Imports

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [2]:
```
### misc stuffs
import spacy
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_sco
from sklearn.model_selection import train_test_split
### torch stuff
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset, TensorDataset
```

In [3]:
```
#must print /content, assuming your structure is content / drive/ myDrive/ 142a colab/ (test or t
test = pd.read_csv('drive/MyDrive/142a colab/test.csv', usecols=[0,2], names = ['posRating', 'te
train = pd.read_csv('drive/MyDrive/142a colab/train.csv', usecols = [0, 2], names = ['posRating'
train.posRating -= 1
test.posRating -= 1
train, val = train_test_split(train, test_size = 0.1, random_state= 69)

nlp = spacy.load('en_core_web_sm')
```

GRU Model

In [4]:
```
def tokenizer(df):
    docs = nlp.pipe(df['text'].str.lower(), batch_size=50, n_process=-1)
    return [[token.text for token in doc if not token.is_punct and not token.is_space and not to

def padTrunc(tokenized_docs, max_length= 30, padToken = '<PAD>'):
    num_docs = len(tokenized_docs)
    result = np.full((num_docs, max_length), padToken, dtype=object)  # Use `object` for string

    for i, doc in enumerate(tokenized_docs):

        truncated = doc[:max_length]  # Truncate to max_length
        result[i, :len(truncated)] = truncated  # Place truncated tokens into the array

    return result

def embed():
    embeddings = {}
    with open('drive/MyDrive/142a colab/glove.6B.50d.txt', "r", encoding="utf-8") as f:
        for line in f:
            values = line.split()
            word = values[0]
            vector = np.asarray(values[1:], dtype="float32")
            embeddings[word] = vector
    return embeddings

def tokenToEmbedding(tokenized_docs, glove_dict, embedding_dim):
```

```
    """
    Convert tokenized documents into embedding vectors.

    Args:
    - tokenized_docs (list of lists): Tokenized and padded documents.
    - glove_dict (dict): Pre-loaded GloVe dictionary mapping words to embeddings.
    - embedding_dim (int): Dimension of GloVe embeddings.

    Returns:
    - np.ndarray: 3D array of embeddings (num_docs, max_length, embedding_dim).
    """
    # Create a 3D array to hold embeddings
    num_docs = len(tokenized_docs)
    max_length = len(tokenized_docs[0])  # Assume all documents are padded to the same length
    embeddings = np.zeros((num_docs, max_length, embedding_dim), dtype=np.float32)

    for i, doc in enumerate(tokenized_docs):
        for j, token in enumerate(doc):
            if token in glove_dict:
                embeddings[i, j] = glove_dict[token]
            else:
                embeddings[i, j] = np.zeros(embedding_dim)  # Use zero vector for OOV or <PAD>

    return embeddings
```

In [5]:
```
embeddings = embed()
```

In [6]:
```
def preProcess(df):
    mat = padTrunc(tokenizer(df))
    Emb = torch.tensor(tokenToEmbedding(mat, embeddings, 50))
    Lbl = torch.tensor(df['posRating'].values).float()
    return TensorDataset(Emb, Lbl)
```

In [7]:
```
#MAX_SEQ_LEN = 30      fixed sequence length as 30, close to mean length
#EMBEDDING_DIM = 50    dimension of GloVe word embeddings

# Initialize DataLoaders
trainMatrix = preProcess(train)
valMatrix = preProcess(val)
train_loader = DataLoader(trainMatrix, batch_size = 27, shuffle = True)
val_loader = DataLoader(valMatrix, batch_size = 20, shuffle = False)
```

