Emotion Detection

emotion detection	
Purpose I have developed a keen interest in natural language processing, and this course has significantly enhanced my understanding of machine learning. Throughout this course, our primary focus has been on utilizing data in integer or float formats. However, I am now eager to explore a different approach by using text as data in machine learning models. This exploration is also aimed at delving deeper into the field of natural language processing. My choice for a project is emotion detection. Emotions are inherently challenging to quantify, and it would be fascinating to observe how a machine learning model can predict them	**Result** The accuracies of each model are as follows: Naive Bayes achieved an accuracy of 0.77, Linear Regression reached 0.88, and the Convolutional Neural Network (CNN) performed with an accuracy of 0.9112.
Procedures I utilized the public dataset from the paper titled 'CARER: Contextualized Affect Representations for Emotion Recognition,' which comprises 416,809 texts, each annotated with corresponding emotions like anger, fear, joy, love, sadness, and surprise. Subsequently, I experimented with three distinct machine learning models: Linear Regression, Naive Bayes, and Convolutional Neural Network (CNN). I evaluated their performance using relevant metrics. Finally, I developed a simple Python program that allows users to experiment with these three models.	**Analysis** The performance of the CNN was the best among the three models, followed by Linear Regression and Naive Bayes. Within each category, 'surprise' had the lowest performance, which can be attributed to it having the smallest number of texts in the dataset.

Report

Visualization

I initially examined the paper titled 'CARER: Contextualized Affect Representations for Emotion Recognition.'[link] The techniques they employed for emotion detection were quite complex, so I chose not to follow their approach. Instead, I was interested in evaluating the performance of standard, in-class machine learning models on this dataset. The authors of the paper had made the dataset they used available at https://github.com/dair-ai/emotion_dataset. This dataset, sourced from real-world Twitter data, has been preprocessed and is ready for use. Then, I downloaded the dataset and then tried to visualize it. All code that I wrote myself will be highlighted in yellow.

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_pickle('merged_training.pkl')
```

```python
df.head(15)
```

	text	emotions
27383	i feel awful about it too because it s my job ...	sadness
110083	im alone i feel awful	sadness
140764	ive probably mentioned this before but i reall...	joy
100071	i was feeling a little low few days back	sadness
2837	i beleive that i am much more sensitive to oth...	love
18231	i find myself frustrated with christians becau...	love
10714	i am one of those people who feels like going ...	joy
35177	i feel especially pleased about this as this h...	joy
122177	i was struggling with these awful feelings and...	joy
26723	i feel so enraged but helpless at the same time	anger
41979	i said feeling a bit rebellious	anger
2046	i also feel disillusioned that someone who cla...	sadness
98659	i mean is on this stupid trip of making the gr...	joy
50434	i woke up feeling particularly vile tried to i...	anger
9280	i could feel the vile moth burrowing its way i...	anger

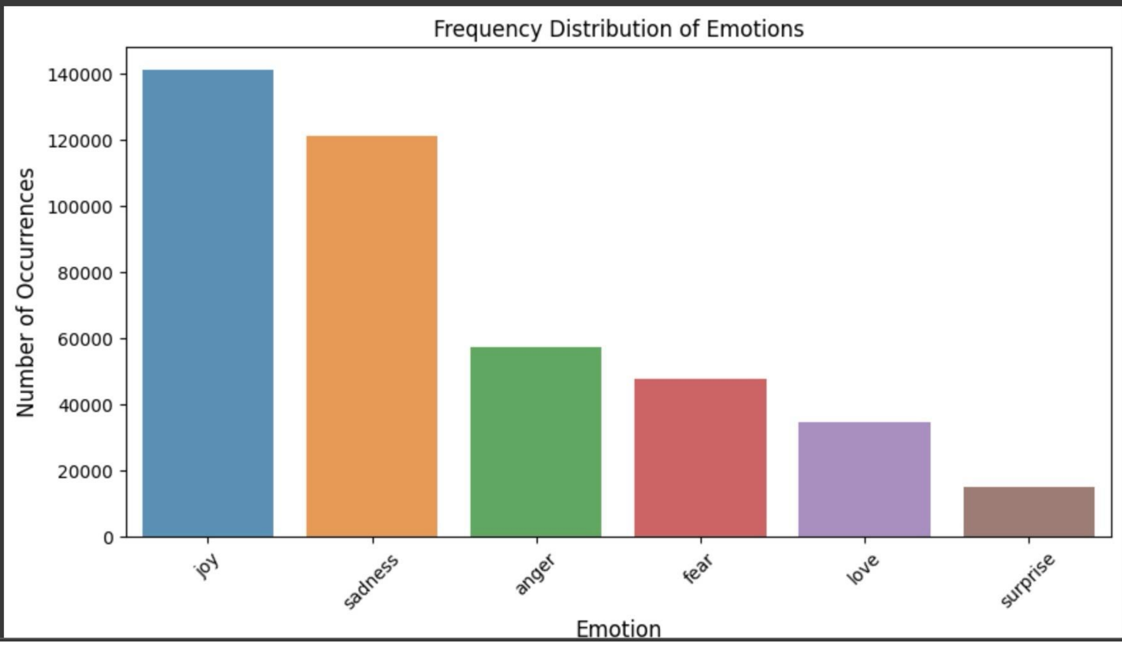
```python
len(df.index)
```

