worse than the final model

### 3.3 Failed Experiments

1. Class weight balancing:
    - Attempted to address class imbalance.

- Led to decreased performance (35.24% public test accuracy).

2. Dimensionality reduction:
   - Tried TruncatedSVD for feature reduction.
   - Selected top 50 features using mutual information.
   - Resulted in poor public test performance (<30%).

## 4. Key Insights

1. Model Complexity:
   - Deep neural networks with proper regularization performed better than simpler models.
   - Batch normalization and dropout were crucial for preventing overfitting.
2. Feature Representation:
   - BERT embeddings generally outperformed TF-IDF features.
   - Preserving emotional signals (emojis, emphasis, hashtags) was important.
3. Class Imbalance:
   - Traditional methods like class weights didn't help.
   - The model performed better when trained on raw data distributions.
4. Preprocessing Impact:
   - Careful preprocessing to preserve emotional signals was crucial.
   - Keeping uppercase patterns and emoji information improved performance.

The final model achieved a public test accuracy of 43.554% and private test accuracy of 42.115%, showing reasonable performance on this challenging multi-class emotion classification task.