NEURAL NETWORKS & DEEP LEARNING: ICP5

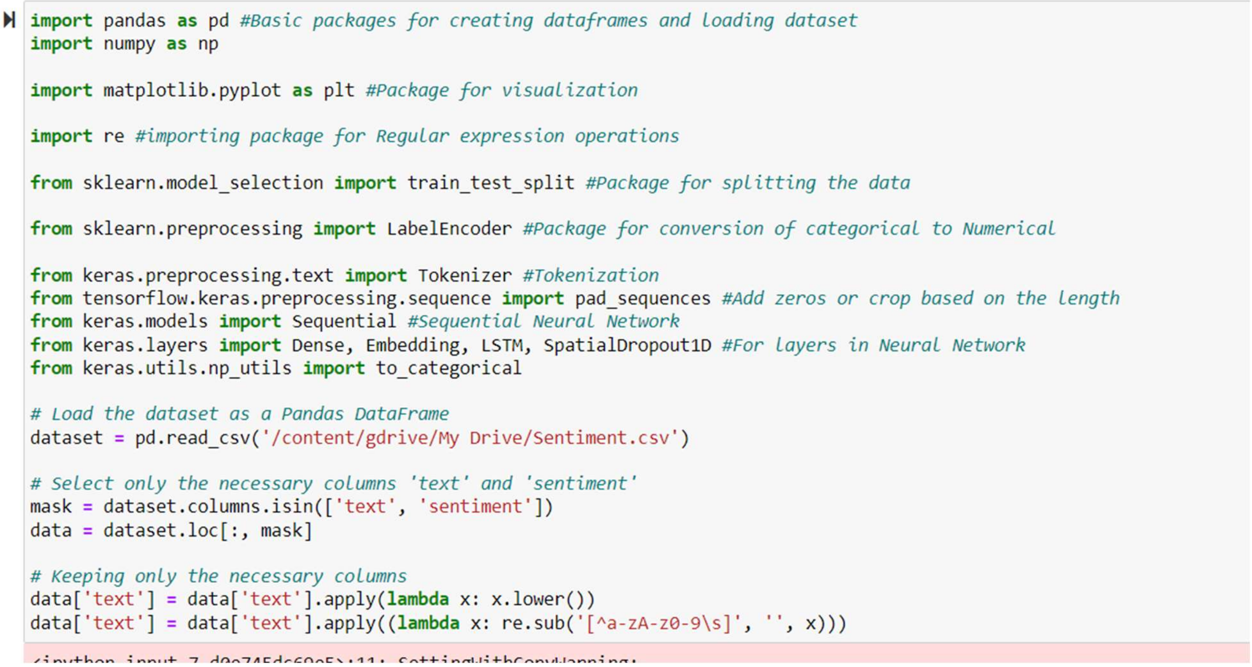
Name: Reshma Maddala

ID: 700740808

GITHUB LINK: https://github.com/maddalareshma/NNDL-ICP5

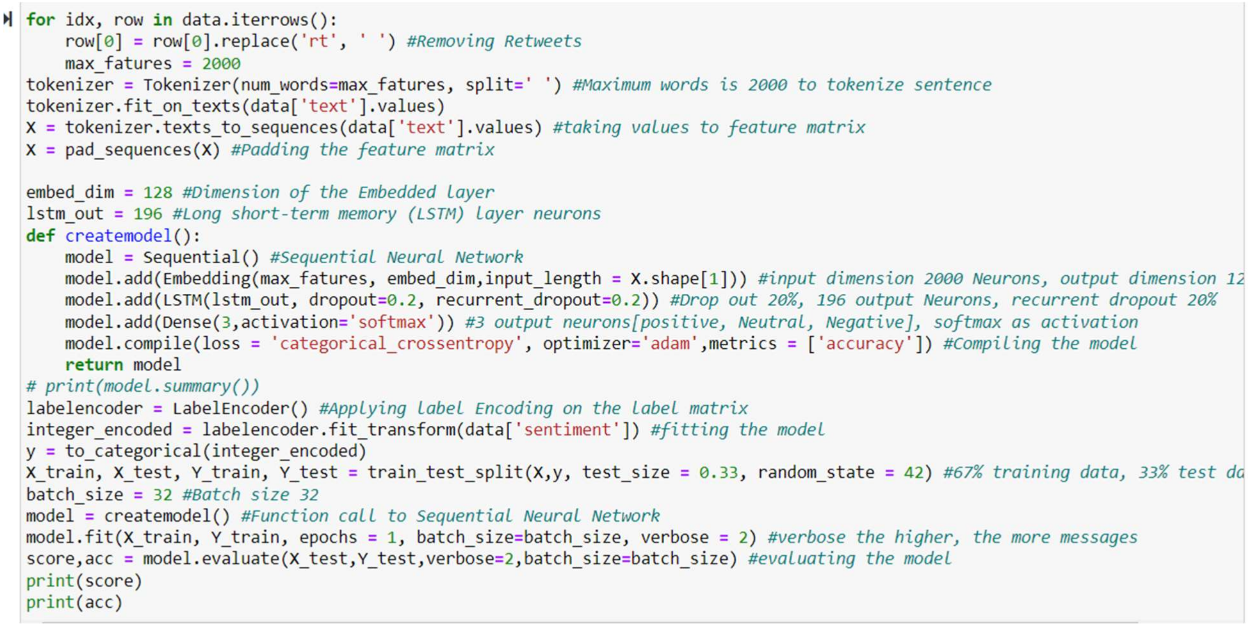
1. Save the model and use the saved model to predict on new text data (ex, “A lot of good things are happening. We are respected again throughout the world, and that's a great thing.@realDonaldTrump”)

2. Apply GridSearchCV on the source code provided in the class

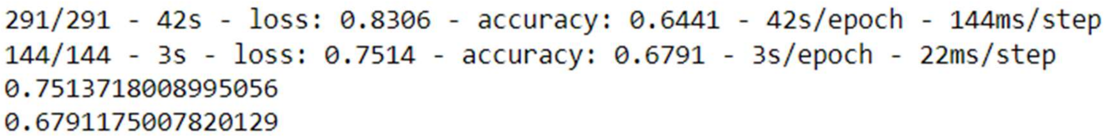


```
import pandas as pd #Basic packages for creating dataframes and loading dataset
import numpy as np

import matplotlib.pyplot as plt #Package for visualization

import re #importing package for Regular expression operations

from sklearn.model_selection import train_test_split #Package for splitting the data

from sklearn.preprocessing import LabelEncoder #Package for conversion of categorical to Numerical

from keras.preprocessing.text import Tokenizer #Tokenization
from tensorflow.keras.preprocessing.sequence import pad_sequences #Add zeros or crop based on the length
from keras.models import Sequential #Sequential Neural Network
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D #For layers in Neural Network
from keras.utils.np_utils import to_categorical

# Load the dataset as a Pandas DataFrame
dataset = pd.read_csv('/content/gdrive/My Drive/Sentiment.csv')

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

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

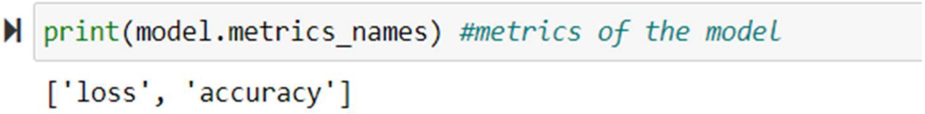
```
<ipython-input-7-d0e745dc69e5>:11: SettingWithCopyWarning:
```



```
for idx, row in data.iterrows():
    row[0] = row[0].replace('rt', ' ') #Removing Retweets
    max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ') #Maximum words is 2000 to tokenize sentence
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values) #taking values to feature matrix
X = pad_sequences(X) #Padding the feature matrix

