lstm

June 23, 2024

[1]:
```
import pandas as pd
```

[2]:
```
df_ = pd.read_json('news.json')
```

[3]:
```
df = df_[['title','related']]
```

[4]:
```
df.head()
```

[4]:

	title	related
0	Corinthians se atrapalha em diagnóstico e tent…	no
1	Cidades da região recebem doses que serão usad…	yes
2	Santa Lúcia confirma caso de raiva bovina	yes
3	Vacina antirrábica é aplicada com agendamento …	yes
4	Divertida Mente 2: Vá da Raiva à Alegria com o…	no

[5]:
```
df.tail()
```

[5]:

	title	related
95	VACINAÇÃO CONTRA RAIVA É REALIZADA NO SÁBADO N…	yes
96	Morcegos infectados com raiva são encontrados …	yes
97	Prefeitura de Montes Claros faz bloqueio contr…	yes
98	Nissan Sentra 2025 foi renovado e mata de raiv…	no
99	Voluntários de resgate ou de abrigos de animai…	yes

[6]:
```
df['out'] = df['related'].map({'yes':1,'no':0})
```

```
/tmp/ipykernel_870141/2043172176.py: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
  df['out'] = df['related'].map({'yes':1,'no':0})
```

[7]:
```
df
```

[7]:

	title	related	out
0	Corinthians se atrapalha em diagnóstico e tent…	no	0
1	Cidades da região recebem doses que serão usad…	yes	1

```
2          Santa Lúcia confirma caso de raiva bovina     yes    1
3   Vacina antirrábica é aplicada com agendamento …     yes    1
4   Divertida Mente 2: Vá da Raiva à Alegria com o…      no    0
..                                                  …     …  …
95  VACINAÇÃO CONTRA RAIVA É REALIZADA NO SÁBADO N…     yes    1
96  Morcegos infectados com raiva são encontrados …     yes    1
97  Prefeitura de Montes Claros faz bloqueio contr…     yes    1
98  Nissan Sentra 2025 foi renovado e mata de raiv…      no    0
99  Voluntários de resgate ou de abrigos de animai…     yes    1

[100 rows x 3 columns]
```

[8]:
```
from sklearn.preprocessing import LabelEncoder
from keras import Sequential
from keras.layers import Embedding, Dense, LSTM
from keras.utils import pad_sequences
import nltk
from nltk.stem.snowball import SnowballStemmer
import regex as re
from nltk.tokenize import sent_tokenize
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
```

```
2024-06-23 09:16:02.871825: I external/local_tsl/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.
2024-06-23 09:16:02.908050: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-06-23 09:16:02.908074: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-06-23 09:16:02.909008: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2024-06-23 09:16:02.914411: I external/local_tsl/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.
2024-06-23 09:16:02.915054: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2024-06-23 09:16:03.656398: W
```

```
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

[9]:
```
from keras.preprocessing.text import one_hot
```

[10]:
```
text_cleaning = "\b0\S*|\b[^A-Za-z0-9]+"
def preprocess_filter(text, stem=False):
    text = re.sub(text_cleaning, " ", str(text.lower()).strip())
    tokens = []
    for token in text.split():
        if token not in stop_words:
            if stem:
                stemmer = SnowballStemmer(language='portuguese')
                token = stemmer.stem(token)
            tokens.append(token)
    return " ".join(tokens)
```

[11]:
```
df['title'].size
```

[11]:
```
100
```

[13]:
```
# download some packages
from nltk.corpus import stopwords

nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

stop_words = stopwords.words('portuguese')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/jpantonow/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /home/jpantonow/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /home/jpantonow/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[17]:
```
# longest sentence length
def longest_sentence_length(text):
    return len(text.split())
df['maximum_length'] = df['title'].apply(lambda x: longest_sentence_length(x))
print('longest sentence having length -')
max_length = max(df['maximum_length'].values)
print(max_length)
```

```
longest sentence having length -
21
```

[18]:
```
size_vocab = df['maximum_length'].sum()
```

[19]:
```
size_vocab
```

[19]: 1309

[20]:
```
# Word embedding with pre padding
def one_hot_encoded(text, vocab_size=size_vocab, max_length=21):
    hot_encoded = one_hot(text, vocab_size)
    return hot_encoded
```

[ ]:
```
# # Word embedding with pre padding
# def one_hot_encoded(text, vocab_size=5000, max_length=40):
#     hot_encoded = to_categorical(text, vocab_size)
#     return hot_encoded
```

[21]:
```
# word embedding pipeline
def word_embedding(text):
    preprocessed_text = preprocess_filter(text)
    return one_hot_encoded(preprocessed_text)
```

