```
from google.colab import drive                                          #mounting the drive to acces the
drive.mount('/content/drive')
```

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

```
!pip install emoji
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd                                                     #helps in data analysis and mani
import pickle
import re                                                               # helps in the removal of unwant
import emoji                                                            # a python library which tranfro
from nltk.stem import PorterStemmer as ps                               # reduces infected words to thei
from tensorflow.keras.preprocessing.text import Tokenizer               # encodes the words to numbers wi
from tensorflow.keras.preprocessing.sequence import pad_sequences       #helps in regularising the lengt
from sklearn.model_selection import train_test_split                    #splits the data into training a
import tensorflow as tf
import keras
from keras.metrics import Precision, Recall
```

```
Requirement already satisfied: emoji in /usr/local/lib/python3.7/dist-packages (1.4.1)
```

```
csv_path = "/content/drive/MyDrive/ENTHIRE/airline_sentiment_analysis.csv"

class airline_data():
    def __init__(self,data_dir_path):
        self.df = pd.read_csv(data_dir_path)                                        #loading the cs
        self.df = pd.concat([self.df["text"],self.df["airline_sentiment"]],axis=1) #concatenating t

    def process_text(self,text):
        new_text = text.lower()                                         #making all the characters lower
        new_text = re.sub(r'@\w+', '', new_text)                        # Remove @s
        new_text = re.sub(r'#', '', new_text)                           # Remove hashtags
        new_text = re.sub(r':', ' ', emoji.demojize(new_text))          # Turn emojis into words
        new_text = re.sub(r'http\S+', '',new_text)                      # Remove URLs
        new_text = re.sub(r'\$\S+', 'dollar', new_text)                 # Change dollar amounts to dollar
        new_text = re.sub(r'[^a-z0-9\s]', '', new_text)                 # Remove punctuation
        new_text = re.sub(r'[0-9]+', 'number', new_text)                # Change number values to number
        new_text = new_text.split(" ")                                  # splits the text into a list of
        new_text = list(map(lambda x: ps().stem(x), new_text))          # Stemming the words
        new_text = list(map(lambda x: x.strip(), new_text))             # Stripping whitespace from the w
        if '' in new_text:
            new_text.remove('')
        return new_text                                                 # returns sentence of words in th

    def preprocess_data(self):
      self.Texts = self.df["text"].apply(self.process_text)
      sentiment_ordering = ['negative','positive']
      self.labels = self.df["airline_sentiment"].apply(lambda x:sentiment_ordering.index(x))


    def retrieve_vocab_info(self):
      self.preprocess_data()
      vocabulary = set()
      for text in self.Texts:
          for word in text:
```

```
                vocabulary.add(word)

      self.vocab_length = len(vocabulary)
      self.max_seq_length = 0
      for text in self.Texts:
          if len(text) > self.max_seq_length:
              self.max_seq_length = len(text)
      return self.vocab_length,self.max_seq_length

    def tokenize_words(self):
      self.retrieve_vocab_info()
      self.training_sentences,self.testing_sentences,self.y_train,self.y_test = train_test_split(se
      tokenizer = Tokenizer(num_words=self.vocab_length,oov_token="<OOV>")
      tokenizer.fit_on_texts(self.training_sentences)
      word_index = tokenizer.word_index
      self.training_sequences = tokenizer.texts_to_sequences(self.training_sentences)
      self.X_train = pad_sequences(self.training_sequences, maxlen=self.max_seq_length, padding='po

      self.testing_sequences = tokenizer.texts_to_sequences(self.testing_sentences)
      self.X_test = pad_sequences(self.testing_sequences, maxlen=self.max_seq_length, padding='post

      pickle.dump(tokenizer,open("/content/drive/MyDrive/ENTHIRE/GRU_Model/tokenizer_file2.pkl","wb

      return self.X_train,self.X_test,self.y_train,self.y_test

# DATA LOADING
data = airline_data(csv_path)
vocab_length,max_seq_length = data.retrieve_vocab_info()
X_train,X_test,y_train,y_test = data.tokenize_words()


class GRU_Model(keras.Model):
  def __init__(self):
        super(GRU_Model,self).__init__()
        self.embed_layer = tf.keras.layers.Embedding(vocab_length, output_dim = 32, input_length=ma
        self.flatten = tf.keras.layers.Flatten()
        self.gru = tf.keras.layers.GRU(units=32)
        self.gru_flatten = tf.keras.layers.Flatten()

        self.dense = tf.keras.layers.Dense(1, activation='sigmoid')

  def call(self, inputs, training=None):
        embed       = self.embed_layer(inputs)
        flatten     = self.flatten(embed)
        gru         = self.gru(embed)
        gru_flatten = self.flatten(gru)
        concat      = tf.keras.layers.concatenate([flatten,gru_flatten])
        out         = self.dense(concat)
        return out

model = GRU_Model()



model.compile(optimizer="adam",loss="binary_crossentropy", metrics =['accuracy',Precision(), Recal
history = model.fit(X_train,y_train,validation_split=0.2,batch_size=32,epochs=100,
                            callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss',patie
```

```
model.save("/content/drive/MyDrive/ENTHIRE/GRU_Model/30_model/")
```