embed_dim = 128 #Dimension of the Embedded layer
lstm_out = 196 #Long short-term memory (LSTM) layer neurons
def createmodel():
    model = Sequential() #Sequential Neural Network
    model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1])) #input dimension 2000 Neurons, output dimension 12
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2)) #Drop out 20%, 196 output Neurons, recurrent dropout 20%
    model.add(Dense(3,activation='softmax')) #3 output neurons[positive, Neutral, Negative], softmax as activation
    model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy']) #Compiling the model
    return model
# print(model.summary())
labelencoder = LabelEncoder() #Applying label Encoding on the label matrix
integer_encoded = labelencoder.fit_transform(data['sentiment']) #fitting the model
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) #67% training data, 33% test da
batch_size = 32 #Batch size 32
model = createmodel() #Function call to Sequential Neural Network
model.fit(X_train, Y_train, epochs = 1, batch_size=batch_size, verbose = 2) #verbose the higher, the more messages
score,acc = model.evaluate(X_test,Y_test,verbose=2,batch_size=batch_size) #evaluating the model
print(score)
print(acc)
```

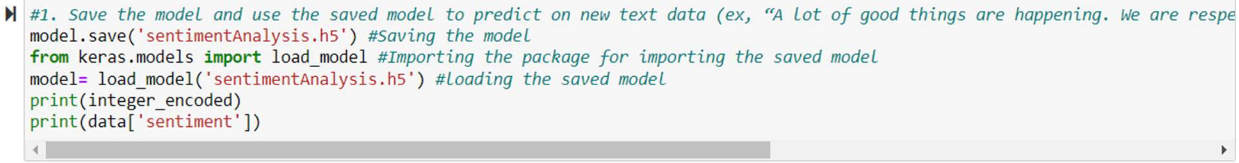


```
291/291 - 42s - loss: 0.8306 - accuracy: 0.6441 - 42s/epoch - 144ms/step
144/144 - 3s - loss: 0.7514 - accuracy: 0.6791 - 3s/epoch - 22ms/step
0.7513718008995056
0.6791175007820129
```

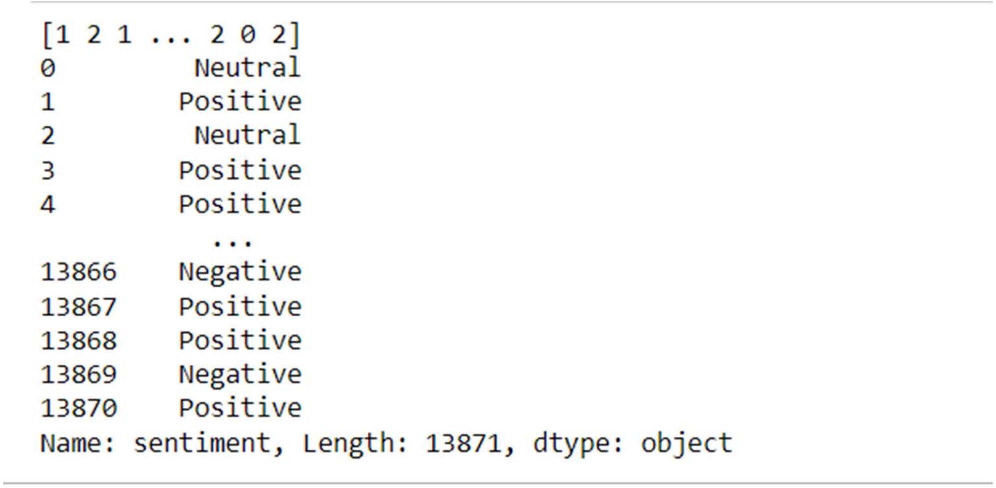