[22]:
```
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense

embedded_features = 21
max_length = 21

model = Sequential()
model.add(Embedding(size_vocab, embedded_features, input_length=max_length))
model.add(LSTM(10))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 21, 21)            27489

 lstm (LSTM)                 (None, 10)                1280

 dense (Dense)               (None, 1)                 11

=================================================================
Total params: 28780 (112.42 KB)
Trainable params: 28780 (112.42 KB)
```

```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

[23]:
```
from keras.preprocessing.text import one_hot
```

[24]:
```
one_hot_encoded_title = df['title'].apply(lambda x: word_embedding(x)).values
```

[25]:
```
one_hot_encoded_title
```

[25]:
```
array([list([60, 1187, 707, 1068, 252, 30, 89]),
       list([379, 799, 571, 599, 589, 906, 6, 493, 30, 606, 1276, 1297, 139,
143]),
       list([184, 695, 310, 442, 30, 473]),
       list([27, 472, 360, 469, 140]),
       list([727, 833, 850, 1294, 30, 784, 536, 1183, 173]),
       list([271, 425, 239, 974, 30]),
       list([27, 472, 232, 482, 982, 254]),
       list([822, 621, 758, 621, 199, 1238, 603, 815, 820, 30, 1150, 365, 220]),
       list([113, 1118, 136, 38, 460, 30, 1297, 109]),
       list([819, 131, 669, 27, 1297, 493, 30, 1153, 1164, 978, 388]),
       list([728, 966, 234, 941, 451, 493, 30, 922]),
       list([6, 493, 30, 1017, 1143, 127, 334, 113]),
       list([770, 30, 1203, 1296, 295, 1302]),
       list([154, 38, 807, 30, 67, 1190, 420, 484, 184, 140]),
       list([6, 493, 30, 267, 941, 451]),
       list([1153, 1164, 148, 6, 472, 629, 735]),
       list([30, 743, 1119, 998, 183, 162]),
       list([727, 833, 850, 1294, 30, 784, 536, 1183, 173]),
       list([1208, 763, 96, 1286, 819, 131, 1282, 10, 460, 30, 221, 819, 131]),
       list([420, 863, 1297, 1259, 234, 493, 30, 804, 270]),
       list([65, 384, 1236, 558, 329, 30, 67, 915]),
       list([560, 756, 121, 30, 75, 978, 711, 415]),
       list([338, 1254, 310, 442, 30, 560, 99, 615, 978, 711, 922]),
       list([222, 336, 1027, 455, 310, 753, 1102, 30, 135]),
       list([1027, 456, 384, 1155, 329, 1302, 123, 30, 107]),
       list([258, 30, 1259, 1082, 640, 4, 1210, 781, 1307]),
       list([819, 374, 977, 38, 807, 30, 67, 799, 1169, 136, 65]),
       list([1194, 125, 1277, 102, 406, 27, 695, 493, 30]),
       list([30, 140, 1058, 829, 570, 793, 970, 232]),
       list([140, 965, 560, 396, 30, 1058, 280, 793, 365, 220]),
       list([318, 30, 1167, 709, 1227, 1082, 849, 44, 1117]),
       list([137, 714, 6, 493, 30, 267, 746, 1153, 1164, 999]),
       list([131, 669, 156, 1161, 1115, 1297, 493, 30, 927, 1164]),
       list([442, 30, 267, 435, 365, 107, 35, 275, 790]),
       list([30, 1000, 113, 965, 560, 575, 864, 136, 922]),
       list([1024, 1004, 664, 440, 1036, 401, 30, 221]),
       list([131, 669, 156, 1161, 709, 1297, 493, 30, 927, 1164]),
```