```
416809
```

```python
# Display the distribution of emotions
emotion_counts = df['emotions'].value_counts()
print(emotion_counts)
```

```
joy         141067
sadness     121187
anger        57317
fear         47712
love         34554
surprise     14972
Name: emotions, dtype: int64
```

```
[ ] # Plotting the distribution of emotions
    plt.figure(figsize=(10, 5))
    sns.barplot(x=emotion_counts.index, y=emotion_counts.values, alpha=0.8)
    plt.title('Frequency Distribution of Emotions')
    plt.ylabel('Number of Occurrences', fontsize=12)
    plt.xlabel('Emotion', fontsize=12)
    plt.xticks(rotation=45)
    plt.show()
```



After analyzing the dataset, I found that it comprises 416,809 texts, each associated with a corresponding emotion. The emotions labeled in the dataset include joy, sadness, anger, fear, love, and surprise. The distribution chart of the dataset reveals a significant skewness towards emotions like joy and sadness, with a notably smaller representation of texts expressing surprise.

```
] min_emotion_count = emotion_counts.min()
  min_emotion = emotion_counts.idxmin()
  print(min_emotion, min_emotion_count)

  surprise 14972
```

The code provided above demonstrates how to identify the emotion that is least represented in the dataset.

<u>Text Vectorization</u>

One aspect that distinguishes text in machine learning from other forms is that machine learning models cannot directly interpret text; they can only process numbers. Therefore, to enable these models to understand the dataset, it is necessary to first convert the text into numerical form. There are several techniques available for this conversion such as bag-of-words, tf-idf, and word embeddings[link]. In this project, I have chosen to use the TF-IDF approach because the dataset is large, and the bag-of-words model simply cannot be applied due to limited computational resources. Additionally, the paper states that TF-IDF works well with their machine learning model[link]. TF-IDF, which stands for Term Frequency-Inverse Document Frequency, is a numerical statistic used in text mining and information retrieval. It reflects how important a word is to a document in a collection or corpus. The TF-IDF value increases proportionally with the number of times a word appears in the document but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general [link].

```
# import required module
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english')

# get tf-df values
X_tfidf = tfidf_vectorizer.fit_transform(df['text'])
```

```
y = df['emotions']
```

```
X_tfidf
```

```
<416809x74964 sparse matrix of type '<class 'numpy.float64'>'
	with 3396200 stored elements in Compressed Sparse Row format>
```

The code above demonstrates the use of TF-IDF for text vectorization, following an example from this tutorial [link].

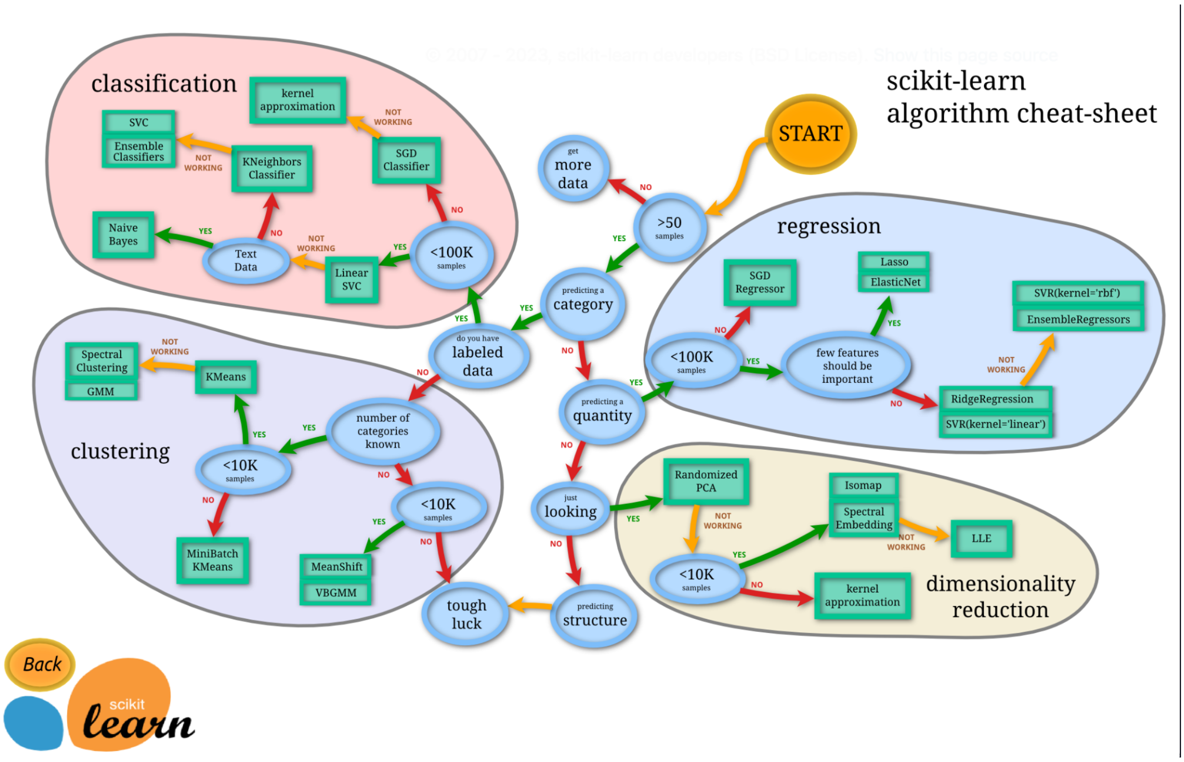
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42, stratify=y)
```

