ICP-10

VVS Murthy kolla
700729142

Video link:
https://drive.google.com/file/d/1blSzZqThUkV__sHQsI1OVy2F-ZR_W79j/view?usp=share_link
Git Hub link: https://github.com/murthykolla/ICP-10.git

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
#Tokenization
from keras.preprocessing.text import Tokenizer
 #Adding zeros or crop based on the length
from tensorflow.keras.preprocessing.sequence import pad_sequences
#Sequential Neural Network
from keras.models import Sequential
 #For layers in Neural Network
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from keras.utils.np_utils import to_categorical
```

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

```
Mounted at /content/gdrive
```

```python
import pandas as pd

# Loading the dataset as a Pandas DataFrame
dataset = pd.read_csv('/content/gdrive/MyDrive/NN drive/Sentiment.csv')

# Selecting only the necessary columns 'text' and 'sentiment' from the dataset
mask = dataset.columns.isin(['text', 'sentiment'])
data = dataset.loc[:, mask]

# Keeping only the necessary columns in the dataset
data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))
```

```
<ipython-input-7-d0e745dc69e5>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['text'] = data['text'].apply(lambda x: x.lower())
<ipython-input-7-d0e745dc69e5>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))
```

```python
for idx, row in data.iterrows():
  #Removing all retweets
    row[0] = row[0].replace('rt', ' ')
    max_fatures = 2000
     #spliting the sentence to max 2000 words
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
 #obstracting values from the futire matrix
X = tokenizer.texts_to_sequences(data['text'].values)
#Padding the feature matrix
X = pad_sequences(X)
#Dimension of the Embedded layer
embed_dim = 128
 #Long short-term memory (LSTM) layer neurons
lstm_out = 196
def createmodel():
   #Sequential Neural Network
    model = Sequential()
     #input dimension 2000 Neurons, output dimension 128 Neurons
    model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1]))
     #Drop out 20%, 196 output Neurons, recurrent dropout 20%
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
    #3 output neurons[positive, Neutral, Negative], softmax as activation
    model.add(Dense(3,activation='softmax'))
    model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])
    return model
# printing(model.summary())
labelencoder = LabelEncoder()
 #model fitting
integer_encoded = labelencoder.fit_transform(data['sentiment'])
y = to_categorical(integer_encoded)
#67% as training data, 33% as test data split
X_train, X_test, Y_train, Y_test = train_test_split(X,y, test_size = 0.33, random_state = 42)
#mentioning batch size as 32
batch_size = 32
#Functioning the  call to Sequential Neural Network
model = createmodel()
```

```
model = createmodel()
 #verbose the higher, the more messages
model.fit(X_train, Y_train, epochs = 1, batch_size=batch_size, verbose = 2)
 #evaluting the model as requried
score,acc = model.evaluate(X_test,Y_test,verbose=2,batch_size=batch_size)
print(score)
print(acc)
```

```
291/291 - 42s - loss: 0.8306 - accuracy: 0.6441 - 42s/epoch - 144ms/step
144/144 - 3s - loss: 0.7514 - accuracy: 0.6791 - 3s/epoch - 22ms/step
0.7513718008995056
0.6791175007820129
```

```
#model metric evaluting
print(model.metrics_names)
```

```
['loss', 'accuracy']
```

```
#1. Save the model and use the saved model to predict on new text data (ex, “A lot of good things are happening. We are respected again throughout the world
```

```
#save model for future use
model.save('sentimentAnalysis.h5')
```

```
from keras.models import load_model
model= load_model('sentimentAnalysis.h5')
```

```
print(integer_encoded)
print(data['sentiment'])
```

```
[1 2 1 ... 2 0 2]
0         Neutral
1        Positive
2         Neutral
3        Positive
4        Positive
           ...
13866    Negative
13867    Positive
13868    Positive
13869    Negative
13870    Positive
Name: sentiment, Length: 13871, dtype: object
```

```
# Predicting on the text data
sentence = ['A lot of good things are happening. We are respected again throughout the world, and that is a great thing.@realDonaldTrump']
sentence = tokenizer.texts_to_sequences(sentence) # Tokenizing the sentence
sentence = pad_sequences(sentence, maxlen=28, dtype='int32', value=0) # Padding the sentence
sentiment_probs = model.predict(sentence, batch_size=1, verbose=2)[0] # Predicting the sentence text
sentiment = np.argmax(sentiment_probs)

print(sentiment_probs)
if sentiment == 0:
    print("Neutral")
elif sentiment < 0:
    print("Negative")
elif sentiment > 0:
    print("Positive")
else:
    print("Cannot be determined")
```

```
1/1 - 0s - 270ms/epoch - 270ms/step
[0.72844136 0.10584743 0.16571125]
Neutral
```

ly GridSearchCV on the source code provided in the ***class***