```
       list([113, 156, 1154, 27, 493, 30, 702, 78, 497, 237, 35, 113, 636,
177]),
       list([395, 177, 576, 1297, 223, 139, 974, 493, 30]),
       list([819, 131, 27, 941, 451, 493, 30, 1153, 1164, 669]),
       list([518, 1257, 21, 829, 6, 493, 30, 221, 420, 131, 267, 364]),
       list([714, 956, 30, 728, 974, 232]),
       list([113, 965, 1192, 442, 30, 560, 232]),
       list([30, 728, 239, 974, 377, 480, 784]),
       list([442, 348, 821, 575, 30, 529, 38, 10, 186, 121, 1076, 554]),
       list([1302, 881, 30, 175, 63, 964]),
       list([113, 669, 1183, 1043, 792, 6, 493, 30, 267]),
       list([922, 310, 557, 442, 560, 396, 30]),
       list([714, 874, 746, 1153, 1164, 6, 493, 30, 941, 451]),
       list([560, 756, 30, 75, 338, 1254]),
       list([406, 310, 442, 30, 473]),
       list([30, 1199, 1296, 71, 594, 273, 232]),
       list([560, 30, 75, 1000, 482, 232, 1308, 993]),
       list([6, 493, 30, 1238, 605, 864, 846, 150, 664, 699]),
       list([30, 221, 435, 815, 974]),
       list([1208, 30, 955, 1280, 11, 311, 798, 857, 499, 154, 1307]),
       list([829, 155, 687, 1066, 637, 30]),
       list([1307, 455, 1141, 348, 442, 821, 575, 30, 437, 1000]),
       list([961, 204, 35, 131, 961, 619, 209, 35, 714, 956, 493, 30, 340, 727,
131, 528]),
       list([27, 493, 30, 742, 516, 1082, 158]),
       list([1072, 1004, 1258, 234, 941, 451, 493, 30, 1153, 1164, 209, 113,
127, 626, 845]),
       list([281, 384, 1155, 329, 30, 1302, 304, 318, 819, 131]),
       list([1188, 39, 96, 1277, 1276, 659, 351, 414, 527, 326, 787, 960]),
       list([867, 379, 1307, 1275, 499, 279, 834, 704, 1084, 30, 728]),
       list([193, 1106, 759, 1304, 678, 1064, 493, 30, 221, 113, 127, 626,
845]),
       list([560, 30, 535, 113, 1000, 391, 864, 136, 906, 38, 671, 799]),
       list([1198, 673, 1268, 6, 941, 451, 493, 30, 1139, 530]),
       list([434, 845, 1183, 376, 47, 10, 155, 460, 30, 221]),
       list([155, 1141, 759, 388, 1302, 881, 30, 338]),
       list([38, 10, 406, 442, 350, 30, 699, 285]),
       list([820, 30, 467, 1096, 1229, 144, 464, 991]),
       list([714, 956, 517, 30, 410, 624, 439, 460, 570, 493, 232]),
       list([1079, 490, 6, 493, 30, 1156, 1157, 195]),
       list([560, 30, 535, 1000, 482, 1296, 131]),
       list([281, 384, 491, 442, 30, 560, 304, 318, 562]),
       list([669, 1303, 1163, 144, 609, 30]),
       list([158, 88, 1076, 1149, 1284, 249, 589, 254, 10, 232, 30]),
       list([6, 493, 30, 941, 451, 265, 1164, 714, 1188, 327, 562]),
       list([579, 470, 941, 451, 191, 493, 30, 683, 1159, 113, 127, 626, 845]),
       list([707, 30, 890, 97, 1027, 456]), list([30, 59, 162, 131, 207]),
```

```
        list([819, 131, 21, 25, 10, 200, 460, 30, 221]),
        list([263, 558, 1302, 881, 30, 1000]),
        list([140, 965, 1302, 123, 30, 1058, 280, 365, 291]),
        list([141, 30, 183, 289, 1280, 1235, 1270]),
        list([1307, 1275, 499, 811, 1030, 1057, 30]),
        list([391, 1282, 125, 1277, 1102, 695, 493, 30, 109]),
        list([560, 30, 535, 1000, 482, 1296, 131]),
        list([1051, 848, 457, 335, 1187, 616, 155, 30]),
        list([819, 374, 38, 10, 295, 460, 493, 30, 1297, 109, 35, 113]),
        list([560, 1071, 165, 664, 30, 20, 38, 267, 615, 574, 232, 482, 71]),
        list([853, 829, 6, 493, 30, 267, 1253, 669, 746, 714, 1188, 107]),
        list([374, 977, 38, 807, 30, 67, 799, 1169, 136, 65]),
        list([819, 131, 310, 280, 329, 30, 1302]),
        list([442, 30, 560, 435, 928, 339]),
        list([6, 493, 30, 54, 1164, 799, 324, 412]),
        list([1302, 881, 30, 175, 964, 242]),
        list([113, 29, 988, 462, 1119, 493, 30, 603, 442, 396, 560]),
        list([533, 203, 34, 834, 783, 30, 281, 50]),
        list([395, 177, 576, 1297, 139, 974, 493, 30])], dtype=object)
```

[26]:
```
# padding to make the size equal of the sequences
padded_encoded_title = pad_sequences(
    one_hot_encoded_title, maxlen=max_length, padding='pre')
```

[27]:
```
import numpy as np
```

[28]:
```
# Splitting
X = padded_encoded_title
y = df['out'].values
y = np.array(y)