In [8]:
```
class GRUSentiment(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, drop):
        super(GRUSentiment, self).__init__()
        self.gru = nn.GRU(input_size, hidden_size, num_layers, batch_first=True, dropout = drop)
        self.output_layer = nn.Linear(hidden_size, 1)


    def forward(self, x):
        gru_out, h_n = self.gru(x)  # gru_out: (batch, sequence length (30) , hidden_size) H_n:
        output = self.output_layer(h_n[-1]).flatten() # last hidden state: (batch, hidden) --> h_
        return output # (batch)

def get_loss_fn():
    return nn.BCEWithLogitsLoss()

def get_validation_accuracy(model, device):
    model.eval()
    correct = 0
```

```
    totalPreds = 0
    yPreds = []
    with torch.no_grad():
        for batch in val_loader:
            x = batch[0].to(device)
            y = batch[1].to(device)
            probs = torch.sigmoid(model.forward(x))
            preds = (probs >= 0.5).float()
            yPreds.extend(preds.cpu().numpy())
            correct += (preds == y).sum().item()
            totalPreds += preds.shape[0]
    return (correct / totalPreds), f1_score(val['posRating'].values, yPreds)

def train_model(model,device):
    model.to(device)
    optimizer = optim.Adam(model.parameters(), lr=0.00015)
    criterion = get_loss_fn()

    for epoch in range(20):
        model.train()

        for batch in train_loader:
            x = batch[0].to(device)
            y = batch[1].to(device).float()
            optimizer.zero_grad()
            logits = model.forward(x)
            loss = criterion(logits, y)
            loss.backward()
            optimizer.step()

        val_accuracy = get_validation_accuracy(model,device)
        print(f"Epoch {epoch} Validation Accuracy = {val_accuracy[0]:.4f}")
        print(f"F1 Score = {val_accuracy[1]:.4f}")

    return
```

In [9]:

```
"""
Model hyperparameters & Training
"""
input_size = 50
hidden_size = 150 # Increased hidden size for better representation
num_layers = 2  # Reduced number of layers to prevent overfitting
drop = 0.1 #model parameter dropout probability
model = GRUSentiment(input_size, hidden_size, num_layers, drop)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
train_model(model, device)
```

```
Epoch 0 Validation Accuracy = 0.6967
F1 Score = 0.6473
Epoch 1 Validation Accuracy = 0.7042
F1 Score = 0.6828
Epoch 2 Validation Accuracy = 0.7000
F1 Score = 0.7183
Epoch 3 Validation Accuracy = 0.7192
F1 Score = 0.7310
Epoch 4 Validation Accuracy = 0.7200
F1 Score = 0.7143
Epoch 5 Validation Accuracy = 0.7242
F1 Score = 0.7436
Epoch 6 Validation Accuracy = 0.7458
F1 Score = 0.7494
Epoch 7 Validation Accuracy = 0.7650
F1 Score = 0.7561
Epoch 8 Validation Accuracy = 0.7625
F1 Score = 0.7718
Epoch 9 Validation Accuracy = 0.7617
F1 Score = 0.7744
Epoch 10 Validation Accuracy = 0.7808
F1 Score = 0.7719
Epoch 11 Validation Accuracy = 0.7617
F1 Score = 0.7769
Epoch 12 Validation Accuracy = 0.7800
F1 Score = 0.7782
Epoch 13 Validation Accuracy = 0.7608
F1 Score = 0.7827
Epoch 14 Validation Accuracy = 0.7683
F1 Score = 0.7491
Epoch 15 Validation Accuracy = 0.7817
F1 Score = 0.7623
Epoch 16 Validation Accuracy = 0.7725
F1 Score = 0.7636
Epoch 17 Validation Accuracy = 0.7533
F1 Score = 0.7709
Epoch 18 Validation Accuracy = 0.7817
F1 Score = 0.7741
Epoch 19 Validation Accuracy = 0.7750
F1 Score = 0.7837
```

In [10]:
```
test_loader = DataLoader(preProcess(test), batch_size = 20, shuffle = False)
```

In [11]:
```
def predict(model, device, test_loader):
    model.eval()
    all_predictions = []

    with torch.no_grad():
        for batch in test_loader:
            x = batch[0].to(device)
            logits = model(x)  # Raw logits
            probs = torch.sigmoid(logits)  # Convert logits to probabilities
            preds = (probs >= 0.5).float()  # Binary predictions
            all_predictions.extend(preds.cpu().numpy())

    return all_predictions
```