Then, I divided the dataset into training and test sets, setting the test size to 0.2. Additionally, I employed stratification to ensure proportional sampling of the dataset, which is important given the imbalance in the data [link].

Model training and testing

I selected three machine learning models: Naive Bayes, Logistic Regression, and CNN. Initially, I researched which models to use and referred to the scikit-learn algorithm cheat-sheet, which can be found at the following link [link]. As suggested in the scikit-learn algorithm cheat-sheet, I followed the guidelines and chose the Naive Bayes classifier, as it is well-suited for text data. Regarding Logistic Regression, I chose it based on my learning in class, understanding that it can be applied to classification tasks. Lastly, I selected CNN (Convolutional Neural Network) because it is a more advanced model.



Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
naive_bayes_classifier = MultinomialNB()

# Train the classifier
naive_bayes_classifier.fit(X_train, y_train)

# Predict on the test data
y_pred = naive_bayes_classifier.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

As I researched on the internet, it appeared that MultinomialNB is better suited than regular Naive Bayes for this purpose. Therefore, I chose to use MultinomialNB. The code I wrote was inspired by a blog post, which can be found here [link].

```
Accuracy: 0.7666682661164559
              precision    recall  f1-score   support

       anger       0.93      0.64      0.76     11463
        fear       0.90      0.48      0.63      9542
         joy       0.71      0.97      0.82     28214
        love       0.95      0.24      0.38      6911
     sadness       0.76      0.94      0.84     24238
    surprise       0.97      0.08      0.14      2994

    accuracy                           0.77     83362
   macro avg       0.87      0.56      0.59     83362
weighted avg       0.81      0.77      0.73     83362
```

The performance of MultinomialNB is illustrated above. Overall, the model achieved an accuracy of 0.76. The F1-scores for most categories are quite decent (above 0.5) except for 'love' and 'surprise'. I suspect this might be due to the dataset containing a smaller amount of data for these two emotions.

```
import joblib

# Save the model
joblib.dump(naive_bayes_classifier, 'naive_bayes_model.pkl')

# Save the vectorizer
joblib.dump(tfidf_vectorizer, 'tfidf_vectorizer.pkl')

['tfidf_vectorizer.pkl']
```

Then, I saved the model for future use in the application phase. I used joblib to save the model and plan to use it for saving the other models as well. The code I wrote is based on a tutorial that explains how to save models, which can be found here [link].

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

# Initialize the Logistic Regression model
# Using 'liblinear' solver for binary classification and 'saga' for multiclass
logreg = LogisticRegression(solver='saga', max_iter=1000,random_state=42)

# Train the model
logreg.fit(X_train, y_train)

# Predict on the test data
y_pred_logreg = logreg.predict(X_test)

# Evaluate the model
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
report_logreg = classification_report(y_test, y_pred_logreg)
```

The code above demonstrates how to train, test, and measure the performance of logistic regression. This code is inspired by the material from our class. Since we are dealing with a multiclass scenario here, I specified the 'saga' solver, which is better suited for large datasets and multinomial classes [link].

