DATASET:
https://www.kaggle.com/datasets/PromptCloudHQ/amazon-reviews-unlocked-mobile-phones

SOURCE CODE

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from wordcloud import WordCloud

from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.feature_extraction.text import
CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import BernoulliNB,
MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.pipeline import Pipeline

from bs4 import BeautifulSoup
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import SnowballStemmer, WordNetLemmatizer
from nltk import sent_tokenize, word_tokenize, pos_tag

import logging
from gensim.models import word2vec
from gensim.models import Word2Vec
from gensim.models.keyedvectors import KeyedVectors

from keras.preprocessing import sequence
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation,
Lambda
from keras.layers.embeddings import Embedding
```

```
from keras.layers.recurrent import LSTM, SimpleRNN, GRU
from keras.preprocessing.text import Tokenizer
from collections import defaultdict
from keras.layers.convolutional import Convolution1D
from keras import backend as K
from keras.layers.embeddings import Embedding

# Loading data file
df = pd.read_csv('dataset.csv')
df.head()

#Data exploration
print("Summary statistics of numerical features : \n",
df.describe())

print("\nTotal number of reviews: ",len(df))
print("\nTotal number of brands: ",
len(list(set(df['Brand Name']))))
print("\nTotal number of unique products: ",
len(list(set(df['Product Name']))))

print("\nPercentage of reviews with neutral sentiment :
{:.2f}%"\
      .format(df[df['Rating']==3]["Reviews"].count()/
len(df)*100))
print("\nPercentage of reviews with positive sentiment :
{:.2f}%"\
      .format(df[df['Rating']>3]["Reviews"].count()/
len(df)*100))
print("\nPercentage of reviews with negative sentiment :
{:.2f}%"\
      .format(df[df['Rating']<3]["Reviews"].count()/
len(df)*100))

# Plot distribution of rating
plt.figure(figsize=(12,8))
# sns.countplot(df['Rating'])
df['Rating'].value_counts().sort_index().plot(kind='bar')
plt.title('Distribution of Rating')
plt.xlabel('Rating')
plt.ylabel('Count')
```

```
#data preparation
df = df.sample(frac=0.1, random_state=0) #uncomment to
use full set of data

# Drop missing values
df.dropna(inplace=True)

# Remove any 'neutral' ratings equal to 3
df = df[df['Rating'] != 3]

# Encode 4s and 5s as 1 (positive sentiment) and 1s and
2s as 0 (negative sentiment)
df['Sentiment'] = np.where(df['Rating'] > 3, 1, 0)
df.head()


# Split data into training set and validation
X_train, X_test, y_train, y_test =
train_test_split(df['Reviews'], df['Sentiment'], \

test_size=0.1, random_state=0)

print('Load %d training examples and %d validation
examples. \n' %(X_train.shape[0],X_test.shape[0]))
print('Show a review in the training set : \n',
X_train.iloc[10])

def cleanText(raw_text, remove_stopwords=False,
stemming=False, split_text=False, \
              ):
    '''
    Convert a raw review to a cleaned review
    '''
    text = BeautifulSoup(raw_text, 'lxml').get_text()
#remove html
    letters_only = re.sub("[^a-zA-Z]", " ", text)  #
remove non-character
    words = letters_only.lower().split() # convert to
lower case

    if remove_stopwords: # remove stopword
        stops = set(stopwords.words("english"))
```

```
        words = [w for w in words if not w in stops]

    if stemming==True: # stemming
#         stemmer = PorterStemmer()
        stemmer = SnowballStemmer('english')
        words = [stemmer.stem(w) for w in words]

    if split_text==True:  # split text
        return (words)

    return( " ".join(words))

# Preprocess text data in training set and validation set
X_train_cleaned = []
X_test_cleaned = []

for d in X_train:
    X_train_cleaned.append(cleanText(d))
print('Show a cleaned review in the training set : \n',
X_train_cleaned[10])

for d in X_test:
    X_test_cleaned.append(cleanText(d))

#countvectorizer with Multinomial Naive Bayes
# Fit and transform the training data to a document-term
matrix using CountVectorizer
countVect = CountVectorizer()
X_train_countVect =
countVect.fit_transform(X_train_cleaned)
print("Number of features : %d \n"
%len(countVect.get_feature_names())) #6378
print("Show some feature names : \n",
countVect.get_feature_names()[::1000])


# Train MultinomialNB classifier
mnb = MultinomialNB()
mnb.fit(X_train_countVect, y_train)

def modelEvaluation(predictions):
    '''
```

```
    Print model evaluation to predicted result
    '''
    print ("\nAccuracy on validation set:
{:.4f}".format(accuracy_score(y_test, predictions)))
    print("\nAUC score :
{:.4f}".format(roc_auc_score(y_test, predictions)))
    print("\nClassification report : \n",
metrics.classification_report(y_test, predictions))
    print("\nConfusion Matrix : \n",
metrics.confusion_matrix(y_test, predictions))