```
print(model.metrics_names) #metrics of the model
```

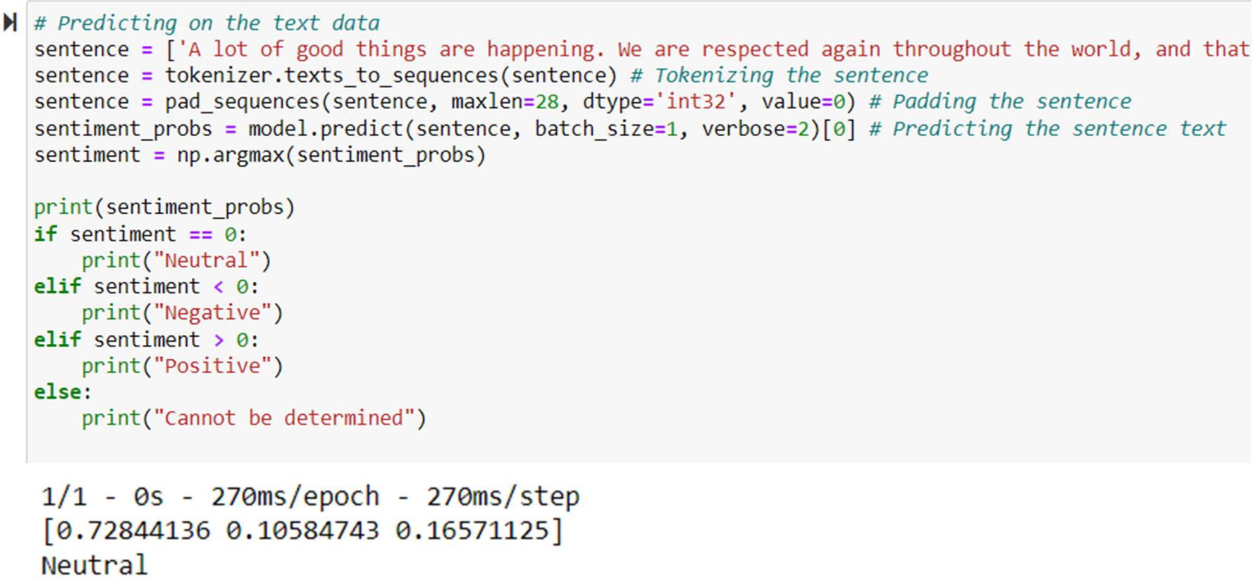
```
['loss', 'accuracy']
```



```
#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 respe
model.save('sentimentAnalysis.h5') #Saving the model
from keras.models import load_model #Importing the package for importing the saved model
model= load_model('sentimentAnalysis.h5') #loading the saved model
print(integer_encoded)
print(data['sentiment'])
```

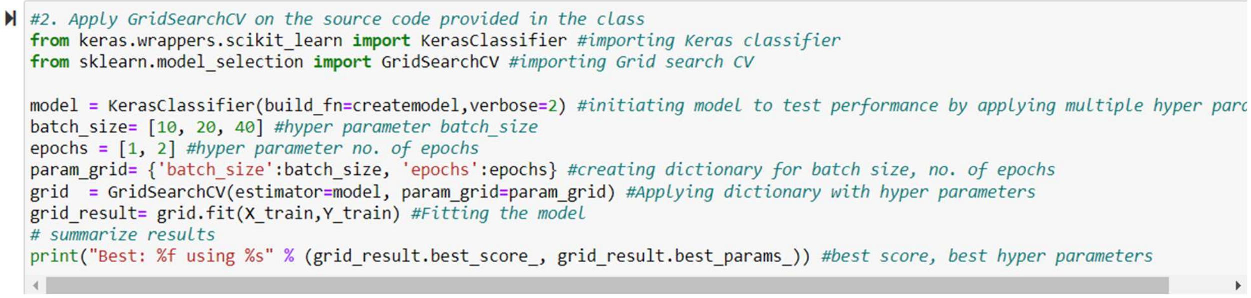