```
accuracy_logreg
```

```
0.8896619562870373
```

```
print(report_logreg)
```

```
              precision    recall  f1-score   support

       anger       0.89      0.90      0.90     11463
        fear       0.84      0.83      0.84      9542
         joy       0.90      0.93      0.91     28214
        love       0.79      0.74      0.77      6911
     sadness       0.94      0.93      0.93     24238
    surprise       0.75      0.69      0.72      2994

    accuracy                           0.89     83362
   macro avg       0.85      0.84      0.84     83362
weighted avg       0.89      0.89      0.89     83362
```

As for performance, the overall accuracy is 0.88, which surpasses the Naive Bayes approach. Additionally, in categories with limited data, such as 'love' and 'surprise,' the performance remains satisfactory, with accuracies higher than 0.5.

```
joblib.dump(logreg, 'logistic_regression_model.joblib')
```

```
['logistic_regression_model.joblib']
```

I also saved the logistic regression model for future use in the application phase.

CNN

```python
from keras.models import Sequential
from keras.layers import Embedding, Conv1D, MaxPooling1D, Flatten, Dense
from keras_preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split
import tensorflow.keras.backend as K

K.clear_session()
# Tokenize and pad sequences
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(df['text'])
sequences = tokenizer.texts_to_sequences(df['text'])
X = pad_sequences(sequences, maxlen=100)

# Prepare the labels
y = pd.get_dummies(df['emotions']).values

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the CNN model
model = Sequential()
model.add(Embedding(input_dim=5000, output_dim=128, input_length=100))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(10, activation='relu'))
model.add(Dense(y.shape[1], activation='softmax'))
```

The code demonstrates my implementation of a CNN, modified from our class material. One key difference is the tokenizer, as there's a specific tokenizer for CNNs in Keras, as implemented in the above code. This tokenizer functions similarly to TF-IDF, transforming text into numerical format to be fed into the CNN.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 128)          640000

 conv1d (Conv1D)             (None, 100, 32)           12320

 max_pooling1d (MaxPooling1  (None, 50, 32)            0
 D)

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 10)                16010

 dense_1 (Dense)             (None, 6)                 66

=================================================================
Total params: 668396 (2.55 MB)
Trainable params: 668396 (2.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
from tensorflow.keras import optimizers

opt = optimizers.Adam(learning_rate=0.001)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

The code above displays the model summary and compiles it with an optimizer.

```
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

# Model checkpoint to save the model after every epoch
checkpoint = ModelCheckpoint("model.h5", monitor='val_loss', verbose=1, save_best_only=True, mode='min')

# Early stopping to stop training when the validation loss does not decrease for 5 epochs
early_stopping = EarlyStopping(monitor='val_loss', mode='min', patience=5, verbose=1)

# Reduce learning rate when the validation loss plateaus
reduce_lr = ReduceLROnPlateau(monitor='val_loss', mode='min', patience=3, factor=0.2, min_lr=0.00001, verbose=1)

# Including these callbacks in the model's fit method
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=64, callbacks=[checkpoint, early_stopping, reduce_lr])
```

I also added three Keras callbacks to help optimize the CNN model. These include checkpoint, early stopping, and reducing the learning rate [link].

```python
import pickle

# Saving the tokenizer
with open('tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)

model.save('CNN.h5')
```

Then, I saved the tokenizer and CNN model for future use in the application phase.

Application

```python
import joblib
import keras
from keras_preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
import numpy as np
import pickle
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score

def preprocess_text(text,tokenizer):
    sequences = tokenizer.texts_to_sequences([text])
    padded_sequences = pad_sequences(sequences, maxlen=100)
    return padded_sequences
def main():
    # Load necessary model
    vectorizer = joblib.load('tfidf_vectorizer.pkl')
    naive_bayes_model = joblib.load('naive_bayes_model.pkl')
    logreg = joblib.load('logistic_regression_model.joblib')
    CNN = keras.models.load_model('CNN.h5')

    with open('tokenizer.pickle', 'rb') as handle:
        tokenizer = pickle.load(handle)

    index_to_emotion = {
    0: 'anger',
    1: 'fear',
    2: 'joy',
    3: 'love',
    4: 'sadness',
    5: 'surprise'
      }

    text=input("Enter your text here:")

    # preprocess text
    processed_text = preprocess_text(text,tokenizer)
    text_vector = vectorizer.transform([text])

    prediction_naive_bayes = naive_bayes_model.predict(text_vector)
    probabilities_naive_bayes = naive_bayes_model.predict_proba(text_vector)
```

