Sentiment Analysis

Installing and Uploading the Dataset

First uploaded the .tar.gz file on google drive and extracted it using tar command.
Further mounted google drive in the content folder of google colab from where we can access the dataset. Code for it is :

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

Installing and Updating the Library

Required libraries to install are : beautifulsoup4, nltk, pandas, numpy, spacy, scikit-learn.
Code for importing the libraries is :

```
import os
import nltk
import re
import numpy as np
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
nltk.download('punkt')
nltk.download('stopwords')
```

Function to strip html from text :

```
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()
```

Load and Organize the Dataset

```
train_pos_dir = "/content/drive/MyDrive/acllmdb/aclImdb/train/"
train_neg_dir = "/content/drive/MyDrive/acllmdb/aclImdb/train/"
test_pos_dir = "/content/drive/MyDrive/acllmdb/aclImdb/test/"
test_neg_dir = "/content/drive/MyDrive/acllmdb/aclImdb/test/"
positive_train_files = [os.path.join(train_pos_dir, 'pos', f) for f in
os.listdir(os.path.join(train_pos_dir, 'pos'))]
negative_train_files = [os.path.join(train_neg_dir, 'neg', f) for f in
os.listdir(os.path.join(train_neg_dir, 'neg'))]
positive_test_files = [os.path.join(test_pos_dir, 'pos', f) for f in
os.listdir(os.path.join(test_pos_dir, 'pos'))]
negative_test_files = [os.path.join(test_neg_dir, 'neg', f) for f in
os.listdir(os.path.join(test_neg_dir, 'neg'))]
```

Data Cleaning and Preprocessing

```
def preprocess_text(text):
    text = strip_html(text) #strip html tags from text
    text = text.lower() #convert to lower case
    text = re.sub(r'<[^>]+>', ' ', text) #another way to remove tags
    text = re.sub('\[[^]]*\]', '', text)#to remove brackets
    pattern=r'[^a-zA-z0-9\s]'
    text=re.sub(pattern,'',text)#to remove special characters
    tokens = word_tokenize(text)#tokenize text to word
    tokens = [word for word in tokens if word.isalnum()]
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]
    stemmer = PorterStemmer() #stemming the word
    tokens = [stemmer.stem(word) for word in tokens]
    return ' '.join(tokens)
```

Load and Preprocess Training and Testing Data

```
# Load and preprocess training data
X_train = []
y_train = []
for file_path in positive_train_files:
    with open(file_path, 'r', encoding='utf-8') as f:
```

```
        review = f.read()
        X_train.append(preprocess_text(review))
        y_train.append(1)  # Positive sentiment label
for file_path in negative_train_files:
    with open(file_path, 'r', encoding='utf-8') as f:
        review = f.read()
        X_train.append(preprocess_text(review))
        y_train.append(0)  # Negative sentiment label

# Load and preprocess test data
X_test = []
y_test = []
for file_path in positive_test_files:
    with open(file_path, 'r', encoding='utf-8') as f:
        review = f.read()
        X_test.append(preprocess_text(review))
        y_test.append(1)  # Positive sentiment label
for file_path in negative_test_files:
    with open(file_path, 'r', encoding='utf-8') as f:
        review = f.read()
        X_test.append(preprocess_text(review))
        y_test.append(0)  # Negative sentiment label
```

Feature Extraction using TF-IDF

```
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)
```

Training the Data

```
classifier = LogisticRegression(max_iter=1000)
classifier.fit(X_train_tfidf, y_train)
```

Evaluating the Data

```
y_pred = classifier.predict(X_test_tfidf)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
```

Accuracy comes out to be : 84.00%

Hyperparameter Tuning using GridSearchCV

```
# example of grid searching key hyperparametres for logistic regression
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
# define dataset
# define models and parameters
model = LogisticRegression()
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]
# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X_train_tfidf, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
```

```
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

Accuracy comes out to be 87.84% with the best parameters as :
{‘C’:1.0,’penalty’:12,’solver’:’liblinear’}

After trying Logistic Regression, lets try SVC and Gradient Boosting with the following code :

SVC

```
from sklearn.svm import SVC
classifier_for_svc = SVC()
classifier_for_svc.fit(X_train_tfidf, y_train)

# Step 5: Evaluation
y_pred = classifier_for_svc.predict(X_test_tfidf)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")
```

Hyperparameter Tuning

```
# example of grid searching key hyperparametres for SVC
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
# define model and parameters
model = SVC()
kernel = ['poly', 'rbf', 'sigmoid']
C = [50, 10, 1.0, 0.1, 0.01]
gamma = ['scale']
# define grid search
grid = dict(kernel=kernel,C=C,gamma=gamma)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X_train_tfidf, y_train)
# summarize results
```

```
print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

Gradient Boosting along with Hyperparameter Tuning

```
# example of grid searching key hyperparameters for
GradientBoostingClassifier
from sklearn.datasets import make_blobs
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
# define dataset
X, y = make_blobs(n_samples=1000, centers=2, n_features=100,
cluster_std=20)
# define models and parameters
model = GradientBoostingClassifier()
n_estimators = [10, 100, 1000]
learning_rate = [0.001, 0.01, 0.1]
subsample = [0.5, 0.7, 1.0]
max_depth = [3, 7, 9]
# define grid search
grid = dict(learning_rate=learning_rate, n_estimators=n_estimators,
subsample=subsample, max_depth=max_depth)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X, y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

Now we can compare accuracy of each model and select the best one.

Method 2 : Using CNN + LSTM

```
train_texts, test_texts, train_labels, test_labels =
train_test_split(concatenated_df["reviews"], concatenated_df["sentiment"],
test_size=0.05, random_state=42)

num_words=10000
maxlen=500
label_encoder = LabelEncoder()

train_labels = label_encoder.fit_transform(train_labels)
test_labels = label_encoder.transform(test_labels)

tokenizer = Tokenizer(num_words=num_words)
tokenizer.fit_on_texts(train_texts)

train_sequences = tokenizer.texts_to_sequences(train_texts)
test_sequences = tokenizer.texts_to_sequences(test_texts)

train_data = pad_sequences(train_sequences, maxlen=maxlen)
test_data = pad_sequences(test_sequences, maxlen=maxlen)

import tensorflow as tf
from keras.layers import Input, Attention,
Concatenate,MaxPooling1D,Flatten
from keras.models import Model

input_sequence = Input(shape=(maxlen,))
embedding_layer = Embedding(input_dim=num_words,
output_dim=maxlen)(input_sequence)
conv_layer = Conv1D(filters=128, kernel_size=5,
activation='relu')(embedding_layer)
pooling_layer = MaxPooling1D(pool_size=4)(conv_layer)
lstm_layer = LSTM(64)(pooling_layer)
flatten_layer = Flatten()(pooling_layer)
merged_layer = Concatenate()([lstm_layer, flatten_layer])
output_layer = Dense(1, activation='sigmoid')(merged_layer)

model = Model(inputs=input_sequence, outputs=output_layer)
```

```
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
history=model.fit(train_data, train_labels, batch_size=128, epochs=5,
validation_split=0.2)

loss, accuracy = model.evaluate(test_data, test_labels)
print(f'Test accuracy: {accuracy}')
```

Accuracy after just 2 epochs comes out to be : 94.00 %