Lab2 Homework Report

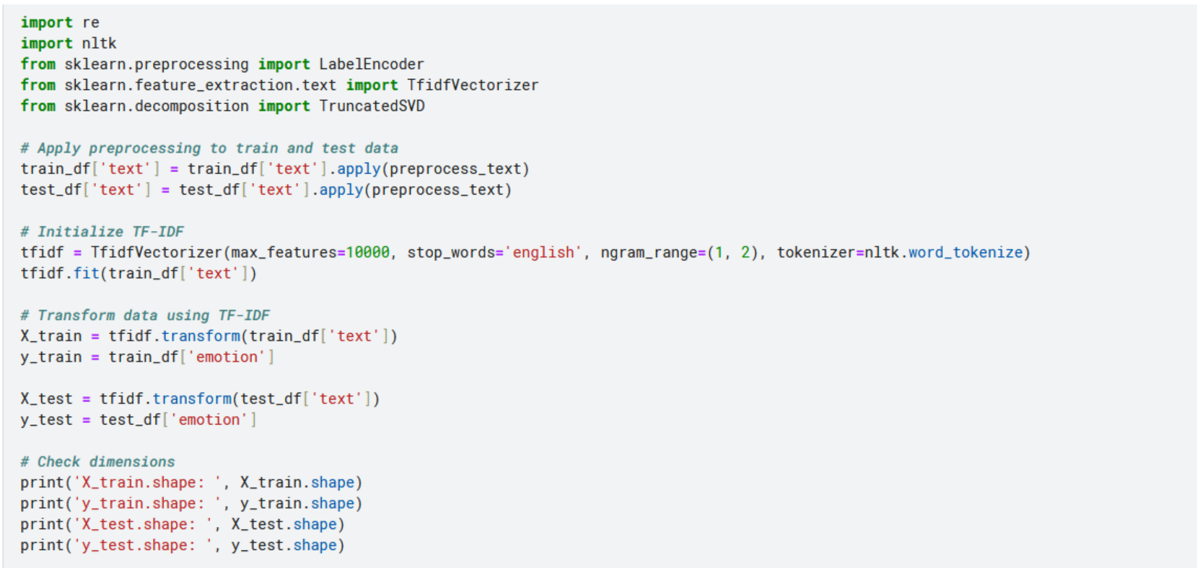
Name: 蘇立光

Student ID: 110000162

A. Overview

The model I used for this assignment comes from the master part. I used the DNN (Dense Neural Network) model with TF-IDF weights.

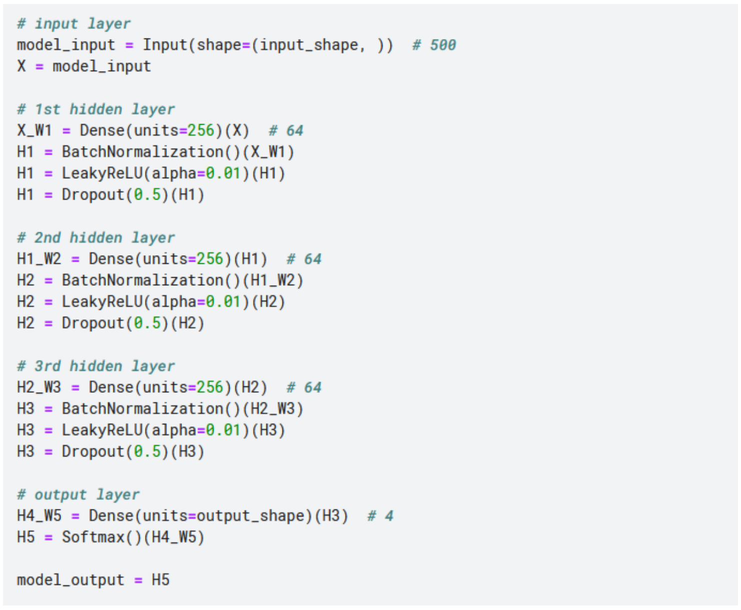
This is the TF-IDF vectorization



```python
import re
import nltk
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD

# Apply preprocessing to train and test data
train_df['text'] = train_df['text'].apply(preprocess_text)
test_df['text'] = test_df['text'].apply(preprocess_text)

# Initialize TF-IDF
tfidf = TfidfVectorizer(max_features=10000, stop_words='english', ngram_range=(1, 2), tokenizer=nltk.word_tokenize)
tfidf.fit(train_df['text'])

# Transform data using TF-IDF
X_train = tfidf.transform(train_df['text'])
y_train = train_df['emotion']

X_test = tfidf.transform(test_df['text'])
y_test = test_df['emotion']

# Check dimensions
print('X_train.shape: ', X_train.shape)
print('y_train.shape: ', y_train.shape)
print('X_test.shape: ', X_test.shape)
print('y_test.shape: ', y_test.shape)
```

