|  |  |
| --- | --- |
| **EX.NO 7 DATA & TEXT CLASSIFICATION USING NEURAL NETWORKS** | |
| IN [1] | **mport** numpy **as** np *# linear algebra*  **import** pandas **as** pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*  **import** matplotlib.pyplot **as** plt  **from** sklearn.feature\_extraction.text **import** CountVectorizer  **from** sklearn.model\_selection **import** train\_test\_split  **from** sklearn.linear\_model **import** LogisticRegression  **from** sklearn.preprocessing **import** LabelEncoder  **from** sklearn.preprocessing **import** OneHotEncoder  **from** sklearn.model\_selection **import** RandomizedSearchCV  **import** tensorflow **as** tf  **from** tensorflow.keras.models **import** Sequential  **from** tensorflow.keras **import** layers  **from** tensorflow.keras.preprocessing.text **import** Tokenizer  **from** tensorflow.keras.preprocessing.sequence **import** pad\_sequences  **from** tensorflow.keras.wrappers.scikit\_learn **import** KerasClassifier  **import** os  plt**.**style**.**use('ggplot') |
| IN [2] | filepath\_dict **=** {'yelp': '../input/yelp\_labelled.txt',  'amazon': '../input/amazon\_cells\_labelled.txt',  'imdb': '../input/imdb\_labelled.txt'}  df\_list **=** []  **for** source, filepath **in** filepath\_dict**.**items():  df **=** pd**.**read\_csv(filepath, names**=**['sentence', 'label'], sep**=**'\t')  df['source'] **=** source  df\_list**.**append(df) |
| IN [3] | df\_list |
| IN [4] | df **=** pd**.**concat(df\_list)  df**.**iloc[0] |
| IN [5] | df**.**head() |
| IN [6] | df**.**tail() |
| IN [7] | sentences **=** ['Rashmi likes ice cream', 'Rashmi hates chocolate.'] |
| IN [8] | vectorizer **=** CountVectorizer(min\_df**=**0, lowercase**=False**)  vectorizer**.**fit(sentences)  vectorizer**.**vocabulary\_ |
| IN [9] | vectorizer**.**transform(sentences)**.**toarray() |
| IN [10] | df\_yelp **=** df[df['source'] **==** 'yelp']  sentences **=** df\_yelp['sentence']**.**values  y **=** df\_yelp['label']**.**values  sentences\_train, sentences\_test, y\_train, y\_test **=** train\_test\_split(sentences, y, test\_size**=**0.25, random\_state**=**1000) |
| IN [11] | vectorizer **=** CountVectorizer()  vectorizer**.**fit(sentences\_train)  X\_train **=** vectorizer**.**transform(sentences\_train)  X\_test **=** vectorizer**.**transform(sentences\_test) |
| IN [12] | X\_train |
| IN [13] | classifier **=** LogisticRegression()  classifier**.**fit(X\_train, y\_train)  score **=** classifier**.**score(X\_test, y\_test)  print("Accuracy:", score) |
| IN [14] | **for** source **in** df['source']**.**unique():  df\_source **=** df[df['source'] **==** source]  sentences **=** df\_source['sentence']**.**values  y **=** df\_source['label']**.**values  sentences\_train, sentences\_test, y\_train, y\_test **=** train\_test\_split(  sentences, y, test\_size**=**0.25, random\_state**=**1000)  vectorizer **=** CountVectorizer()  vectorizer**.**fit(sentences\_train)  X\_train **=** vectorizer**.**transform(sentences\_train)  X\_test **=** vectorizer**.**transform(sentences\_test)  classifier **=** LogisticRegression()  classifier**.**fit(X\_train, y\_train)  score **=** classifier**.**score(X\_test, y\_test)  print('Accuracy for {} data: {:.4f}'**.**format(source, score)) |
| IN [15] | input\_dim **=** X\_train**.**shape[1]  model **=** Sequential()  model**.**add(layers**.**Dense(10, input\_dim**=**input\_dim, activation**=**'relu'))  model**.**add(layers**.**Dense(1, activation**=**'sigmoid')) |
| IN [16] | model**.**compile(loss**=**'binary\_crossentropy',  optimizer**=**'adam',  metrics**=**['accuracy'])  model**.**summary() |
| IN [17] | history **=** model**.**fit(X\_train, y\_train,  epochs**=**100,  verbose**=True**,  validation\_data**=**(X\_test, y\_test),  batch\_size**=**10) |