```
#importing Keras classifier
from keras.wrappers.scikit_learn import KerasClassifier
#importing Grid search CV
from sklearn.model_selection import GridSearchCV
#initiating model to test performance by applying multiple hyper parameters
model = KerasClassifier(build_fn=createmodel,verbose=2)
#hyper parameter batch_size
batch_size= [10, 20, 40]
#hyper parameter no. of epochs
epochs = [1, 2]
#creating dictionary for batch size, no. of epochs
param_grid= {'batch_size':batch_size, 'epochs':epochs}
#Applying dictionary with hyper parameters
grid  = GridSearchCV(estimator=model, param_grid=param_grid)
 #Fitting the model
grid_result= grid.fit(X_train,Y_train)
# summarize results
#best score, best hyper parameters
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Epoch 2/2
372/372 - 46s - loss: 0.6790 - accuracy: 0.7088 - 46s/epoch - 124ms/step
93/93 - 2s - loss: 0.7583 - accuracy: 0.6600 - 2s/epoch - 18ms/step
Epoch 1/2
372/372 - 49s - loss: 0.8330 - accuracy: 0.6394 - 49s/epoch - 132ms/step
```

```
372/372 - 45s - loss: 0.6831 - accuracy: 0.7144 - 45s/epoch - 120ms/step
93/93 - 3s - loss: 0.7555 - accuracy: 0.6837 - 3s/epoch - 28ms/step
Epoch 1/2
372/372 - 54s - loss: 0.8312 - accuracy: 0.6424 - 54s/epoch - 146ms/step
Epoch 2/2
372/372 - 50s - loss: 0.6755 - accuracy: 0.7126 - 50s/epoch - 135ms/step
93/93 - 2s - loss: 0.7513 - accuracy: 0.6717 - 2s/epoch - 19ms/step
Epoch 1/2
372/372 - 49s - loss: 0.8253 - accuracy: 0.6475 - 49s/epoch - 132ms/step
Epoch 2/2
372/372 - 46s - loss: 0.6669 - accuracy: 0.7196 - 46s/epoch - 125ms/step
93/93 - 2s - loss: 0.7966 - accuracy: 0.6561 - 2s/epoch - 17ms/step
186/186 - 30s - loss: 0.8402 - accuracy: 0.6375 - 30s/epoch - 163ms/step
47/47 - 1s - loss: 0.7865 - accuracy: 0.6374 - 1s/epoch - 23ms/step
186/186 - 33s - loss: 0.8433 - accuracy: 0.6355 - 33s/epoch - 180ms/step
47/47 - 1s - loss: 0.7775 - accuracy: 0.6713 - 1s/epoch - 26ms/step
186/186 - 32s - loss: 0.8462 - accuracy: 0.6342 - 32s/epoch - 169ms/step
47/47 - 2s - loss: 0.7659 - accuracy: 0.6789 - 2s/epoch - 39ms/step
186/186 - 31s - loss: 0.8494 - accuracy: 0.6336 - 31s/epoch - 164ms/step
47/47 - 1s - loss: 0.7577 - accuracy: 0.6787 - 1s/epoch - 24ms/step
186/186 - 33s - loss: 0.8412 - accuracy: 0.6383 - 33s/epoch - 179ms/step
47/47 - 1s - loss: 0.7749 - accuracy: 0.6642 - 1s/epoch - 26ms/step
Epoch 1/2
186/186 - 31s - loss: 0.8417 - accuracy: 0.6414 - 31s/epoch - 167ms/step
Epoch 2/2
186/186 - 30s - loss: 0.6924 - accuracy: 0.7037 - 30s/epoch - 161ms/step
47/47 - 1s - loss: 0.7302 - accuracy: 0.6832 - 1s/epoch - 28ms/step
Epoch 1/2
186/186 - 31s - loss: 0.8377 - accuracy: 0.6377 - 31s/epoch - 166ms/step
Epoch 2/2
186/186 - 27s - loss: 0.6905 - accuracy: 0.7086 - 27s/epoch - 148ms/step
47/47 - 2s - loss: 0.7384 - accuracy: 0.6826 - 2s/epoch - 41ms/step
Epoch 1/2
186/186 - 31s - loss: 0.8403 - accuracy: 0.6391 - 31s/epoch - 168ms/step
Epoch 2/2
186/186 - 29s - loss: 0.6859 - accuracy: 0.7066 - 29s/epoch - 153ms/step
```

```
Epoch 2/2
186/186 - 29s - loss: 0.6859 - accuracy: 0.7066 - 29s/epoch - 153ms/step
47/47 - 2s - loss: 0.7515 - accuracy: 0.6724 - 2s/epoch - 42ms/step
Epoch 1/2
186/186 - 31s - loss: 0.8492 - accuracy: 0.6293 - 31s/epoch - 164ms/step
Epoch 2/2
186/186 - 28s - loss: 0.6843 - accuracy: 0.7050 - 28s/epoch - 150ms/step
47/47 - 2s - loss: 0.7519 - accuracy: 0.6787 - 2s/epoch - 41ms/step
Epoch 1/2
186/186 - 30s - loss: 0.8361 - accuracy: 0.6401 - 30s/epoch - 160ms/step
Epoch 2/2
186/186 - 27s - loss: 0.6828 - accuracy: 0.7119 - 27s/epoch - 148ms/step
47/47 - 3s - loss: 0.7860 - accuracy: 0.6625 - 3s/epoch - 58ms/step
Epoch 1/2
233/233 - 40s - loss: 0.8312 - accuracy: 0.6396 - 40s/epoch - 170ms/step
Epoch 2/2
233/233 - 37s - loss: 0.6839 - accuracy: 0.7096 - 37s/epoch - 158ms/step
Best: 0.675884 using {'batch_size': 40, 'epochs': 2}
```