```
[1 2 1 ... 2 0 2]
0         Neutral
1        Positive
2         Neutral
3        Positive
4        Positive
           ...
13866    Negative
13867    Positive
13868    Positive
13869    Negative
13870    Positive
Name: sentiment, Length: 13871, dtype: object
```

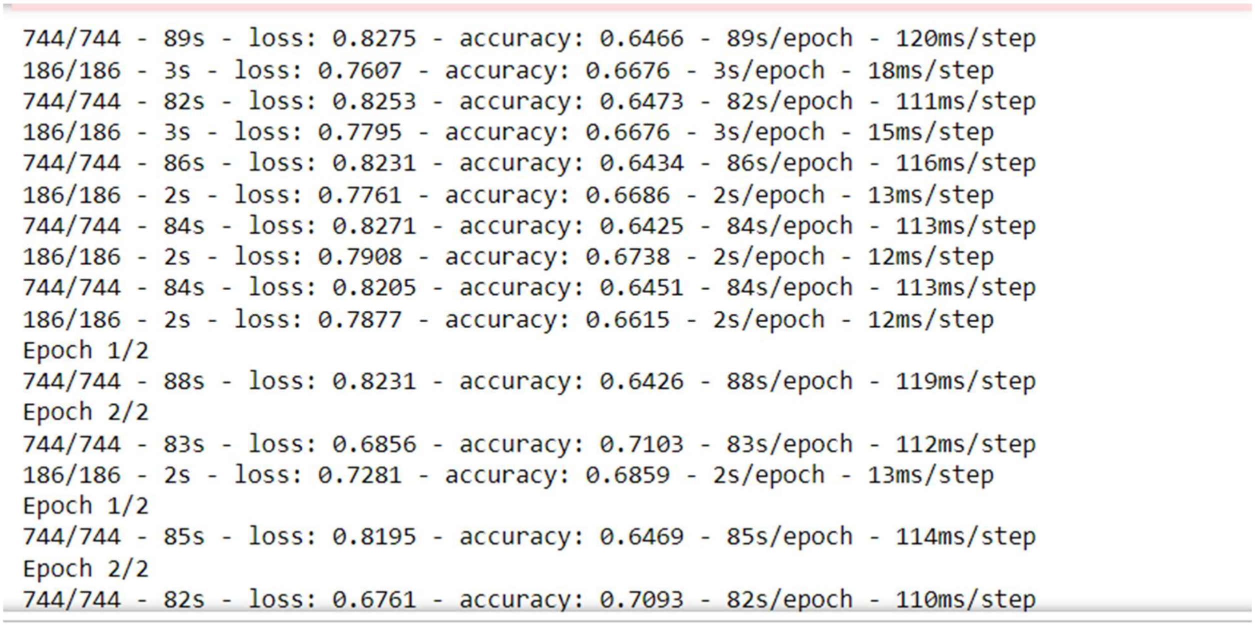


```python
# Predicting on the text data
sentence = ['A lot of good things are happening. We are respected again throughout the world, and that
sentence = tokenizer.texts_to_sequences(sentence) # Tokenizing the sentence
sentence = pad_sequences(sentence, maxlen=28, dtype='int32', value=0) # Padding the 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
```



```python
#2. Apply GridSearchCV on the source code provided in the class
from keras.wrappers.scikit_learn import KerasClassifier #importing Keras classifier
from sklearn.model_selection import GridSearchCV #importing Grid search CV

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



```
744/744 - 89s - loss: 0.8275 - accuracy: 0.6466 - 89s/epoch - 120ms/step
186/186 - 3s - loss: 0.7607 - accuracy: 0.6676 - 3s/epoch - 18ms/step
744/744 - 82s - loss: 0.8253 - accuracy: 0.6473 - 82s/epoch - 111ms/step
186/186 - 3s - loss: 0.7795 - accuracy: 0.6676 - 3s/epoch - 15ms/step
744/744 - 86s - loss: 0.8231 - accuracy: 0.6434 - 86s/epoch - 116ms/step
186/186 - 2s - loss: 0.7761 - accuracy: 0.6686 - 2s/epoch - 13ms/step
744/744 - 84s - loss: 0.8271 - accuracy: 0.6425 - 84s/epoch - 113ms/step
186/186 - 2s - loss: 0.7908 - accuracy: 0.6738 - 2s/epoch - 12ms/step
744/744 - 84s - loss: 0.8205 - accuracy: 0.6451 - 84s/epoch - 113ms/step
186/186 - 2s - loss: 0.7877 - accuracy: 0.6615 - 2s/epoch - 12ms/step
Epoch 1/2
744/744 - 88s - loss: 0.8231 - accuracy: 0.6426 - 88s/epoch - 119ms/step
Epoch 2/2
744/744 - 83s - loss: 0.6856 - accuracy: 0.7103 - 83s/epoch - 112ms/step
186/186 - 2s - loss: 0.7281 - accuracy: 0.6859 - 2s/epoch - 13ms/step
Epoch 1/2
744/744 - 85s - loss: 0.8195 - accuracy: 0.6469 - 85s/epoch - 114ms/step
Epoch 2/2
744/744 - 82s - loss: 0.6761 - accuracy: 0.7093 - 82s/epoch - 110ms/step
```

The create_model function, which returns a Keras model with the provided hyperparameters, is defined in this code. The create_model function is wrapped by the KerasClassifier class so that it may be utilized by GridSearchCV as an estimator. The param_grid dictionary contains definitions for the hyperparameters that must be tuned. Then, using the KerasClassifier object and the param_grid dictionary, GridSearchCV is invoked.