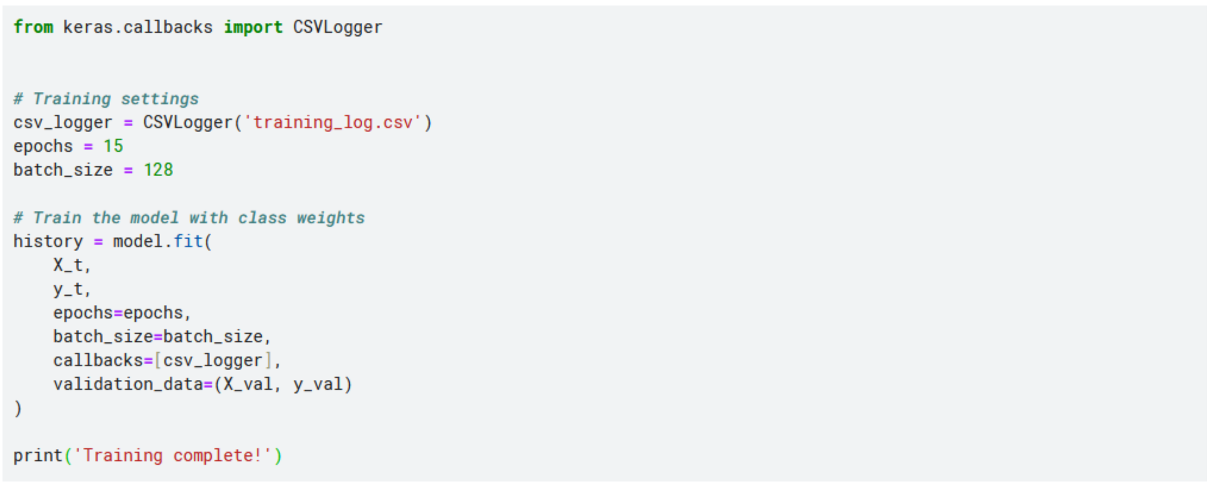
I used 10000 features and removing common English stop words.

Here is the DNN model specification



```python
# input layer
model_input = Input(shape=(input_shape, ))  # 500
X = model_input

# 1st hidden layer
X_W1 = Dense(units=256)(X)  # 64
H1 = BatchNormalization()(X_W1)
H1 = LeakyReLU(alpha=0.01)(H1)
H1 = Dropout(0.5)(H1)

# 2nd hidden layer
H1_W2 = Dense(units=256)(H1)  # 64
H2 = BatchNormalization()(H1_W2)
H2 = LeakyReLU(alpha=0.01)(H2)
H2 = Dropout(0.5)(H2)

# 3rd hidden layer
H2_W3 = Dense(units=256)(H2)  # 64
H3 = BatchNormalization()(H2_W3)
H3 = LeakyReLU(alpha=0.01)(H3)
H3 = Dropout(0.5)(H3)

# output layer
H4_W5 = Dense(units=output_shape)(H3)  # 4
H5 = Softmax()(H4_W5)

model_output = H5
```

I used a DNN model with 3 hidden layers, each layer is 256 in size. Additionally, I used Batch Normalization and Dropout for optimization.



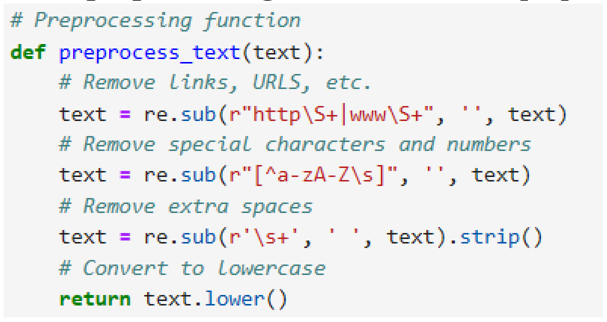
```python
from keras.callbacks import CSVLogger


# Training settings
csv_logger = CSVLogger('training_log.csv')
epochs = 15
batch_size = 128

# Train the model with class weights
history = model.fit(
    X_t,
    y_t,
    epochs=epochs,
    batch_size=batch_size,
    callbacks=[csv_logger],
    validation_data=(X_val, y_val)
)

print('Training complete!')
```

For the model fit, I trained it for 15 epochs and with 128 batch size (this just makes it faster).