In [12]:
```
y_pred = predict(model, device, test_loader)
y_true = test.posRating
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
```

```
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
roc_auc = roc_auc_score(y_true, y_pred)
print(f"Test Accuracy: {accuracy:.4f}")
print(f"Test Precision: {precision:.4f}")
print(f"Test Recall: {recall:.4f}")
print(f"Test F1-Score: {f1:.4f}")
print(f"Test ROC-AUC: {roc_auc:.4f}")
```

```
Test Accuracy: 0.7583
Test Precision: 0.7509
Test Recall: 0.8023
Test F1-Score: 0.7758
Test ROC-AUC: 0.7564
```

Multinomial Naive Bayes Model

In [13]:
```
import pandas as pd

train = pd.read_csv('drive/MyDrive/142a colab/train.csv', header=None, nrows = 12000)
test = pd.read_csv('drive/MyDrive/142a colab/test.csv', header=None, nrows = 3000)
train.columns = ['posRating', 'first_review', 'second_review']
test.columns = ['posRating', 'first_review', 'second_review']
```

In [14]:
```
# Melt the train DataFrame
train_long = pd.melt(
    train,
    id_vars='posRating',
    value_vars=['first_review', 'second_review'],
    var_name='original_column',
    value_name='text'
)
train_long.drop(columns='original_column', inplace=True)

# Melt the test DataFrame
test_long = pd.melt(
    test,
    id_vars='posRating',
    value_vars=['first_review', 'second_review'],
    var_name='original_column',
    value_name='text'
)
test_long.drop(columns='original_column', inplace=True)

# Map the posRating values: 1 -> 0 (negative), 2 -> 1 (positive)
rating_map = {1: 0, 2: 1}
train_long['posRating'] = train_long['posRating'].replace(rating_map)
test_long['posRating'] = test_long['posRating'].replace(rating_map)
train_long.dropna(subset=['posRating', 'text'], inplace=True)
test_long.dropna(subset=['posRating', 'text'], inplace=True)
```

In [15]:
```
#Sam
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# Extract training and test data
train_texts = train_long['text']
train_labels = train_long['posRating']
```

```
test_texts = test_long['text']
test_labels = test_long['posRating']

# Vectorize text (bag-of-words representation)
vectorizer = TfidfVectorizer(stop_words='english')  # limit vocabulary for efficiency
X_train = vectorizer.fit_transform(train_texts)
X_test = vectorizer.transform(test_texts)

# Initialize and train the Multinomial Naive Bayes model
mnb_model = MultinomialNB()
mnb_model.fit(X_train, train_labels)

# Predict on the test set
y_pred = mnb_model.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(test_labels, y_pred))
print("Classification Report:")
print(classification_report(test_labels, y_pred))
```

```
Accuracy: 0.7880960320106702
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.81      0.78      2873
           1       0.81      0.77      0.79      3125

    accuracy                           0.79      5998
   macro avg       0.79      0.79      0.79      5998
weighted avg       0.79      0.79      0.79      5998
```

Logistic Regression Model

In [16]:

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import numpy as np
import string
import os

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

print(os.getcwd())

train = pd.read_csv('drive/MyDrive/142a colab/train.csv', header=None, nrows = 12000)
test = pd.read_csv('drive/MyDrive/142a colab/test.csv', header=None, nrows = 3000)

train.columns = ['posRating', 'subject', 'text']
test.columns = ['posRating', 'subject', 'text']

