Name:konyala sai vardhan reddy

Student id:700745733

Github link: https://github.com/vardhan141/deeplearningicp5

Video link: https://drive.google.com/file/d/1skER2TveTutL3hS3sv2ONmnCfqL92hHv/view?usp=sharing

ICP5

First we import all the packages required for implementing the model and reading the dataset

```
import pandas as pd # packages for creating dataframes and loading dataset
import numpy as np
# package for plotting
import matplotlib.pyplot as plt
# for regular expressions
import re
# for implementing machine learning functions
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder
# for deep learning models and functions
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from keras.utils.np_utils import to_categorical
```

We read the data and select only the necessary columns such as text and sentiment

```
[ ] import pandas as pd
    # Load the dataset as a Pandas DataFrame
    dataset = pd.read_csv('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-3-cee1da567eb8>:1: 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-3-cee1da567eb8>:2: 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)))
```

Here we remove the rt word from tweets and create a sequential neural network model and evaluate and print the accuracy of the model

```
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 128 Neurons
    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()
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 #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 modelprint(score)
print(acc)
```

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”)

```
[ ] 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
```

Here we save the model into file name called sentimentAnalysis.h5

And then load the model into model variable

```
# 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 - 1s - 507ms/epoch - 507ms/step
[0.65985817 0.09624108 0.2439008 ]
Neutral
```

Then we predict on the text data which we got as neutral

`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 parameters
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
```

```
372/372 - 53s - loss: 0.6842 - accuracy: 0.7093 - 53s/epoch - 142ms/step
93/93 - 2s - loss: 0.7401 - accuracy: 0.6815 - 2s/epoch - 20ms/step
Epoch 1/2
372/372 - 58s - loss: 0.8397 - accuracy: 0.6376 - 58s/epoch - 155ms/step
Epoch 2/2
372/372 - 53s - loss: 0.6847 - accuracy: 0.7109 - 53s/epoch - 143ms/step
93/93 - 2s - loss: 0.7354 - accuracy: 0.6842 - 2s/epoch - 26ms/step
Epoch 1/2
186/186 - 31s - loss: 0.6977 - accuracy: 0.7012 - 31s/epoch - 165ms/step
47/47 - 2s - loss: 0.7360 - accuracy: 0.6772 - 2s/epoch - 43ms/step
Epoch 1/2
186/186 - 36s - loss: 0.8427 - accuracy: 0.6333 - 36s/epoch - 196ms/step
Epoch 2/2
186/186 - 33s - loss: 0.6837 - accuracy: 0.7117 - 33s/epoch - 177ms/step
47/47 - 1s - loss: 0.7399 - accuracy: 0.6832 - 1s/epoch - 29ms/step
Epoch 1/2
186/186 - 35s - loss: 0.8396 - accuracy: 0.6359 - 35s/epoch - 190ms/step
Epoch 2/2
186/186 - 33s - loss: 0.6828 - accuracy: 0.7130 - 33s/epoch - 176ms/step
47/47 - 1s - loss: 0.7470 - accuracy: 0.6722 - 1s/epoch - 29ms/step
Epoch 1/2
186/186 - 35s - loss: 0.8440 - accuracy: 0.6352 - 35s/epoch - 190ms/step
Epoch 2/2
186/186 - 32s - loss: 0.6799 - accuracy: 0.7108 - 32s/epoch - 173ms/step
47/47 - 1s - loss: 0.7818 - accuracy: 0.6781 - 1s/epoch - 29ms/step
Epoch 1/2
465/465 - 72s - loss: 0.8116 - accuracy: 0.6428 - 72s/epoch - 155ms/step
Epoch 2/2
465/465 - 68s - loss: 0.6727 - accuracy: 0.7176 - 68s/epoch - 146ms/step
Best: 0.681373 using {'batch_size': 20, 'epochs': 2}
```

We got the best accuracy for the 0.681374 and epochs 2