amazon-reviews-short

February 24, 2021

1 Sentiment Analysis with Deep Learning through Keras

Georg Zhelev

1.1 Table of Contents

[ ]:

```python
#load data
import os # accessing directory structure
import bz2 #unzip and load

#view/work with data
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np

#split
from sklearn.model_selection import train_test_split

#pre-process
from tensorflow.keras.preprocessing.text import Tokenizer, text_to_word_sequence
from tensorflow.keras.preprocessing.sequence import pad_sequences

#fit DL
import tensorflow
from tensorflow.python.keras import models, layers, optimizers
from keras.models import Sequential
from keras import layers

#fit ML
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

#from sklearn.model_selection import cross_val_score
```

```
#from sklearn.model_selection import StratifiedKFold
#from sklearn.model_selection import GridSearchCV

#evaluate
from sklearn.metrics import f1_score, roc_auc_score, accuracy_score

#to reset the trained model
#from keras.backend import clear_session
```

1.1.1 Show file names in directory

[ ]:
```
print(os.listdir('../input'))
```

1.1.2 1. Data Description

- Amazon customer reviews (input) and star ratings (output)
- industrial level dataset (3,6 mil. train)

The data format is the following: - label, Text (all in one line) - label 1 corresponds to 1- and 2-star reviews - label 2 corresponds to 4- and 5-star reviews - Most of the reviews are in English

1.1.3 1.1 Load and decompress Files

Decompress files and return a list containing each line as a list item.

[55]:
```
train_file = bz2.BZ2File('../input/train.ft.txt.bz2')
test_file = bz2.BZ2File('../input/test.ft.txt.bz2')

train_file_lines = train_file.readlines(int(448.374641923*50000))
test_file_lines = test_file.readlines(int(452.488687783*10000))
```

[56]:
```
print(int(448.374641923*500000))  #bytes to read
print(int(452.488687783*100000)) #bytes to read
```

```
224187320
45248868
```

[57]:
```
type(train_file_lines)
train_file_lines[0]
```

[57]:
```
b'__label__2 Stuning even for the non-gamer: This sound track was beautiful! It
paints the senery in your mind so well I would recomend it even to people who
hate vid. game music! I have played the game Chrono Cross but out of all of the
games I have ever played it has the best music! It backs away from crude
keyboarding and takes a fresher step with grate guitars and soulful orchestras.
It would impress anyone who cares to listen! ^_^\n'
```

1.1.4 1.2 Decode from raw binary strings to strings that can be parsed. Extract labels and extract texts.

[90]:
```
dec=train_file_lines[0].decode("utf-8")

print(dec[0:10])
print(dec[10:])
```

```
__label__2
 Stuning even for the non-gamer: This sound track was beautiful! It paints the
senery in your mind so well I would recomend it even to people who hate vid.
game music! I have played the game Chrono Cross but out of all of the games I
have ever played it has the best music! It backs away from crude keyboarding and
takes a fresher step with grate guitars and soulful orchestras. It would impress
anyone who cares to listen! ^_^
```

[60]:
```
def get_labels_and_texts(file):
    labels = []
    texts = []
    for line in file:
        x = line.decode("utf-8")
        labels.append(int(x[9]) - 1)
        texts.append(x[10:].strip())
    return np.array(labels), texts
```

strip() remove spaces at the beginning and at the end of the string: 1 is subtracted from labeles 1 and 2 to result in labels 0 and 1

[61]:
```
train_labels_l, train_texts_l = get_labels_and_texts(train_file_lines)
test_labels, test_texts_l = get_labels_and_texts(test_file_lines)

del train_file, test_file
```

[62]:
```
print(test_labels)
train_texts_l[0]
```

```
[1 1 0 … 1 0 1]
```

[62]:
```
'Stuning even for the non-gamer: This sound track was beautiful! It paints the
senery in your mind so well I would recomend it even to people who hate vid.
game music! I have played the game Chrono Cross but out of all of the games I
have ever played it has the best music! It backs away from crude keyboarding and
takes a fresher step with grate guitars and soulful orchestras. It would impress
anyone who cares to listen! ^_^'
```

1.1.5 2. View Prepared Data

Text is saved into a list, while labels saved into an array.