train.head()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/conte
nt/drive", force_remount=True).
/content
```

Out[16]:

	posRating	subject	text
0	2	Stuning even for the non-gamer	This sound track was beautiful! It paints the ...
1	2	The best soundtrack ever to anything.	I'm reading a lot of reviews saying that this ...
2	2	Amazing!	This soundtrack is my favorite music of all ti...
3	2	Excellent Soundtrack	I truly like this soundtrack and I enjoy video...
4	2	Remember, Pull Your Jaw Off The Floor After He...	If you've played the game, you know how divine...

In [17]:
```
x_train = train['text']
y_train = train['posRating']

x_test = test['text']
y_test = test['posRating']
```

In [18]:
```
vectorizer = TfidfVectorizer(stop_words='english', lowercase=True)
```

In [19]:
```
X_train_tfidf = vectorizer.fit_transform(x_train)
X_test_tfidf = vectorizer.transform(x_test)
```

In [20]:
```
model = LogisticRegression(max_iter=1000)
model.fit(X_train_tfidf, y_train)
```

Out[20]:

LogisticRegression

LogisticRegression(max_iter=1000)

In [21]:
```
y_pred = model.predict(X_test_tfidf)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.816
```

FastText Method

In [22]:
```
!pip install gensim
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.11/dist-packages (4.3.3)
Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.11/dist-packages (fro
m gensim) (1.26.4)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.11/dist-packages (f
rom gensim) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from
gensim) (7.1.0)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>
=1.8.1->gensim) (1.17.2)
```

In [23]:
```
from gensim.models import FastText
import os
print(os.getcwd())
```

```
/content
```

In [24]:
```
train = pd.read_csv('drive/MyDrive/142a colab/train.csv', header=None, nrows = 12000)
test = pd.read_csv('drive/MyDrive/142a colab/test.csv', header=None, nrows = 3000)

train.columns = ['posRating', 'subject', 'text']
test.columns = ['posRating', 'subject', 'text']
```

In [25]:
```
X_train = train['text'].astype(str).apply(lambda x: x.lower().split())
y_train = train['posRating']

X_test = test['text'].astype(str).apply(lambda x: x.lower().split())
y_test = test['posRating']
```

In [26]:
```
fasttext_model = FastText(sentences=X_train, vector_size=100, window=5, min_count=5, workers=4)

def get_average_embedding(tokens, model):
    valid_tokens = [t for t in tokens if t in model.wv]
    if not valid_tokens:
        return np.zeros(model.wv.vector_size)
    return np.mean([model.wv[t] for t in valid_tokens], axis=0)
```

In [27]:
```
X_train_emb = np.vstack(X_train.apply(lambda x: get_average_embedding(x, fasttext_model)))
X_test_emb = np.vstack(X_test.apply(lambda x: get_average_embedding(x, fasttext_model)))
```

In [28]:
```
model = LogisticRegression(max_iter=1000)
model.fit(X_train_emb, y_train)
```

Out[28]:
```
LogisticRegression
LogisticRegression(max_iter=1000)
```

In [29]:
```
y_pred = model.predict(X_test_emb)

acc = accuracy_score(y_test, y_pred)
print("Test Accuracy (FastText + LR):", acc)
```

```
Test Accuracy (FastText + LR): 0.7053333333333334
```

Conv1D CNN Model

In [30]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt_tab')
nltk.download('punkt')
nltk.download('stopwords')

punc = string.punctuation

test = pd.read_csv('drive/MyDrive/142a colab/test.csv', names=['posRating', 'subject', 'text'],
train = pd.read_csv('drive/MyDrive/142a colab/train.csv', names=['posRating', 'subject', 'text']

translation_table = str.maketrans("", "", punc)
for df in [train, test]:
```

```
    df['subject'] = df['subject'].astype(str).str.translate(translation_table).str.lower().apply
    df['text'] = df['text'].astype(str).str.translate(translation_table).str.lower().apply(word_

train.posRating -= 1
test.posRating -= 1

train, val = train_test_split(train, test_size=0.005)
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [31]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences


train['combined'] = train['subject'] + train['text']
val['combined'] = val['subject'] + val['text']
test['combined'] = test['subject'] + test['text']


tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(train['combined'])

X_train = tokenizer.texts_to_sequences(train['combined'])
X_val = tokenizer.texts_to_sequences(val['combined'])
X_test = tokenizer.texts_to_sequences(test['combined'])

maxlen = 100
X_train = pad_sequences(X_train, maxlen=maxlen, padding='post')
X_val = pad_sequences(X_val, maxlen=maxlen, padding='post')
X_test = pad_sequences(X_test, maxlen=maxlen, padding='post')


y_train = np.array(train['posRating'])
y_val = np.array(val['posRating'])
y_test = np.array(test['posRating'])
```