| IN [18] | loss, accuracy **=** model**.**evaluate(X\_train, y\_train, verbose**=False**)  print("Training Accuracy: {:.4f}"**.**format(accuracy))  loss, accuracy **=** model**.**evaluate(X\_test, y\_test, verbose**=False**)  print("Testing Accuracy: {:.4f}"**.**format(accuracy)) |
| IN [19] | **def** plot\_history(history):  acc **=** history**.**history['acc']  val\_acc **=** history**.**history['val\_acc']  loss **=** history**.**history['loss']  val\_loss **=** history**.**history['val\_loss']  x **=** range(1, len(acc) **+** 1)  plt**.**figure(figsize**=**(12, 5))  plt**.**subplot(1, 2, 1)  plt**.**plot(x, acc, 'b', label**=**'Training acc')  plt**.**plot(x, val\_acc, 'r', label**=**'Validation acc')  plt**.**title('Training and validation accuracy')  plt**.**legend()  plt**.**subplot(1, 2, 2)  plt**.**plot(x, loss, 'b', label**=**'Training loss')  plt**.**plot(x, val\_loss, 'r', label**=**'Validation loss')  plt**.**title('Training and validation loss')  plt**.**legend() |
| IN [20] | pip install plot\_keras\_history |
| IN [21] | plot\_history(history) |
| IN [22] | cities **=** ['London', 'Berlin', 'Berlin', 'New York', 'London'] |
| IN [23] | encoder **=** LabelEncoder()  city\_labels **=** encoder**.**fit\_transform(cities) |
| IN [24] | encoder **=** OneHotEncoder(sparse**=False**)  city\_labels **=** city\_labels**.**reshape((5, 1))  encoder**.**fit\_transform(city\_labels) |
| IN [25] | tokenizer **=** Tokenizer(num\_words**=**5000)  tokenizer**.**fit\_on\_texts(sentences\_train)  X\_train **=** tokenizer**.**texts\_to\_sequences(sentences\_train)  X\_test **=** tokenizer**.**texts\_to\_sequences(sentences\_test)  vocab\_size **=** len(tokenizer**.**word\_index) **+** 1  print(sentences\_train[2])  print(X\_train[2]) |
| IN [26] | **for** word **in** ['the', 'all','fan']:  print('{}: {}'**.**format(word, tokenizer**.**word\_index[word])) |
| IN [27] | maxlen **=** 100  X\_train **=** pad\_sequences(X\_train, padding**=**'post', maxlen**=**maxlen)  X\_test **=** pad\_sequences(X\_test, padding**=**'post', maxlen**=**maxlen)  print(X\_train[0, :]) |
| IN [28] | embedding\_dim **=** 50  model **=** Sequential()  model**.**add(layers**.**Embedding(input\_dim**=**vocab\_size,  output\_dim**=**embedding\_dim,  input\_length**=**maxlen))  model**.**add(layers**.**Flatten())  model**.**add(layers**.**Dense(10, activation**=**'relu'))  model**.**add(layers**.**Dense(1, activation**=**'sigmoid'))  model**.**compile(optimizer**=**'adam',  loss**=**'binary\_crossentropy',  metrics**=**['accuracy'])  model**.**summary() |
| IN [29] | history **=** model**.**fit(X\_train, y\_train,  epochs**=**20,  verbose**=True**,  validation\_data**=**(X\_test, y\_test),  batch\_size**=**10)  loss, accuracy **=** model**.**evaluate(X\_train, y\_train, verbose**=False**)  print("Training Accuracy: {:.4f}"**.**format(accuracy))  loss, accuracy **=** model**.**evaluate(X\_test, y\_test, verbose**=False**)  print("Testing Accuracy: {:.4f}"**.**format(accuracy)) |
| IN [30] | embedding\_dim **=** 50  model **=** Sequential()  model**.**add(layers**.**Embedding(input\_dim**=**vocab\_size,  output\_dim**=**embedding\_dim,  input\_length**=**maxlen))  model**.**add(layers**.**GlobalMaxPool1D())  model**.**add(layers**.**Dense(10, activation**=**'relu'))  model**.**add(layers**.**Dense(1, activation**=**'sigmoid'))  model**.**compile(optimizer**=**'adam',  loss**=**'binary\_crossentropy',  metrics**=**['accuracy'])  model**.**summary() |
