BODIGE SAINATH

700756620

ICP – 7

Git – https://github.com/BodigeSainath/icp7

Video - https://drive.google.com/file/d/1Ko6Wpj1ls61_8usfNE1iWgawxbon6X3q/view?usp=sharing

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")

```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from keras.preprocessing.text import Tokenizer
from 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 tensorflow.keras.utils import to_categorical
import re

from sklearn.preprocessing import LabelEncoder

data = pd.read_csv('/content/Sentiment (3).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_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, 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_fatures, 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)
```

```
291/291 - 48s - loss: 0.8208 - accuracy: 0.6428 - 48s/epoch - 166ms/step
144/144 - 4s - loss: 0.7668 - accuracy: 0.6614 - 4s/epoch - 31ms/step
0.7668231725692749
0.6614242196083069
['loss', 'accuracy']
```

```
import tweepy
from keras.models import load_model
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import re

# Load the saved model
model = load_model("/content/sentiment_model.h5")

# Define a function for preprocessing text
def preprocess_text(text):
    text = text.lower()
    text = re.sub('[^a-zA-z0-9\s]', '', text)
    return text

# Example new text data
```

```
new_text = "A lot of good things are happening. We are respected again
throughout the world, and that's a great thing. @realDonaldTrump"

# Preprocess the new text data
new_text = preprocess_text(new_text)

# Tokenize and pad the new text data
max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts([new_text])
X_new = tokenizer.texts_to_sequences([new_text])
X_new = pad_sequences(X_new, maxlen=model.input_shape[1])

# Make predictions
predictions = model.predict(X_new)

# Determine the sentiment based on the prediction
sentiments = ['Negative', 'Neutral', 'Positive']
predicted_sentiment = sentiments[predictions.argmax()]

# Print the result
print("Predicted Sentiment: " + predicted_sentiment)
```

```
1/1 [==============================] - 0s 296ms/step
Predicted Sentiment: Negative
```

---

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

```
!pip install scikeras
```

```
Collecting scikeras
  Downloading scikeras-0.12.0-py3-none-any.whl (27 kB)
Requirement already satisfied: packaging>=0.21 in
/usr/local/lib/python3.10/dist-packages (from scikeras) (24.0)
Requirement already satisfied: scikit-learn>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikeras) (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras)
(1.25.2)
Requirement already satisfied: scipy>=1.3.2 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras)
(1.11.4)
Requirement already satisfied: joblib>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras)
(1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras)
(3.4.0)
Installing collected packages: scikeras
Successfully installed scikeras-0.12.0
```

```python
from scikeras.wrappers import KerasClassifier

import pandas as pd
import re
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from scikeras.wrappers import KerasClassifier

# Assuming the data loading and preprocessing steps are the same

max_features = 2000
tokenizer = Tokenizer(num_words=max_features, split=' ')
# Assuming tokenizer fitting and text preprocessing is done here

def createmodel(optimizer='adam'):
    model = Sequential()
    model.add(Embedding(max_features, embed_dim, input_length=X.shape[1]))
    model.add(SpatialDropout1D(0.2))
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer,
metrics=['accuracy'])
    return model

# Define the KerasClassifier with the build_fn as our model creation function
model = KerasClassifier(model=createmodel, verbose=2)

# Define hyperparameters to tune
param_grid = {
    'batch_size': [32, 64],
    'epochs': [1, 2],
    'optimizer': ['adam', 'rmsprop']
}

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

# Summarize results
```