B. Development steps

Here are some of the things I tried:

- **Increasing TF-IDF features**: Without any preprocessing, the F1 score in the competition is around 0.39. This is still using only 1000 TF-IDF features. I gradually increase the TF-IDF features until reaching 10000 before I see severe overfitting. (Increase to around 0.025 – 0.03)
- **Some preprocessing**: I did some minor preprocessing to the text data.

  ```python
  # Preprocessing function
  def preprocess_text(text):
      # Remove links, URLS, etc.
      text = re.sub(r"http\S+|www\S+", '', text)
      # Remove special characters and numbers
      text = re.sub(r"[^a-zA-Z\s]", '', text)
      # Remove extra spaces
      text = re.sub(r'\s+', ' ', text).strip()
      # Convert to lowercase
      return text.lower()
  ```

  The things I did are
  - Remove links
  - Remove special characters and numbers
  - Remove extra spaces
  - Convert all the text to lowercase

  This preprocessing doesn't really increase the F1 score much (around 0.01 increase). The main contributor to the minor increase in F1 score is the lowercase
- **Modification to the DNN model:** I also change the DNN model.

The things I did are

- Increasing the number of hidden layers
- Increasing the size of each hidden layer
- Using Dropout and Batch Normalization

Increasing the number of hidden layers and each of their size increase the F1 score a bit (only like 0.001). This also include Dropout and Batch Normalization. To clarify,

- Dropout is a regularization technique that drop a fraction of the neurons randomly, this is to decrease the effect of memorization of the NN
- Batch Normalization normalize the value of inputs in each layer of each mini-batch. This is to decrease instability which is exploding gradient or vanishing gradient

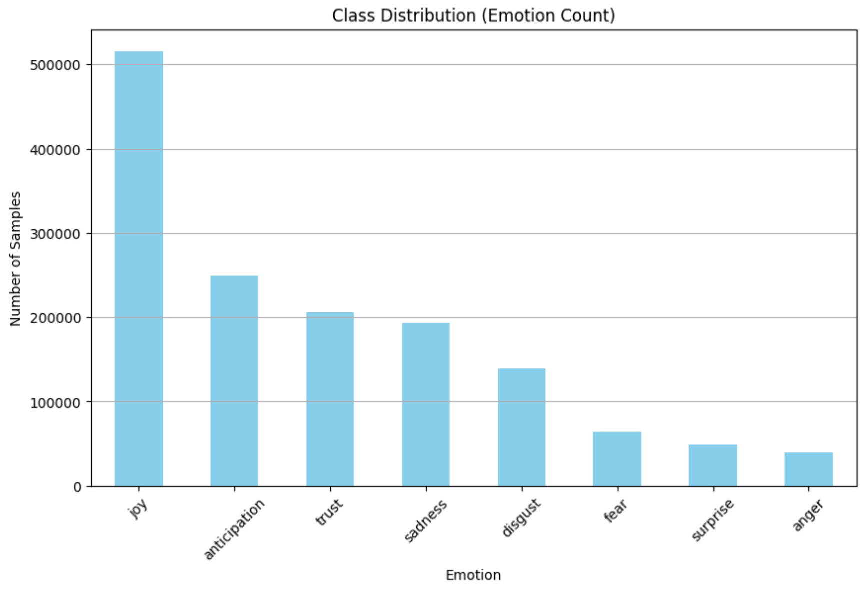
C. Data Exploration

The first thing I noticed instantly was overfitting. Using the 80:20 training-validation split in the model, the validation accuracy stagnates then decreases after 4-5 epochs, this is a sign of overfitting.

From my experience from the doing the final project, class imbalance is a major issue.

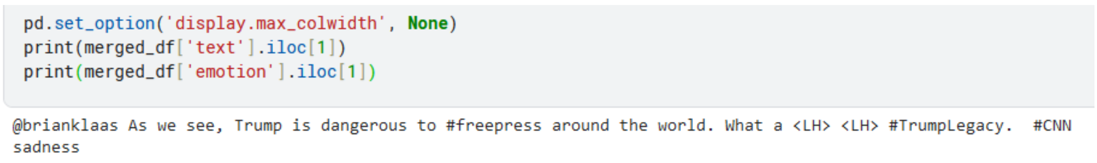
```python
import matplotlib.pyplot as plt

# Count the number of occurrences of each emotion
class_counts = train_df['emotion'].value_counts()

# Plot the class distribution
plt.figure(figsize=(10, 6))
class_counts.plot(kind='bar', color='skyblue')
plt.title('Class Distribution (Emotion Count)')
plt.xlabel('Emotion')
plt.ylabel('Number of Samples')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.show()
```

Using this code, I check the data's class distribution. Here is the result



The class imbalance is very visible. Joy is taking over one third of the data. The data is clearly biased toward the class Joy.