[11]:
```python
#print(train_labels_l.shape)
#print(type(train_labels_l))
#print(type(train_texts_l))

print("Length train:", len(train_texts_l))
print("Length test:", len(test_texts_l))
```

```
Length train: 49473
Length test: 10007
```

1.1.6 2.2 Show examples of the Data

Positive and negative reviews (y-variable)

[66]:
```python
train_labels_l[:10]
```

[66]:
```
array([1, 1, 1, 1, 1, 1, 0, 1, 1, 1])
```

First text (x-variable)

1.1.7 3. Pre-Process

- bild a small and efficient vocabulary
- Stopwords only blow up the vocabulary
- non-numerical values
- Using the regular expressions module
- Match characters and subsititute them with spaces

1. Lowercase text
2. Remove non-word characters:
   - numbers and punctuation
3. Removes non-english language characters

[68]:
```python
##### %%time

import re
NON_ALPHANUM = re.compile(r'[\W]')
NON_ASCII = re.compile(r'[^a-z0-1\s]')

def normalize_texts(texts):
    normalized_texts = []
    for text in texts:
        lower = text.lower()
        no_punctuation = NON_ALPHANUM.sub(r' ', lower)
        no_non_ascii = NON_ASCII.sub(r'', no_punctuation)
        normalized_texts.append(no_non_ascii)
    return normalized_texts

train_texts_p = normalize_texts(train_texts_l)
```

```
test_texts_p = normalize_texts(test_texts_l)
```

- .compile = Compile a regular expression pattern into a regular expression object, which can be used for matching
- .sub = Returns the string obtained by replacing the non-overlapping occurrences of pattern in string by the replacement r' '
- r' ' is an empty space to be replaced with

Stopwords and a Lematizer are not applied because Word Embedings will be used, which makes this step redundant.

[69]:
```
print(len(train_texts_p))
print(len(test_texts_p))
```

```
49473
10007
```

[70]:
```
print(train_texts_l[0:1])
print("\n")
print(train_texts_p[0:1])
```

```
['Stuning even for the non-gamer: This sound track was beautiful! It paints the
senery in your mind so well I would recomend it even to people who hate vid.
game music! I have played the game Chrono Cross but out of all of the games I
have ever played it has the best music! It backs away from crude keyboarding and
takes a fresher step with grate guitars and soulful orchestras. It would impress
anyone who cares to listen! ^_^']


['stuning even for the non gamer  this sound track was beautiful  it paints the
senery in your mind so well i would recomend it even to people who hate vid
game music  i have played the game chrono cross but out of all of the games i
have ever played it has the best music  it backs away from crude keyboarding and
takes a fresher step with grate guitars and soulful orchestras  it would impress
anyone who cares to listen    ']
```

1.1.8 3.1 Describe Data

[65]:
```
print(pd.value_counts(train_labels_l))
```

```
1    25217
0    24256
dtype: int64
```

About equal distribution of classes. (1 is positive, while 0 a negative review).

1.1.9 4. Split Data

[19]:
```
train_texts_s, val_texts_s, train_labels, val_labels =␣
↪train_test_split(train_texts_p, train_labels_l, random_state=57643892,␣
↪test_size=0.2)

print("Length train texts:", len(train_texts_s))
#print(len(train_labels))

print("Length validation texts:", len(val_texts_s))
#print(len(val_labels))

print("Length text texts", len(test_texts_p))
#print(len(test_labels))
```

```
Length train texts: 39578
Length validation texts: 9895
Length text texts 10007
```

1.2 5 .Tokenize Text

- split texts into lists of tokens.
- assign max features (1200 most common words)
- creates the vocabulary based on train data
- resulting vectors equal the length of each text

[72]:
```
%%time

MAX_FEATURES = 12000
tokenizer = Tokenizer(num_words=MAX_FEATURES)
tokenizer.fit_on_texts(train_texts_s)
```

```
CPU times: user 3.25 s, sys: 11.4 ms, total: 3.26 s
Wall time: 3.26 s
```

1.2.1 5.1 Encode training data sentences into sequences

- Transforms each text into a sequence of integers.
- Assigns an integer to each word
- Can access the word index (a dictionary) to verify assigned integer to the word

[73]:
```
%%time

train_texts_b = tokenizer.texts_to_sequences(train_texts_s)
val_texts_b = tokenizer.texts_to_sequences(val_texts_s)
test_texts_b = tokenizer.texts_to_sequences(test_texts_p)
```

```
CPU times: user 4.36 s, sys: 11.1 ms, total: 4.37 s
Wall time: 4.37 s
```