```
    prediction_logreg = logreg.predict(text_vector)
    probabilities_logreg = logreg.predict_proba(text_vector)

    prediction_CNN = CNN.predict(processed_text)
    emotion_index = np.argmax(prediction_CNN)
    emotion_label = index_to_emotion[emotion_index]

    print("\nPredictions and Probabilities:\n")
    print("Naive Bayes Prediction: ", prediction_naive_bayes[0])
    print("Naive Bayes Probabilities: ")
    for i in range(len(probabilities_naive_bayes[0])):
        print(f"{index_to_emotion[i]}:{probabilities_naive_bayes[0][i]} ")
    print("\nLogistic Regression Prediction: ", prediction_logreg[0])
    for i in range(len(probabilities_logreg[0])):
        print(f"{index_to_emotion[i]}:{probabilities_logreg[0][i]} ")
    print("\nCNN Prediction (Emotion): ", emotion_label)
    for i in range(len(prediction_CNN[0])):
        print(f"{index_to_emotion[i]}:{prediction_CNN[0][i]} ")

if __name__ == "__main__":
    main()
```

This program asks users to input text, then predicts the emotion within the text and provides the probability for each label. Note that here I use the 'predict_proba' method to calculate the probability for each label [link].

```
Enter your text here:I slammed my fist in anger, but then burst into laughter, realizing the absurdity of it all.
1/1 [==============================] - 0s 42ms/step

Predictions and Probabilities:

Naive Bayes Prediction:  joy
Naive Bayes Probabilities:
anger:0.2190863061281364
fear:0.1388256211466991
joy:0.26072114153468845
love:0.11434620221122123
sadness:0.24181941676627788
surprise:0.02520131221297537

Logistic Regression Prediction:  anger
anger:0.4017210688862057
fear:0.12184924117849924
joy:0.23832651889324377
love:0.06885038501146967
sadness:0.1315110941870479
surprise:0.03774169184353367

CNN Prediction (Emotion):  anger
anger:0.5201478004455566
fear:0.33720672130584717
joy:0.0031150185968726873
love:0.00024720156216062605
sadness:0.13907475769519806
surprise:0.0002085332671413198
```

Example input/output shows above.

References

Brownlee, J. (n.d.). How to Prepare Text Data for Machine Learning with scikit-learn. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/

Saravia, E., Liu, H.-C. T., Huang, Y.-H., Wu, J., & Chen, Y.-S. (2018). CARER: Contextualized Affect Representations for Emotion Recognition. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. ACL Anthology. https://aclanthology.org/D18-1404/

MonkeyLearn Team. (n.d.). What is TF-IDF? MonkeyLearn Blog. Retrieved from https://monkeylearn.com/blog/what-is-tf-idf/

freeCodeCamp. (n.d.). What is Stratified Random Sampling: Definition and Python Example. Retrieved from https://www.freecodecamp.org/news/what-is-stratified-random-sampling-definition-and-python-example/

scikit-learn. (n.d.). Choosing the right estimator. scikit-learn. Retrieved from https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

Brownlee, J. (n.d.). How to Save and Load Your Machine Learning Models in Python. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/

Arnav, R. (n.d.). scikit-learn solvers explained. Medium. Retrieved from https://medium.com/@arnavr/scikit-learn-solvers-explained

Brownlee, J. (n.d.). How to Prepare Text Data for Deep Learning with Keras. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/

Educative. (n.d.). Difference between predict and predict_proba in sklearn. Retrieved from https://www.educative.io/answers/difference-between-predict-and-predictproba-in-sklearn

project

December 23, 2023

0.1 Emotion Detection

0.2 Loading dataset

[20]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_pickle('merged_training.pkl')
```

[21]:
```
df.head(15)
```

[21]:
```
                                                     text emotions
27383   i feel awful about it too because it s my job …  sadness
110083                              im alone i feel awful  sadness
140764  ive probably mentioned this before but i reall…      joy
100071           i was feeling a little low few days back  sadness
2837    i beleive that i am much more sensitive to oth…     love
18231   i find myself frustrated with christians becau…     love
10714   i am one of those people who feels like going …      joy
35177   i feel especially pleased about this as this h…      joy
122177  i was struggling with these awful feelings and…      joy
26723     i feel so enraged but helpless at the same time    anger
41979                          i said feeling a bit rebellious    anger
2046    i also feel disillusioned that someone who cla…  sadness
98659   i mean is on this stupid trip of making the gr…      joy
50434   i woke up feeling particularly vile tried to i…    anger
9280    i could feel the vile moth burrowing its way i…    anger
```

