```
# -*- coding: utf-8 -*-
"""All Models for SA with conf matrix heatmap.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1SGhruP5jfkqgtHIfEQPx_XEG2U-
YYCrv
"""

# Install necessary libraries
!pip install datasets pandas tensorflow scikit-learn transformers -U

!pip install seaborn

"""Import necessary libraries"""

import pandas as pd
import numpy as np
from datasets import load_dataset
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score,
precision_score, recall_score, f1_score
import tensorflow as tf  # Add this import for TensorFlow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Bidirectional,
Dense, Dropout, Conv1D, GRU
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from transformers import BertTokenizer, TFBertModel

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

"""Load the dataset from Hugging Face"""

dataset = load_dataset("amirpoudel/bert-reviews-data")
df = pd.DataFrame(dataset['train'])

"""Prepare the data"""

X = df['text']
y = df['label']

"""Encode labels"""

class_names = ['neutral', 'positive', 'negative']
label_map = {label: idx for idx, label in enumerate(class_names)}
```

```
y_encoded = df['label'].map(label_map).values

"""Split the dataset into training and validation sets"""

X_train, X_val, y_train, y_val = train_test_split(X, y_encoded,
test_size=0.2, random_state=42)

"""Function to evaluate model"""

def evaluate_model(y_val, y_pred):
    accuracy = accuracy_score(y_val, y_pred)
    precision = precision_score(y_val, y_pred, average='weighted')
    recall = recall_score(y_val, y_pred, average='weighted')
    f1 = f1_score(y_val, y_pred, average='weighted')

    print(f"Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall:
{recall:.4f}, F1 Score: {f1:.4f}")
    # return accuracy, precision, recall, f1

"""Function to plot confusion matrix"""

def plot_confusion_matrix(y_true, y_pred, classes, model_name):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=classes, yticklabels=classes)
    plt.title(f'Confusion Matrix for {model_name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

"""Multinomial Naive Bayes"""

vectorizer = TfidfVectorizer()
X_train_nb = vectorizer.fit_transform(X_train)
X_val_nb = vectorizer.transform(X_val)

nb_model = MultinomialNB()
nb_model.fit(X_train_nb, y_train)
y_pred_nb = nb_model.predict(X_val_nb)
print("Multinomial Naive Bayes:")
evaluate_model(y_val, y_pred_nb)
print("")

plot_confusion_matrix(y_val, y_pred_nb, class_names, "Multinomial Naive
Bayes")

"""K-Nearest Neighbors"""

knn_model = KNeighborsClassifier()
knn_model.fit(X_train_nb, y_train)
y_pred_knn = knn_model.predict(X_val_nb)
print("\nK-Nearest Neighbors:")
evaluate_model(y_val, y_pred_nb)
```

```
print("")

plot_confusion_matrix(y_val, y_pred_knn, class_names, "K-Nearest
Neighbors")

"""Random Forest"""

rf_model = RandomForestClassifier()
rf_model.fit(X_train_nb, y_train)
y_pred_rf = rf_model.predict(X_val_nb)
print("\nRandom Forest:")
evaluate_model(y_val, y_pred_nb)
print("")

plot_confusion_matrix(y_val, y_pred_rf, class_names, "Random Forest")

"""Support Vector Machine"""

svm_model = SVC()
svm_model.fit(X_train_nb, y_train)
y_pred_svm = svm_model.predict(X_val_nb)
print("\nSupport Vector Machine:")
evaluate_model(y_val, y_pred_nb)
print("")

plot_confusion_matrix(y_val, y_pred_svm, class_names, "Support Vector
Machine")

"""RNN Model"""

max_words = 5000
max_len = 100
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X_train)
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_val_seq = tokenizer.texts_to_sequences(X_val)
X_train_pad = pad_sequences(X_train_seq, maxlen=max_len)
X_val_pad = pad_sequences(X_val_seq, maxlen=max_len)

rnn_model = Sequential()
rnn_model.add(Embedding(input_dim=max_words, output_dim=128,
input_length=max_len))
rnn_model.add(GRU(128))
rnn_model.add(Dense(len(class_names), activation='softmax'))
rnn_model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
rnn_model.fit(X_train_pad, y_train, epochs=5, batch_size=32,
validation_data=(X_val_pad, y_val))
y_pred_rnn = np.argmax(rnn_model.predict(X_val_pad), axis=1)
print("\nRNN Model:")
evaluate_model(y_val, y_pred_nb)
print("")

plot_confusion_matrix(y_val, y_pred_rnn, class_names, "RNN Model")
```

```
"""LSTM Model"""

lstm_model = Sequential()
lstm_model.add(Embedding(input_dim=max_words, output_dim=128,
input_length=max_len))
lstm_model.add(LSTM(128))
lstm_model.add(Dense(len(class_names), activation='softmax'))
lstm_model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
lstm_model.fit(X_train_pad, y_train, epochs=5, batch_size=32,
validation_data=(X_val_pad, y_val))
y_pred_lstm = np.argmax(lstm_model.predict(X_val_pad), axis=1)
print("\nLSTM Model:")
evaluate_model(y_val, y_pred_nb)
print("")

plot_confusion_matrix(y_val, y_pred_lstm, class_names, "LSTM Model")

"""Bidirectional LSTM Model"""

bilstm_model = Sequential()
bilstm_model.add(Embedding(input_dim=max_words, output_dim=128,
input_length=max_len))
bilstm_model.add(Bidirectional(LSTM(128)))
bilstm_model.add(Dense(len(class_names), activation='softmax'))
bilstm_model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
bilstm_model.fit(X_train_pad, y_train, epochs=5, batch_size=32,
validation_data=(X_val_pad, y_val))
y_pred_bilstm = np.argmax(bilstm_model.predict(X_val_pad), axis=1)
print("\nBidirectional LSTM Model:")
evaluate_model(y_val, y_pred_nb)
print("")

plot_confusion_matrix(y_val, y_pred_bilstm, class_names, "Bidirectional
LSTM Model")

"""CNN Model"""

cnn_model = Sequential()
cnn_model.add(Embedding(input_dim=max_words, output_dim=128,
input_length=max_len))
cnn_model.add(Conv1D(filters=64, kernel_size=5, activation='relu'))
cnn_model.add(Dropout(0.5))
cnn_model.add(GRU(128))
cnn_model.add(Dense(len(class_names), activation='softmax'))
cnn_model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
cnn_model.fit(X_train_pad, y_train, epochs=5, batch_size=32,
validation_data=(X_val_pad, y_val))
y_pred_cnn = np.argmax(cnn_model.predict(X_val_pad), axis=1)
print("\nCNN Model:")
evaluate_model(y_val, y_pred_nb)
```

```
print("")

plot_confusion_matrix(y_val, y_pred_cnn, class_names, "CNN Model")
```