# shapes
print(X.shape)
print(y.shape)
```

```
(100, 21)
(100,)
```

[ ]:
```
X
```

[ ]:
```
y
```

[29]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=5)

# Shape and size of train and test dataset
print('X train shape {}'.format(X_train.shape))
print('X test shape {}'.format(X_test.shape))
print('y train shape {}'.format(y_train.shape))
```

```
print('y test shape {}'.format(y_test.shape))
```

```
X train shape (75, 21)
X test shape (25, 21)
y train shape (75,)
y test shape (25,)
```

[30]:
```
history= model.fit(X_train, y_train, validation_data=(
    X_test, y_test), epochs=10, batch_size=4)
```

```
Epoch 1/10
19/19 [==============================] - 2s 26ms/step - loss: 0.6770 - accuracy:
0.8533 - val_loss: 0.6476 - val_accuracy: 0.9200
Epoch 2/10
19/19 [==============================] - 0s 6ms/step - loss: 0.6260 - accuracy:
0.8267 - val_loss: 0.5628 - val_accuracy: 0.9200
Epoch 3/10
19/19 [==============================] - 0s 5ms/step - loss: 0.5298 - accuracy:
0.8267 - val_loss: 0.3988 - val_accuracy: 0.9200
Epoch 4/10
19/19 [==============================] - 0s 6ms/step - loss: 0.4339 - accuracy:
0.8267 - val_loss: 0.2975 - val_accuracy: 0.9200
Epoch 5/10
19/19 [==============================] - 0s 6ms/step - loss: 0.3890 - accuracy:
0.8267 - val_loss: 0.2788 - val_accuracy: 0.9200
Epoch 6/10
19/19 [==============================] - 0s 6ms/step - loss: 0.3373 - accuracy:
0.8267 - val_loss: 0.2526 - val_accuracy: 0.9200
Epoch 7/10
19/19 [==============================] - 0s 6ms/step - loss: 0.2769 - accuracy:
0.8267 - val_loss: 0.2054 - val_accuracy: 0.9200
Epoch 8/10
19/19 [==============================] - 0s 6ms/step - loss: 0.2170 - accuracy:
0.8667 - val_loss: 0.1601 - val_accuracy: 0.9200
Epoch 9/10
19/19 [==============================] - 0s 8ms/step - loss: 0.1723 - accuracy:
0.9333 - val_loss: 0.1341 - val_accuracy: 0.9600
Epoch 10/10
19/19 [==============================] - 0s 6ms/step - loss: 0.1437 - accuracy:
0.9733 - val_loss: 0.1136 - val_accuracy: 0.9600
```

[72]:
```
def predict(text):
    encoded = word_embedding(text)
    padded_encoded_title = pad_sequences(
        [encoded], maxlen=max_length, padding='pre')
    output = model.predict(padded_encoded_title)
    print(output)
    output = np.where(output >= 0.85, 1, 0)
```

```
    if output[0][0] == 1:
        return 'Relacionado ao tema'
    return 'Não relacionado ao tema'
```

[73]:
```
predict('oi')
```

```
1/1 [==============================] - 0s 16ms/step
[[0.6916496]]
```

[73]: 'Não relacionado ao tema'

[74]:
```
predict('raiva corinthians')
```

```
1/1 [==============================] - 0s 16ms/step
[[0.6922663]]
```

[74]: 'Não relacionado ao tema'

[75]:
```
predict('raiva do caralho')
```

```
1/1 [==============================] - 0s 16ms/step
[[0.77842915]]
```

[75]: 'Não relacionado ao tema'

[76]:
```
predict('Vacina antirrábica é aplicada ')
```

```
1/1 [==============================] - 0s 19ms/step
[[0.9097004]]
```

[76]: 'Relacionado ao tema'

[77]:
```
predict('policial tem surto de raiva')
```

```
1/1 [==============================] - 0s 17ms/step
[[0.8266267]]
```

[77]: 'Não relacionado ao tema'

[78]:
```
predict('Vacina antirrábica é aplicada com agendamento na VISAM')
```

```
1/1 [==============================] - 0s 16ms/step
[[0.9483973]]
```

[78]: 'Relacionado ao tema'

[79]:
```
predict('morcego com raiva é detectado')
```

```
1/1 [==============================] - 0s 17ms/step
[[0.85924757]]
```

[79]:
```
'Relacionado ao tema'
```

[80]:
```
predict('Secretaria de Saúde de Indaiatuba vacina animais contra raiva neste␣
↪sábado no bairro Tombadouro')
```

```
1/1 [==============================] - 0s 16ms/step
[[0.9704852]]
```

[80]:
```
'Relacionado ao tema'
```

[85]:
```
predict('Jovem mata mãe em surto de raiva')
```

```
1/1 [==============================] - 0s 16ms/step
[[0.6890552]]
```

[85]:
```
'Não relacionado ao tema'
```