cnn-for-text-classification

March 22, 2024

0.1 Import the necessary libraries and read the dataset

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer,␣
 ↪TfidfVectorizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Embedding,GlobalMaxPooling1D, Conv1D
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Embedding
```

```
2024-03-22 10:46:24.334532: I tensorflow/core/util/port.cc:113] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-03-22 10:46:24.335251: I external/local_tsl/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2024-03-22 10:46:24.341689: I external/local_tsl/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2024-03-22 10:46:24.429780: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
2024-03-22 10:46:25.887709: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

[2]:
```
# nltk.download('stopwords')
```

[3]:
```
# nltk.download('wordnet')
```

[4]:
```
data = pd.read_csv(r'spam.csv')
```

[5]:
```
data.head()
```

[5]:
```
     v1                                                 v2 Unnamed: 2  \
0   ham  Go until jurong point, crazy.. Available only …         NaN
1   ham                      Ok lar… Joking wif u oni…         NaN
2  spam  Free entry in 2 a wkly comp to win FA Cup fina…         NaN
3   ham  U dun say so early hor… U c already then say…         NaN
4   ham  Nah I don't think he goes to usf, he lives aro…         NaN

  Unnamed: 3 Unnamed: 4
0        NaN        NaN
1        NaN        NaN
2        NaN        NaN
3        NaN        NaN
4        NaN        NaN
```

0.2 Text Preprocessing

[6]:
```
data.columns
```

[6]:
```
Index(['v1', 'v2', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], dtype='object')
```

[7]:
```
data.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], inplace=True)
```

[8]:
```
data.columns = ['label','data']
```

[9]:
```
data.nunique()
```

[9]:
```
label       2
data     5169
dtype: int64
```

0.2.1 Encode the label column

[10]:
```
label_encoder = LabelEncoder()
data['label']= label_encoder.fit_transform(data['label'])
```

[11]:
```
data.head()
```

[11]:
```
   label                                               data
0      0  Go until jurong point, crazy.. Available only …
1      0                      Ok lar… Joking wif u oni…
```

```
2      1  Free entry in 2 a wkly comp to win FA Cup fina…
3      0  U dun say so early hor… U c already then say…
4      0  Nah I don't think he goes to usf, he lives aro…
```

0.2.2 Remove unwanted characters

[12]:
```
data['data'][4]
```

[12]:
```
"Nah I don't think he goes to usf, he lives around here though"
```

[13]:
```
def clean_data(x):
    review = re.sub('[^a-zA-Z]', ' ',x)
    review = x.lower()
    return x

data['data'] = data['data'].apply(clean_data)
```

[14]:
```
data['data'][4]
```

[14]:
```
"Nah I don't think he goes to usf, he lives around here though"
```

0.2.3 Remove Stop Words

[15]:
```
def remove_stop_words(x):
    l = []
    stop_words = stopwords.words('english')
    x = x.split()
    x = [l.append(word) for word in x if word not in stop_words]
    x = ' '.join(l)
    return x
```

[16]:
```
data['data'] = data['data'].apply(remove_stop_words)
```

[17]:
```
data['data'][4]
```

[17]:
```
'Nah I think goes usf, lives around though'
```

0.2.4 Lemmatization

[18]:
```
lemmatizer = WordNetLemmatizer()
def lematize(x):
    x = x.split()
    x = [lemmatizer.lemmatize(w) for w in x]
    x = ' '.join(x)
    return x
```

[19]:
```
data['data'] = data['data'].apply(lematize)
```

[20]:
```
data['data'][4]
```

[20]:
```
'Nah I think go usf, life around though'
```

0.2.5 Vectorization

[21]:
```
corpus = list(data['data'])
```

[22]:
```
cv = CountVectorizer(max_features = 1000)
X = cv.fit_transform(corpus).toarray()
y = data['label'].values
```

[23]:
```
tf_transformer = TfidfTransformer()
X = tf_transformer.fit_transform(X).toarray()
```

