ICP7

1.

```
!pip install tensorflow==2.12.0
!pip install keras==2.12.0
!pip install keras.utils
```

```
# Mounting Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

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

```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
import re

from sklearn.preprocessing import LabelEncoder

data = pd.read_csv('/content/drive/My Drive/Sentiment.csv')
# Keeping only the neccessary columns
data = data[['text','sentiment']]

data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))

for idx, row in data.iterrows():
    row[0] = row[0].replace('rt', ' ')
```

```
max_features = 2000
tokenizer = Tokenizer(num_words=max_features, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)

X = pad_sequences(X)

embed_dim = 128
lstm_out = 196
def createmodel():
    model = Sequential()
    model.add(Embedding(max_features, embed_dim,input_length = X.shape[1]))
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(3,activation='softmax'))
    model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])
    return model
# print(model.summary())

labelencoder = LabelEncoder()
integer_encoded = labelencoder.fit_transform(data['sentiment'])
y = to_categorical(integer_encoded)
X_train, X_test, Y_train, Y_test = train_test_split(X,y, test_size = 0.33, random_state = 42)

batch_size = 32
model = createmodel()
model.fit(X_train, Y_train, epochs = 1, batch_size=batch_size, verbose = 2)
score,acc = model.evaluate(X_test,Y_test,verbose=2,batch_size=batch_size)
print(score)
print(acc)
```

```
print(model.metrics_names)
```

```
<ipython-input-5-79347c4597c4>:21: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future
  row[0] = row[0].replace('rt', ' ')
<ipython-input-5-79347c4597c4>:21: FutureWarning: Series.__setitem__ treating keys as positions is deprecated. In a future
  row[0] = row[0].replace('rt', ' ')
291/291 - 50s - loss: 0.8268 - accuracy: 0.6403 - 50s/epoch - 171ms/step
144/144 - 3s - loss: 0.7453 - accuracy: 0.6752 - 3s/epoch - 20ms/step
0.745284914970398
0.6751856803894043
['loss', 'accuracy']
```

```
model.save('sentimentAnalysis.h5')
```

```
from keras.models import load_model
import numpy as np

loaded_model = load_model('sentimentAnalysis.h5')

new_text = ["A lot of good things are happening. We are respected again throughout the world, and that's a great thing.@realDonaldTrump"]
new_text = tokenizer.texts_to_sequences(new_text)
new_text = pad_sequences(new_text, maxlen=X.shape[1], dtype='int32', value=0)
sentiment_pro = loaded_model.predict(new_text, batch_size=1, verbose=2)[0]

sentiment_classes = ['Positive', 'Neutral', 'Negative']
sentiment_pre = sentiment_classes[np.argmax(sentiment_prob)]

print("Predicted sentiment: ", sentiment_pre)
print("Predicted probabilities: ", sentiment_pro)
```

```
1/1 - 1s - 809ms/epoch - 809ms/step
Predicted sentiment:  Positive
Predicted probabilities:  [0.44116956 0.16455497 0.39427555]
```

2.

```python
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.layers import LSTM

# Function to create the model, as it's required by KerasClassifier
def create_model(lstm_out=196, dropout=0.2):
    model = Sequential()
    model.add(Embedding(max_features, embed_dim, input_length=X.shape[1]))
    model.add(LSTM(lstm_out, dropout=dropout, recurrent_dropout=dropout))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

# Create the KerasClassifier
model = KerasClassifier(build_fn=create_model, verbose=0)

batch_size_1 = [10, 20, 40]
epochs_1 = [1, 2, 3]

# Define the grid of parameters to search
param_grid = dict(batch_size=batch_size_1, epochs=epochs_1)

# Create GridSearchCV
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(X_train, Y_train)

# Summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
<ipython-input-13-3e27ad9c23bd>:15: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (
  model = KerasClassifier(build_fn=create_model, verbose=0)
Best: 0.676638 using {'batch_size': 40, 'epochs': 2}
```

GitHub link: https://github.com/amanmushnam/BDA.git