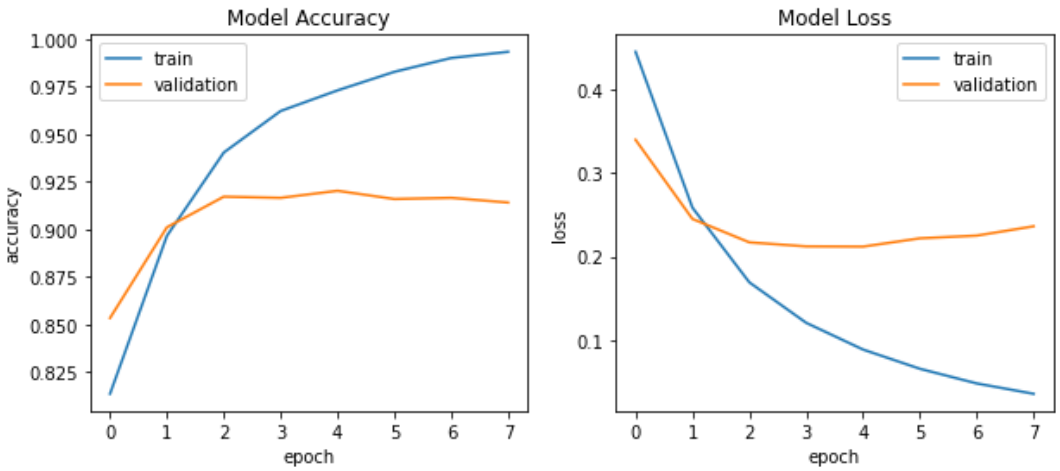
```
Epoch 1/100
202/202 [==============================] - 26s 52ms/step - loss: 0.5079 - accuracy: 0.7926 - p
Epoch 2/100
202/202 [==============================] - 10s 48ms/step - loss: 0.2943 - accuracy: 0.8782 - p
Epoch 3/100
202/202 [==============================] - 10s 47ms/step - loss: 0.1749 - accuracy: 0.9360 - p
Epoch 4/100
202/202 [==============================] - 10s 48ms/step - loss: 0.1159 - accuracy: 0.9658 - p
Epoch 5/100
202/202 [==============================] - 10s 48ms/step - loss: 0.0873 - accuracy: 0.9740 - p
Epoch 6/100
202/202 [==============================] - 10s 48ms/step - loss: 0.0647 - accuracy: 0.9837 - p
Epoch 7/100
202/202 [==============================] - 10s 49ms/step - loss: 0.0493 - accuracy: 0.9902 - p
Epoch 8/100
202/202 [==============================] - 10s 48ms/step - loss: 0.0355 - accuracy: 0.9936 - p
Restoring model weights from the end of the best epoch.
Epoch 00008: early stopping
WARNING:tensorflow:Skipping full serialization of Keras layer <tensorflow.python.keras.layers.
WARNING:absl:Found untraced functions such as embedding_3_layer_call_and_return_conditional_lo
INFO:tensorflow:Assets written to: /content/drive/MyDrive/ENTHIRE/GRU_Model/30_model/assets
INFO:tensorflow:Assets written to: /content/drive/MyDrive/ENTHIRE/GRU_Model/30_model/assets
```

```
import keras.backend as K
def f1_score(precision, recall):
    ''' Function to calculate f1 score '''

    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
loss, accuracy, precision, recall = model.evaluate(X_test, y_test, verbose=0)
# Print metrics
print('')
print('Accuracy  : {:.4f}'.format(accuracy))
print('Precision : {:.4f}'.format(precision))
print('Recall    : {:.4f}'.format(recall))
print('F1 Score  : {:.4f}'.format(f1_score(precision, recall)))
```

```
Accuracy  : 0.9209
Precision : 0.8191
Recall    : 0.7767
F1 Score  : 0.7973
```

```
def plot_training_hist(history):
    # '''Function to plot history for accuracy and loss
        fig, ax = plt.subplots(1, 2, figsize=(10,4))
        # first plot
        #plots the training and validation accuracy over all the epochs
        ax[0].plot(history.history['accuracy'])
        ax[0].plot(history.history['val_accuracy'])
        ax[0].set_title('Model Accuracy')
        ax[0].set_xlabel('epoch')
        ax[0].set_ylabel('accuracy')
        ax[0].legend(['train', 'validation'], loc='best')
        # second plot
        #plots the training and validation loss over all the epochs
        ax[1].plot(history.history['loss'])
```

```
        ax[1].plot(history.history['val_loss'])
        ax[1].set_title('Model Loss')
        ax[1].set_xlabel('epoch')
        ax[1].set_ylabel('loss')
        ax[1].legend(['train', 'validation'], loc='best')

plot_training_hist(history)
```



Start coding or generate with AI.