[24]:
```
tfidfVectorizer = TfidfVectorizer(max_features =1000)
X = tfidfVectorizer.fit_transform(corpus).toarray()
```

0.3 Train Test Split

[25]:
```
X_train, X_test , y_train, y_test = train_test_split(X, y , test_size = 0.20,␣
↪random_state=101)
```

[26]:
```
print(f"Training data shape: {X_train.shape}, {y_train.shape}")
print(f"Testing data shape: {X_test.shape}, {y_test.shape}")
```

```
Training data shape: (4457, 1000), (4457,)
Testing data shape: (1115, 1000), (1115,)
```

[27]:
```
X_train = pad_sequences(X_train)
X_test = pad_sequences(X_test)
```

[28]:
```
vocab_size = 5000
model = Sequential([
    Embedding(vocab_size, 8),
    Conv1D(128, 5, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(10, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

[29]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

[31]:
```
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test,␣
↪y_test))
```

```
Epoch 1/10
140/140             4s 26ms/step -
acc: 0.8674 - loss: 0.3920 - val_acc: 0.8762 - val_loss: 0.3730
Epoch 2/10
140/140             4s 26ms/step -
acc: 0.8735 - loss: 0.3824 - val_acc: 0.8762 - val_loss: 0.3767
Epoch 3/10
140/140             3s 24ms/step -
acc: 0.8625 - loss: 0.3998 - val_acc: 0.8762 - val_loss: 0.3711
Epoch 4/10
140/140             4s 26ms/step -
acc: 0.8520 - loss: 0.4174 - val_acc: 0.8762 - val_loss: 0.3698
Epoch 5/10
140/140             3s 22ms/step -
acc: 0.8620 - loss: 0.3971 - val_acc: 0.8762 - val_loss: 0.3695
Epoch 6/10
140/140             3s 23ms/step -
acc: 0.8697 - loss: 0.3847 - val_acc: 0.8762 - val_loss: 0.3729
Epoch 7/10
140/140             3s 22ms/step -
acc: 0.8541 - loss: 0.4155 - val_acc: 0.8762 - val_loss: 0.3701
Epoch 8/10
140/140             3s 22ms/step -
acc: 0.8634 - loss: 0.3942 - val_acc: 0.8762 - val_loss: 0.3740
Epoch 9/10
140/140             3s 23ms/step -
acc: 0.8633 - loss: 0.3939 - val_acc: 0.8762 - val_loss: 0.3787
Epoch 10/10
140/140             3s 23ms/step -
acc: 0.8616 - loss: 0.4024 - val_acc: 0.8762 - val_loss: 0.3739
```

[32]:
```
loss, accuracy = model.evaluate(X_test,y_test)
print('Testing Accuracy is {} '.format(accuracy*100))
```

```
35/35            0s 5ms/step - acc:
0.8807 - loss: 0.3650
Testing Accuracy is 87.6233160495758
```

0.4 Inference

[33]:
```
text = "You are invited for the grand launch of XYZ. You get a chance to win␣
↪cash in millions."
```

[34]:
```
text = clean_data(text)
text = remove_stop_words(text)
text = lematize(text)
```

[35]:
```
print(text)
```

```
You invited grand launch XYZ. You get chance win cash million
```

[36]:
```
inf_X = cv.fit_transform([text]).toarray()
inf_X_tf = tf_transformer.fit_transform(inf_X).toarray()
tfidf = tfidfVectorizer.fit_transform([text]).toarray()
```

[37]:
```
final_text = pad_sequences(tfidf)
```

[38]:
```
prediction = model.predict(final_text)
prediction
```

```
1/1             0s 107ms/step
```

[38]:
```
array([[0.16587973]], dtype=float32)
```

[39]:
```
print("Predicted Value: ", "Not Spam" if prediction > 0.5 else "Spam")
```

```
Predicted Value:  Spam
```

Reference - https://cnvrg.io/cnn-sentence-classification/