[22]:
```
len(df.index)
```

[22]:
```
416809
```

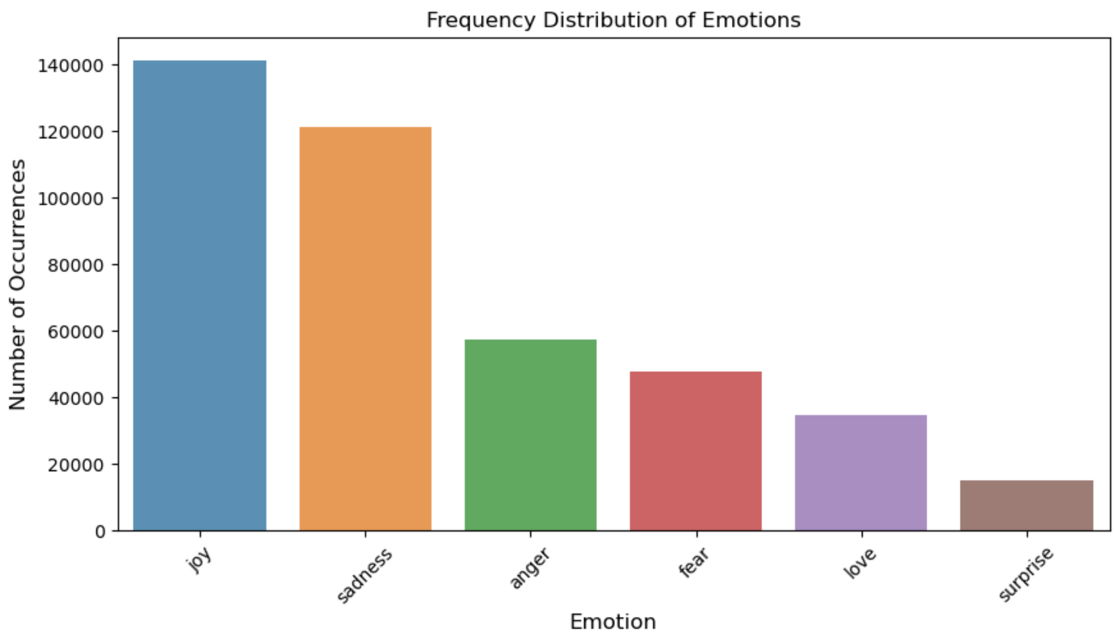
[23]:
```
# Display the distribution of emotions
emotion_counts = df['emotions'].value_counts()
print(emotion_counts)
```

```
joy        141067
sadness    121187
anger       57317
```

```
fear          47712
love          34554
surprise      14972
Name: emotions, dtype: int64
```

[24]:

```
# Plotting the distribution of emotions
plt.figure(figsize=(10, 5))
sns.barplot(x=emotion_counts.index, y=emotion_counts.values, alpha=0.8)
plt.title('Frequency Distribution of Emotions')
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Emotion', fontsize=12)
plt.xticks(rotation=45)
plt.show()
```



[25]:

```
min_emotion_count = emotion_counts.min()
min_emotion = emotion_counts.idxmin()
print(min_emotion, min_emotion_count)
```

```
surprise 14972
```

the dataset is very skew. it should be split equally among the rest. we can Stratified Split(cite) to split dataset equally.

but first we should convert text to number so the ml model can process it

[26]:

```
# import required module
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tfidf_vectorizer = TfidfVectorizer(stop_words='english')

# get tf-df values
X_tfidf = tfidf_vectorizer.fit_transform(df['text'])
```

[27]:
```
y = df['emotions']
```

[28]:
```
X_tfidf
```

[28]:
```
<416809x74964 sparse matrix of type '<class 'numpy.float64'>'
	with 3396200 stored elements in Compressed Sparse Row format>
```

[29]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42, stratify=y)
```

1 Model

1.1 Naive Bayes

[30]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
naive_bayes_classifier = MultinomialNB()

# Train the classifier
naive_bayes_classifier.fit(X_train, y_train)

# Predict on the test data
y_pred = naive_bayes_classifier.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.7666682661164559
              precision    recall  f1-score   support

       anger       0.93      0.64      0.76     11463
        fear       0.90      0.48      0.63      9542
         joy       0.71      0.97      0.82     28214
        love       0.95      0.24      0.38      6911
     sadness       0.76      0.94      0.84     24238
    surprise       0.97      0.08      0.14      2994

    accuracy                           0.77     83362
   macro avg       0.87      0.56      0.59     83362
```

```
weighted avg       0.81      0.77      0.73     83362
```

[31]:
```python
import joblib

# Save the model
joblib.dump(naive_bayes_classifier, 'naive_bayes_model.pkl')

# Save the vectorizer
joblib.dump(tfidf_vectorizer, 'tfidf_vectorizer.pkl')
```

[31]:
```
['tfidf_vectorizer.pkl']
```

1.2 Linear Regression

[32]:
```python
from sklearn.linear_model import LogisticRegression

# Initialize the Logistic Regression model
# Using 'liblinear' solver for binary classification and 'saga' for multiclass
logreg = LogisticRegression(solver='saga', max_iter=1000,random_state=42)

# Train the model
logreg.fit(X_train, y_train)

# Predict on the test data
y_pred_logreg = logreg.predict(X_test)

# Evaluate the model
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
report_logreg = classification_report(y_test, y_pred_logreg)
```