In [32]:
```
glove_path = 'drive/MyDrive/142a colab/glove.6B.50d.txt'

embeddings_index = {}
with open(glove_path, 'r', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = vector

print(f"Loaded {len(embeddings_index)} word vectors.")

embedding_dim = 50


embedding_matrix = np.zeros((len(tokenizer.word_index) + 1, embedding_dim))

for word, i in tokenizer.word_index.items():
    if word in embeddings_index:
        embedding_matrix[i] = embeddings_index[word]
```

```
    else:
        embedding_matrix[i] = np.random.randn(embedding_dim)

print(f"Embedding matrix shape: {embedding_matrix.shape}")

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense, Dropout

model = Sequential([
    Embedding(input_dim=len(tokenizer.word_index) + 1,
              output_dim=embedding_dim,
              weights=[embedding_matrix],
              input_length=maxlen,
              trainable=False),
    Conv1D(filters=128, kernel_size=5, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])


model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.summary()


history = model.fit(
    X_train, y_train,
    epochs=10,
    batch_size=512,
    validation_data=(X_val, y_val)
)

test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc:.2f}")

model.save('sentiment_model.h5')

from tensorflow.keras.models import load_model
model = load_model('sentiment_model.h5')
```

```
Loaded 400000 word vectors.
Embedding matrix shape: (61381, 50)
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	3,069,050
conv1d (Conv1D)	?	0 (unbuilt)
global_max_pooling1d (GlobalMaxPooling1D)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)
dropout (Dropout)	?	0 (unbuilt)
dense_1 (Dense)	?	0 (unbuilt)

Total params: 3,069,050 (11.71 MB)

Trainable params: 0 (0.00 B)

Non-trainable params: 3,069,050 (11.71 MB)

```
Epoch 1/10
39/39 ━━━━━━━━━━━━━━━━━━━━ 10s 117ms/step - accuracy: 0.5650 - loss: 0.7064 - val_accuracy: 0.730
0 - val_loss: 0.5822
Epoch 2/10
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7032 - loss: 0.5780 - val_accuracy: 0.7800 -
val_loss: 0.4743
Epoch 3/10
39/39 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.7631 - loss: 0.4969 - val_accuracy: 0.7600 -
val_loss: 0.4512
Epoch 4/10
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7996 - loss: 0.4462 - val_accuracy: 0.7600 -
val_loss: 0.4308
Epoch 5/10
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8241 - loss: 0.3972 - val_accuracy: 0.7400 -
val_loss: 0.5129
Epoch 6/10
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8275 - loss: 0.3903 - val_accuracy: 0.7700 -
val_loss: 0.4189
Epoch 7/10
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8523 - loss: 0.3510 - val_accuracy: 0.7900 -
val_loss: 0.4113
Epoch 8/10
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.8718 - loss: 0.3241 - val_accuracy: 0.8200 -
val_loss: 0.4094
Epoch 9/10
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8805 - loss: 0.3006 - val_accuracy: 0.8200 -
val_loss: 0.3991
Epoch 10/10
39/39 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.8984 - loss: 0.2657 - val_accuracy: 0.7800 -
val_loss: 0.4048
157/157 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8143 - loss: 0.4113
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_m
odel(model)`. This file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')
`.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.com
pile_metrics` will be empty until you train or evaluate the model.
Test Accuracy: 0.82
```