| IN [31] | history **=** model**.**fit(X\_train, y\_train,  epochs**=**50,  verbose**=False**,  validation\_data**=**(X\_test, y\_test),  batch\_size**=**10)  loss, accuracy **=** model**.**evaluate(X\_train, y\_train, verbose**=False**)  print("Training Accuracy: {:.4f}"**.**format(accuracy))  loss, accuracy **=** model**.**evaluate(X\_test, y\_test, verbose**=False**)  print("Testing Accuracy: {:.4f}"**.**format(accuracy)) |
| IN [32] | embedding\_dim **=** 100  model **=** Sequential()  model**.**add(layers**.**Embedding(vocab\_size, embedding\_dim, input\_length**=**maxlen))  model**.**add(layers**.**Conv1D(128, 5, activation**=**'relu'))  model**.**add(layers**.**GlobalMaxPooling1D())  model**.**add(layers**.**Dense(10, activation**=**'relu'))  model**.**add(layers**.**Dense(1, activation**=**'sigmoid'))  model**.**compile(optimizer**=**'adam',  loss**=**'binary\_crossentropy',  metrics**=**['accuracy'])  model**.**summary() |
| IN [33] | embedding\_dim **=** 100  model **=** Sequential()  model**.**add(layers**.**Embedding(vocab\_size, embedding\_dim, input\_length**=**maxlen))  model**.**add(layers**.**Conv1D(128, 5, activation**=**'relu'))  model**.**add(layers**.**GlobalMaxPooling1D())  model**.**add(layers**.**Dense(10, activation**=**'relu'))  model**.**add(layers**.**Dense(1, activation**=**'sigmoid'))  model**.**compile(optimizer**=**'adam',  loss**=**'binary\_crossentropy',  metrics**=**['accuracy'])  model**.**summary() |
| IN [34] | **def** create\_model(num\_filters, kernel\_size, vocab\_size, embedding\_dim, maxlen):  model **=** Sequential()  model**.**add(layers**.**Embedding(vocab\_size, embedding\_dim, input\_length**=**maxlen))  model**.**add(layers**.**Conv1D(num\_filters, kernel\_size, activation**=**'relu'))  model**.**add(layers**.**GlobalMaxPooling1D())  model**.**add(layers**.**Dense(10, activation**=**'relu'))  model**.**add(layers**.**Dense(1, activation**=**'sigmoid'))  model**.**compile(optimizer**=**'adam',  loss**=**'binary\_crossentropy',  metrics**=**['accuracy'])  **return** model |
| IN [35] | param\_grid **=** dict(num\_filters**=**[32, 64, 128],  kernel\_size**=**[3, 5, 7],  vocab\_size**=**[5000],  embedding\_dim**=**[50],  maxlen**=**[100]) |
| IN [36] | epochs **=** 20  embedding\_dim **=** 50  maxlen **=** 100  output\_file **=** 'output.txt'  **for** source, frame **in** df**.**groupby('source'):  print('Running grid search for data set :', source)  sentences **=** df['sentence']**.**values  y **=** df['label']**.**values  sentences\_train, sentences\_test, y\_train, y\_test **=** train\_test\_split(  sentences, y, test\_size**=**0.25, random\_state**=**1000)  tokenizer **=** Tokenizer(num\_words**=**5000)  tokenizer**.**fit\_on\_texts(sentences\_train)  X\_train **=** tokenizer**.**texts\_to\_sequences(sentences\_train)  X\_test **=** tokenizer**.**texts\_to\_sequences(sentences\_test)  vocab\_size **=** len(tokenizer**.**word\_index) **+** 1  X\_train **=** pad\_sequences(X\_train, padding**=**'post', maxlen**=**maxlen)  X\_test **=** pad\_sequences(X\_test, padding**=**'post', maxlen**=**maxlen)  param\_grid **=** dict(num\_filters**=**[32, 64, 128],  kernel\_size**=**[3, 5, 7],  vocab\_size**=**[vocab\_size],  embedding\_dim**=**[embedding\_dim],  maxlen**=**[maxlen])  model **=** KerasClassifier(build\_fn**=**create\_model,  epochs**=**epochs, batch\_size**=**10,  verbose**=False**)  grid **=** RandomizedSearchCV(estimator**=**model, param\_distributions**=**param\_grid,  cv**=**4, verbose**=**1, n\_iter**=**5)  grid\_result **=** grid**.**fit(X\_train, y\_train)    test\_accuracy **=** grid**.**score(X\_test, y\_test)  s **=** ('Running {} data set\nBest Accuracy : '  '{:.4f}\n{}\nTest Accuracy : {:.4f}\n\n')  output\_string **=** s**.**format(  source,  grid\_result**.**best\_score\_,  grid\_result**.**best\_params\_,  test\_accuracy)  print(output\_string) |