[33]:
```python
accuracy_logreg
```

[33]:
```
0.8896619562870373
```

[34]:
```python
print(report_logreg)
```

```
              precision    recall  f1-score   support

       anger       0.89      0.90      0.90     11463
        fear       0.84      0.83      0.84      9542
         joy       0.90      0.93      0.91     28214
        love       0.79      0.74      0.77      6911
     sadness       0.94      0.93      0.93     24238
    surprise       0.75      0.69      0.72      2994

    accuracy                           0.89     83362
   macro avg       0.85      0.84      0.84     83362
```

```
weighted avg       0.89      0.89      0.89     83362
```

[35]:
```
joblib.dump(logreg, 'logistic_regression_model.joblib')
```

[35]:
```
['logistic_regression_model.joblib']
```

1.3 CNN

[36]:
```
from keras.models import Sequential
from keras.layers import Embedding, Conv1D, MaxPooling1D, Flatten, Dense
from keras_preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split
import tensorflow.keras.backend as K

K.clear_session()
# Tokenize and pad sequences
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(df['text'])
sequences = tokenizer.texts_to_sequences(df['text'])
X = pad_sequences(sequences, maxlen=100)

# Prepare the labels
y = pd.get_dummies(df['emotions']).values

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Build the CNN model
model = Sequential()
model.add(Embedding(input_dim=5000, output_dim=128, input_length=100))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(10, activation='relu'))
model.add(Dense(y.shape[1], activation='softmax'))
```

[37]:
```
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 128)          640000


 conv1d (Conv1D)             (None, 100, 32)           12320
```

```
 max_pooling1d (MaxPooling1D  (None, 50, 32)           0
 )

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 10)                16010

 dense_1 (Dense)             (None, 6)                 66

=================================================================
Total params: 668,396
Trainable params: 668,396
Non-trainable params: 0
_________________________________________________________________
```

[38]:
```python
from tensorflow.keras import optimizers

opt = optimizers.Adam(learning_rate=0.001)
model.compile(loss='categorical_crossentropy', optimizer=opt,␣
 ↪metrics=['accuracy'])
```

[39]:
```python
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

# Model checkpoint to save the model after every epoch
checkpoint = ModelCheckpoint("model.h5", monitor='val_loss', verbose=1,␣
 ↪save_best_only=True, mode='min')

# Early stopping to stop training when the validation loss does not decrease␣
 ↪for 5 epochs
early_stopping = EarlyStopping(monitor='val_loss', mode='min', patience=5,␣
 ↪verbose=1)

# Reduce learning rate when the validation loss plateaus
reduce_lr = ReduceLROnPlateau(monitor='val_loss', mode='min', patience=3,␣
 ↪factor=0.2, min_lr=0.00001, verbose=1)

# Including these callbacks in the model's fit method
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
 ↪epochs=10, batch_size=64, callbacks=[checkpoint, early_stopping, reduce_lr])
```

```
Epoch 1/10
5206/5211 [============================>.] - ETA: 0s - loss: 0.3762 - accuracy:
0.8509
Epoch 1: val_loss improved from inf to 0.18887, saving model to model.h5
5211/5211 [==============================] - 37s 7ms/step - loss: 0.3761 -
accuracy: 0.8510 - val_loss: 0.1889 - val_accuracy: 0.9176 - lr: 0.0010
Epoch 2/10
```

```
5210/5211 [============================>.] - ETA: 0s - loss: 0.1643 - accuracy:
0.9239
Epoch 2: val_loss improved from 0.18887 to 0.16805, saving model to model.h5
5211/5211 [==============================] - 34s 7ms/step - loss: 0.1643 -
accuracy: 0.9239 - val_loss: 0.1680 - val_accuracy: 0.9198 - lr: 0.0010
Epoch 3/10
5204/5211 [============================>.] - ETA: 0s - loss: 0.1457 - accuracy:
0.9282
Epoch 3: val_loss improved from 0.16805 to 0.16208, saving model to model.h5
5211/5211 [==============================] - 35s 7ms/step - loss: 0.1457 -
accuracy: 0.9281 - val_loss: 0.1621 - val_accuracy: 0.9190 - lr: 0.0010
Epoch 4/10
5208/5211 [============================>.] - ETA: 0s - loss: 0.1348 - accuracy:
0.9304
Epoch 4: val_loss did not improve from 0.16208
5211/5211 [==============================] - 34s 7ms/step - loss: 0.1347 -
accuracy: 0.9304 - val_loss: 0.1627 - val_accuracy: 0.9175 - lr: 0.0010
Epoch 5/10
5205/5211 [============================>.] - ETA: 0s - loss: 0.1258 - accuracy:
0.9332
Epoch 5: val_loss did not improve from 0.16208
5211/5211 [==============================] - 35s 7ms/step - loss: 0.1258 -
accuracy: 0.9332 - val_loss: 0.1682 - val_accuracy: 0.9147 - lr: 0.0010
Epoch 6/10
5210/5211 [============================>.] - ETA: 0s - loss: 0.1191 - accuracy:
0.9353
Epoch 6: val_loss did not improve from 0.16208

Epoch 6: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
5211/5211 [==============================] - 36s 7ms/step - loss: 0.1191 -
accuracy: 0.9353 - val_loss: 0.1702 - val_accuracy: 0.9149 - lr: 0.0010
Epoch 7/10
5210/5211 [============================>.] - ETA: 0s - loss: 0.1006 - accuracy:
0.9423
Epoch 7: val_loss did not improve from 0.16208
5211/5211 [==============================] - 37s 7ms/step - loss: 0.1006 -
accuracy: 0.9422 - val_loss: 0.1741 - val_accuracy: 0.9088 - lr: 2.0000e-04
Epoch 8/10
5211/5211 [==============================] - ETA: 0s - loss: 0.0960 - accuracy:
0.9439
Epoch 8: val_loss did not improve from 0.16208
5211/5211 [==============================] - 37s 7ms/step - loss: 0.0960 -
accuracy: 0.9439 - val_loss: 0.1784 - val_accuracy: 0.9064 - lr: 2.0000e-04
Epoch 8: early stopping
```