# Evaluate the model on validaton set
predictions =
mnb.predict(countVect.transform(X_test_cleaned))
modelEvaluation(predictions)

# Fit and transform the training data to a document-term
matrix using TfidfVectorizer
tfidf = TfidfVectorizer(min_df=5) #minimum document
frequency of 5
X_train_tfidf = tfidf.fit_transform(X_train)
print("Number of features : %d \n"
%len(tfidf.get_feature_names())) #1722
print("Show some feature names : \n",
tfidf.get_feature_names()[::1000])

# Logistic Regression
lr = LogisticRegression()
lr.fit(X_train_tfidf, y_train)

# Look at the top 10 features with smallest and the
largest coefficients
feature_names = np.array(tfidf.get_feature_names())
sorted_coef_index = lr.coef_[0].argsort()
print('\nTop 10 features with smallest coefficients :\n{}
\n'.format(feature_names[sorted_coef_index[:10]]))
print('Top 10 features with largest coefficients :
\n{}'.format(feature_names[sorted_coef_index[:-11:-1]]))

# Evaluate on the validaton set
predictions = lr.predict(tfidf.transform(X_test_cleaned))
modelEvaluation(predictions)
```

```
# Building a pipeline
estimators = [("tfidf", TfidfVectorizer()), ("lr",
LogisticRegression())]
model = Pipeline(estimators)


# Grid search
params = {"lr__C":[0.1, 1, 10], #regularization param of
logistic regression
          "tfidf__min_df": [1, 3], #min count of words
          "tfidf__max_features": [1000, None], #max
features
          "tfidf__ngram_range": [(1,1), (1,2)], #1-grams
or 2-grams
          "tfidf__stop_words": [None, "english"]} #use
stopwords or don't

grid = GridSearchCV(estimator=model, param_grid=params,
scoring="accuracy", n_jobs=-1)
grid.fit(X_train_cleaned, y_train)
print("The best paramenter set is : \n",
grid.best_params_)


# Evaluate on the validaton set
predictions = grid.predict(X_test_cleaned)
modelEvaluation(predictions)


top_words = 20000
maxlen = 100
batch_size = 32
nb_classes = 2
epochs = 3


# Vectorize X_train and X_test to 2D tensor
tokenizer = Tokenizer(num_words=top_words) #only consider
top 20000 words in the corpse
tokenizer.fit_on_texts(X_train)
```

```
# tokenizer.word_index #access word-to-index dictionary
of trained tokenizer

sequences_train = tokenizer.texts_to_sequences(X_train)
sequences_test = tokenizer.texts_to_sequences(X_test)

X_train_seq = sequence.pad_sequences(sequences_train,
maxlen=maxlen)
X_test_seq = sequence.pad_sequences(sequences_test,
maxlen=maxlen)


# one-hot encoding of y_train and y_test
y_train_seq = np_utils.to_categorical(y_train,
nb_classes)
y_test_seq = np_utils.to_categorical(y_test, nb_classes)

print('X_train shape:', X_train_seq.shape) #(27799, 100)
print('X_test shape:', X_test_seq.shape) #(3089, 100)
print('y_train shape:', y_train_seq.shape) #(27799, 2)
print('y_test shape:', y_test_seq.shape) #(3089, 2)


# Construct a simple LSTM
model1 = Sequential()
model1.add(Embedding(top_words, 128,
input_length=maxlen))
model1.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model1.add(Dense(nb_classes))
model1.add(Activation('softmax'))
model1.summary()

# Compile LSTM
model1.compile(loss='binary_crossentropy',
               optimizer='adam',
               metrics=['accuracy'])

model1.fit(X_train_seq, y_train_seq,
batch_size=batch_size, epochs=epochs, verbose=1)

# Model evluation
```

```
score = model1.evaluate(X_test_seq, y_test_seq,
batch_size=batch_size)
print('Test loss : {:.4f}'.format(score[0]))
print('Test accuracy : {:.4f}'.format(score[1]))
# get weight matrix of the embedding layer
model1.layers[0].get_weights()[0] # weight matrix of the
embedding layer, word-by-dim matrix
print("Size of weight matrix in the embedding layer : ",
\
      model1.layers[0].get_weights()[0].shape) #(20000,
128)

# get weight matrix of the hidden layer
print("Size of weight matrix in the hidden layer : ", \
      model1.layers[1].get_weights()[0].shape) #(128,
512)  weight dim of LSTM - w

# get weight matrix of the output layer
print("Size of weight matrix in the output layer : ", \
      model1.layers[2].get_weights()[0].shape) #(128, 2)
weight dim of dense layer
```