I also read some of the text and compare them with their corresponding assigned emotions.



```
pd.set_option('display.max_colwidth', None)
print(merged_df['text'].iloc[1])
print(merged_df['emotion'].iloc[1])
```

```
@brianklaas As we see, Trump is dangerous to #freepress around the world. What a <LH> <LH> #TrumpLegacy.  #CNN
sadness
```

Sometimes, some of the assigned emotion can be a bit wrong. Such as the example above. In my opinion the text above should be classified as anger instead of sadness.

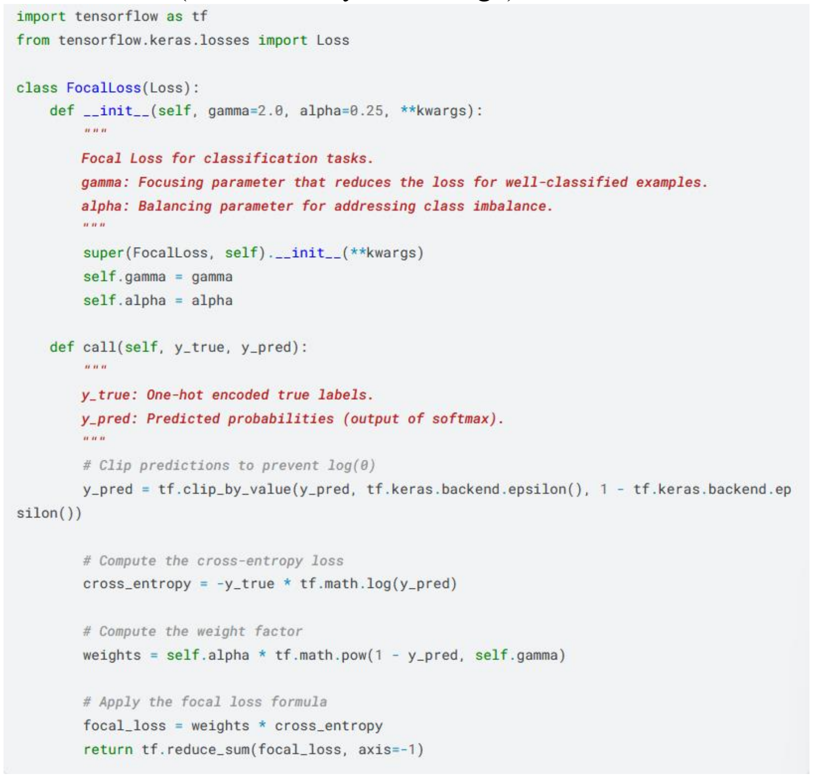
There are other text which are sarcastic text, but the assigned emotion is not reflecting the actual underlying meaning of the sarcastic text.

D. Unsuccessful attempt

These are the other attempts to increase F1 score I tried.

- Class weight
  In the previous point, I mentioned class imbalance. So, I made the sensible choice to use class weight. This seems to make things worse. I put more weight into the rare classes (such as anger, fear), but this just make it too biased to the rare classes

- Changing loss function
  I tried some other way to address the overfitting problem. One thing is to use Focal Loss for the loss calculation. Since keras doesn't have a predefined focal loss function (at least to my knowledge), I used a defined function online



```
import tensorflow as tf
from tensorflow.keras.losses import Loss

class FocalLoss(Loss):
    def __init__(self, gamma=2.0, alpha=0.25, **kwargs):
        """
        Focal Loss for classification tasks.
        gamma: Focusing parameter that reduces the loss for well-classified examples.
        alpha: Balancing parameter for addressing class imbalance.
        """
        super(FocalLoss, self).__init__(**kwargs)
        self.gamma = gamma
        self.alpha = alpha

    def call(self, y_true, y_pred):
        """
        y_true: One-hot encoded true labels.
        y_pred: Predicted probabilities (output of softmax).
        """
        # Clip predictions to prevent log(0)
        y_pred = tf.clip_by_value(y_pred, tf.keras.backend.epsilon(), 1 - tf.keras.backend.ep
silon())

        # Compute the cross-entropy loss
        cross_entropy = -y_true * tf.math.log(y_pred)

        # Compute the weight factor
        weights = self.alpha * tf.math.pow(1 - y_pred, self.gamma)

        # Apply the focal loss formula
        focal_loss = weights * cross_entropy
        return tf.reduce_sum(focal_loss, axis=-1)
```

My idea for using Focal Loss here is to dynamically assign the class weight to address the class imbalance. The class weight from the previous point, I assign it manually. Unfortunately, the F1 score is lower with Focal Loss than the F1 score using cross entropy

- Using WordVectorizer
  I tried using CountVectorizer instead of TF-IDF vectorizer. This decreases the accuracy instead

- Using other models
  From my experience with the final project, I tried to use LightBGM, CatBoost, and XGBoost models. These 3 models are not suitable for this dataset. It takes way too long (4 hours each), and the result is just bad.