[40]:
```
import pickle
```

```
# Saving the tokenizer
with open('tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)

model.save('CNN.h5')
```

2 Application

[41]:
```
import joblib
import keras
from keras_preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
import numpy as np
import pickle
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score

def preprocess_text(text,tokenizer):
    sequences = tokenizer.texts_to_sequences([text])
    padded_sequences = pad_sequences(sequences, maxlen=100)
    return padded_sequences
def main():
    # Load necessary model
    vectorizer = joblib.load('tfidf_vectorizer.pkl')
    naive_bayes_model = joblib.load('naive_bayes_model.pkl')
    logreg = joblib.load('logistic_regression_model.joblib')
    CNN = keras.models.load_model('CNN.h5')

    with open('tokenizer.pickle', 'rb') as handle:
        tokenizer = pickle.load(handle)

    index_to_emotion = {
    0: 'anger',
    1: 'fear',
    2: 'joy',
    3: 'love',
    4: 'sadness',
    5: 'surprise'
      }

    text=input("Enter your text here:")

    # preprocess text
    processed_text = preprocess_text(text,tokenizer)
    text_vector = vectorizer.transform([text])
```

```
    prediction_naive_bayes = naive_bayes_model.predict(text_vector)
    probabilities_naive_bayes = naive_bayes_model.predict_proba(text_vector)

    prediction_logreg = logreg.predict(text_vector)
    probabilities_logreg = logreg.predict_proba(text_vector)

    prediction_CNN = CNN.predict(processed_text)
    emotion_index = np.argmax(prediction_CNN)
    emotion_label = index_to_emotion[emotion_index]

    print("\nPredictions and Probabilities:\n")
    print("Naive Bayes Prediction: ", prediction_naive_bayes[0])
    print("Naive Bayes Probabilities: ")
    for i in range(len(probabilities_naive_bayes[0])):
        print(f"{index_to_emotion[i]}:{probabilities_naive_bayes[0][i]} ")
    print("\nLogistic Regression Prediction: ", prediction_logreg[0])
    for i in range(len(probabilities_logreg[0])):
        print(f"{index_to_emotion[i]}:{probabilities_logreg[0][i]} ")
    print("\nCNN Prediction (Emotion): ", emotion_label)
    for i in range(len(prediction_CNN[0])):
        print(f"{index_to_emotion[i]}:{prediction_CNN[0][i]} ")

if __name__ == "__main__":
    main()
```

```
Enter your text here:I slammed my fist in anger, but then burst into laughter,
realizing the absurdity of it all.
1/1 [==============================] - 0s 38ms/step

Predictions and Probabilities:

Naive Bayes Prediction:  joy
Naive Bayes Probabilities:
anger:0.2190863061281364
fear:0.1388256211466991
joy:0.26072114153468845
love:0.11434620221122123
sadness:0.24181941676627788
surprise:0.02520131221297537

Logistic Regression Prediction:  anger
anger:0.4017210688862057
fear:0.12184924117849924
joy:0.23832651889324377
love:0.06885038501146967
sadness:0.1315110941870479
surprise:0.03774169184353367
```

```
CNN Prediction (Emotion):  anger
anger:0.5735710859298706
fear:0.19589252769947052
joy:0.05969538912177086
love:0.0007390704704448581
sadness:0.1674700677394867
surprise:0.0026317897718399763
```