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

```
194/194 - 37s - loss: 0.8596 - accuracy: 0.6328 - 37s/epoch - 192ms/step
97/97 - 2s - 2s/epoch - 23ms/step
194/194 - 41s - loss: 0.8563 - accuracy: 0.6297 - 41s/epoch - 210ms/step
97/97 - 3s - 3s/epoch - 34ms/step
194/194 - 36s - loss: 0.8773 - accuracy: 0.6278 - 36s/epoch - 186ms/step
97/97 - 2s - 2s/epoch - 23ms/step
194/194 - 32s - loss: 0.8712 - accuracy: 0.6326 - 32s/epoch - 167ms/step
97/97 - 3s - 3s/epoch - 28ms/step
194/194 - 33s - loss: 0.8588 - accuracy: 0.6292 - 33s/epoch - 171ms/step
97/97 - 3s - 3s/epoch - 27ms/step
194/194 - 34s - loss: 0.8675 - accuracy: 0.6252 - 34s/epoch - 173ms/step
97/97 - 2s - 2s/epoch - 23ms/step
Epoch 1/2
194/194 - 33s - loss: 0.8632 - accuracy: 0.6300 - 33s/epoch - 171ms/step
Epoch 2/2
194/194 - 29s - loss: 0.7171 - accuracy: 0.6888 - 29s/epoch - 151ms/step
97/97 - 3s - 3s/epoch - 32ms/step
Epoch 1/2
194/194 - 33s - loss: 0.8599 - accuracy: 0.6271 - 33s/epoch - 170ms/step
Epoch 2/2
194/194 - 30s - loss: 0.6978 - accuracy: 0.6991 - 30s/epoch - 157ms/step
97/97 - 2s - 2s/epoch - 23ms/step
Epoch 1/2
194/194 - 35s - loss: 0.8553 - accuracy: 0.6285 - 35s/epoch - 179ms/step
Epoch 2/2
194/194 - 29s - loss: 0.6883 - accuracy: 0.7022 - 29s/epoch - 151ms/step
97/97 - 2s - 2s/epoch - 23ms/step
Epoch 1/2
194/194 - 35s - loss: 0.8565 - accuracy: 0.6320 - 35s/epoch - 178ms/step
Epoch 2/2
194/194 - 29s - loss: 0.7122 - accuracy: 0.6949 - 29s/epoch - 150ms/step
97/97 - 3s - 3s/epoch - 34ms/step
Epoch 1/2
194/194 - 33s - loss: 0.8660 - accuracy: 0.6295 - 33s/epoch - 168ms/step
Epoch 2/2
194/194 - 30s - loss: 0.7025 - accuracy: 0.6999 - 30s/epoch - 157ms/step
97/97 - 2s - 2s/epoch - 23ms/step
Epoch 1/2
194/194 - 35s - loss: 0.8494 - accuracy: 0.6320 - 35s/epoch - 181ms/step
Epoch 2/2
194/194 - 30s - loss: 0.6845 - accuracy: 0.7093 - 30s/epoch - 156ms/step
97/97 - 3s - 3s/epoch - 30ms/step
97/97 - 30s - loss: 0.8820 - accuracy: 0.6182 - 30s/epoch - 309ms/step
49/49 - 3s - 3s/epoch - 51ms/step
97/97 - 28s - loss: 0.8731 - accuracy: 0.6228 - 28s/epoch - 290ms/step
49/49 - 3s - 3s/epoch - 52ms/step
97/97 - 30s - loss: 0.8955 - accuracy: 0.6165 - 30s/epoch - 307ms/step
```

```
49/49 - 2s - 2s/epoch - 51ms/step
97/97 - 29s - loss: 0.8696 - accuracy: 0.6263 - 29s/epoch - 298ms/step
49/49 - 2s - 2s/epoch - 50ms/step
97/97 - 29s - loss: 0.8740 - accuracy: 0.6218 - 29s/epoch - 304ms/step
49/49 - 3s - 3s/epoch - 65ms/step
97/97 - 28s - loss: 0.8783 - accuracy: 0.6241 - 28s/epoch - 289ms/step
49/49 - 3s - 3s/epoch - 67ms/step
Epoch 1/2
97/97 - 29s - loss: 0.8779 - accuracy: 0.6242 - 29s/epoch - 302ms/step
Epoch 2/2
97/97 - 25s - loss: 0.7220 - accuracy: 0.6949 - 25s/epoch - 259ms/step
49/49 - 3s - 3s/epoch - 68ms/step
Epoch 1/2
97/97 - 29s - loss: 0.8862 - accuracy: 0.6176 - 29s/epoch - 303ms/step
Epoch 2/2
97/97 - 25s - loss: 0.7242 - accuracy: 0.6894 - 25s/epoch - 254ms/step
49/49 - 2s - 2s/epoch - 50ms/step
Epoch 1/2
97/97 - 28s - loss: 0.8839 - accuracy: 0.6164 - 28s/epoch - 287ms/step
Epoch 2/2
97/97 - 25s - loss: 0.7149 - accuracy: 0.6877 - 25s/epoch - 255ms/step
49/49 - 3s - 3s/epoch - 52ms/step
Epoch 1/2
97/97 - 30s - loss: 0.8833 - accuracy: 0.6216 - 30s/epoch - 309ms/step
Epoch 2/2
97/97 - 26s - loss: 0.7304 - accuracy: 0.6931 - 26s/epoch - 272ms/step
49/49 - 4s - 4s/epoch - 83ms/step
Epoch 1/2
97/97 - 39s - loss: 0.8786 - accuracy: 0.6179 - 39s/epoch - 398ms/step
Epoch 2/2
97/97 - 27s - loss: 0.7233 - accuracy: 0.6889 - 27s/epoch - 278ms/step
49/49 - 4s - 4s/epoch - 83ms/step
Epoch 1/2
97/97 - 33s - loss: 0.8707 - accuracy: 0.6198 - 33s/epoch - 336ms/step
Epoch 2/2
97/97 - 30s - loss: 0.7207 - accuracy: 0.6833 - 30s/epoch - 308ms/step
49/49 - 3s - 3s/epoch - 52ms/step
Epoch 1/2
291/291 - 49s - loss: 0.8301 - accuracy: 0.6416 - 49s/epoch - 170ms/step
Epoch 2/2
291/291 - 46s - loss: 0.6884 - accuracy: 0.7066 - 46s/epoch - 158ms/step
Best: 0.672548 using {'batch_size': 32, 'epochs': 2, 'optimizer': 'adam'}
```