5.2 Show an encoded Sequence

[74]:
```
data-visibility="hidden"
print("First review:", train_texts_s[0], end=" ")
print("\n")
print("First encoded review:", train_texts_b[0], end=" ")
print("\n")

print("Lenth before encoding", len(train_texts_s[0]))
print("Lenth before encoding", len(train_texts_b[0]))

print("wonderful", tokenizer.word_index["wonderful"])
print("inspiring", tokenizer.word_index["inspiring"])
```

```
First review: wonderful  inspiring music   so many artists struggle to put 10
songs on an album  of which maybe half could be considered decent  joseph arthur
manages to create 1 for this album and there s not a loser in the bunch his
songs are pure poetry surrounded by swirling layers of gorgeous music
sometimes simplistic folk  other times upbeat rock  but his lyrics carry each
one with often times devastating results  in a good way   tales of love lost and
struggles to love are the most common  but they never get tiring due to the
diversity of the tracks for those who do love this album as much as i do  check
out gavin degraw as well  his album chariot is arguably the best of 00  ebhp

First encoded review: [235, 1992, 123, 29, 106, 1404, 2140, 5, 162, 240, 154,
20, 43, 104, 7, 91, 290, 374, 96, 27, 1598, 719, 2603, 3312, 2260, 5, 1275, 77,
12, 8, 104, 3, 52, 17, 16, 4, 4605, 10, 1, 1098, 54, 154, 25, 982, 2180, 7582,
53, 4939, 7, 2261, 123, 568, 3313, 3251, 79, 185, 3651, 447, 18, 54, 677, 1528,
272, 26, 19, 519, 185, 9023, 1222, 10, 4, 34, 99, 1844, 7, 78, 466, 3, 3217, 5,
78, 25, 1, 113, 1201, 18, 36, 118, 61, 8065, 771, 5, 1, 8066, 7, 1, 571, 12,
171, 65, 69, 78, 8, 104, 24, 73, 24, 2, 69, 589, 47, 24, 70, 54, 104, 9, 7347,
1, 82, 7, 310]

Lenth before encoding 684
Lenth before encoding 121
wonderful 235
inspiring 1992
```

5.3. Unique tokens and Document Count

[75]:
```
print('Found %d unique words.' % len(tokenizer.word_index))
print("Documents",tokenizer.document_count)
```

```
Found 64191 unique words.
Documents 39578
```

5.4 Word Index (according to its frequency)

[76]:
```
print("Word Index", list(tokenizer.word_index.items())[0:5])
print(list(tokenizer.word_index.items())[-5:])
print(list(tokenizer.word_index.items())[500:505])
```

```
Word Index [('the', 1), ('i', 2), ('and', 3), ('a', 4), ('to', 5)]
[('revengeful', 64187), ('dices', 64188), ('laryngitis', 64189), ('guitarrist',
64190), ('punchless', 64191)]
[('mr', 501), ('working', 502), ('entire', 503), ('name', 504), ('totally',
505)]
```

[77]:
```
tokenizer.word_index["the"]
```

[77]:
```
1
```

5.5 Word Counts

[78]:
```
print("Word Counts", list(tokenizer.word_counts.items())[:5])
print(list(tokenizer.word_counts.items())[-5:])
print(list(tokenizer.word_counts.items())[100:105])

print(tokenizer.word_counts["the"])
```

```
Word Counts [('wonderful', 1724), ('inspiring', 132), ('music', 3601), ('so',
13166), ('many', 3935)]
[('revengeful', 1), ('dices', 1), ('laryngitis', 1), ('guitarrist', 1),
('punchless', 1)]
[('0', 3342), ('just', 10498), ('over', 3880), ('month', 620), ('now', 3644)]
164470
```

5.6 Count of Words in Documents

[79]:
```
print("Count of words in Documents")

print(list(tokenizer.word_docs.items())[0:5])
print(list(tokenizer.word_docs.items())[-5:])
```

```
Count of words in Documents
[('simplistic', 65), ('ebhp', 1), ('maybe', 1169), ('gorgeous', 104), ('tales',
120)]
[('revengeful', 1), ('laryngitis', 1), ('dices', 1), ('guitarrist', 1),
('punchless', 1)]
```

[80]:
```
tokenizer.word_docs["the"]
```

[80]:
```
35598
```

6. Word Embeddings

Represent words numerically in two ways:

1. Index words (single feature vector made up of integers)
2. Vectorize words:
    - Sparce Vector of one-hot-encoded word counts (BOW-SciKit Learn)
    - Word Embedding (each word is represented by an n-dimentional vector)
        - To allow for word meanings and language structure
        - common way to use text in neural networs

Word Embeddings: - map the statistical structure of the language used in the corpus - map semantic meaning into a geometric space - map semantically similar words close on the embedding space like numbers or colors - embedding captures the relationship between words - vector arithmetic becomes possible - A famous example: map King - Man + Woman = Queen

1.3 6. Padding with Keras

- text lengths are not be uniform
- a neural network requeres it
- select a maximum length
- pad shorter sentences with 0
- needed, to use batches effectively
- equal the length of the longest sentance

[81]:
```
%%time

MAX_LENGTH = max(len(train_ex) for train_ex in train_texts_b)

train_texts = pad_sequences(train_texts_b, maxlen=MAX_LENGTH)
val_texts = pad_sequences(val_texts_b, maxlen=MAX_LENGTH)
test_texts = pad_sequences(test_texts_b, maxlen=MAX_LENGTH)
```

```
CPU times: user 1.34 s, sys: 52.8 ms, total: 1.39 s
Wall time: 1.39 s
```

6.1 Maximum and Minimum Length

[82]:
```
print(max(len(train_ex) for train_ex in train_texts_b))
print(min(len(train_ex) for train_ex in train_texts_b))
```

```
241
3
```

6.2 Length Before Padding

[83]:
```
print(val_texts_b[0], end=" ")
print("\n")
print(len(val_texts_b[0]))
```

```
[4, 260, 1143, 159, 63, 7, 1, 1842, 3, 25, 295, 106, 25, 85, 1143, 52, 25, 4,
177, 7, 2069, 60, 831, 621, 91, 9, 356, 18, 1, 108, 66, 21, 247, 111, 2, 102, 1,
465, 7, 8, 15, 9, 35, 1728, 89, 2111, 66, 21, 196, 117, 38, 28, 163, 30, 94, 25,
536, 265, 18, 12, 1]
```

61

6.3 Length after Padding

```
[84]: print(val_texts[0], end=" ")
print("\n")
print(len(val_texts[0]))
print(len(val_texts[501]))
```

```
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    4  260
 1143  159   63    7    1 1842    3   25  295  106   25   85 1143   52
   25    4  177    7 2069   60  831  621   91    9  356   18    1  108
   66   21  247  111    2  102    1  465    7    8   15    9   35 1728
   89 2111   66   21  196  117   38   28  163   30   94   25  536  265
   18   12    1]
```

```
241
241
```

1.4 8. Baseline Model: Logistic Regression

- returns a list of (term, frequency) pairs for each distinct term in the corpus
- a sparce vector results
- no word embedding
- dimentions: training samples x size of the vocabulary
- therefore not padding necessery
- was able to show the resulting sparce matrix
- for a 500k train sample and 100k test sample, accuracy was 90,1% with a run time of 2 min.

1.5 7. Neural Network

Input Parameters into Neural Network

- Embedding Layer
- input dimention: size of the vocabulary
- output dimention: embedding size

- learned embedding
- for dense layer include length of input sequences

- Total Vocabulary:
- Selected Features:
- Lenght:
- Shape of Input:

[85]:
```
vocab_size = len(tokenizer.word_index) + 1
length = min(len(train_ex) for train_ex in train_texts)
embedding_dim = 100

print(vocab_size)
print(MAX_FEATURES)

print(length)
print(train_texts.shape)
```

```
64192
12000
241
(39578, 241)
```

[86]:
```
model = Sequential()
model.add(layers.Embedding(MAX_FEATURES, 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()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 241, 100)          1200000
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 237, 128)          64128
_________________________________________________________________
global_max_pooling1d_3 (Glob (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 11
```

```
=================================================================
Total params: 1,265,429
Trainable params: 1,265,429
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
%%time

history = model.fit(train_texts, train_labels,
                    epochs=3,
                    verbose=True,
                    validation_data=(val_texts, val_labels),
                    batch_size=512)
```

7.1 Accuracy Evaluation

[88]:
```
loss, accuracy = model.evaluate(val_texts, val_labels, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))

loss, accuracy = model.evaluate(test_texts, test_labels, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

```
Training Accuracy: 0.9063
Testing Accuracy:  0.9056
```

1.5.1 8. Architecture Simulation Study and Baseline Model

- Dense with no embedding: Acc: 0.50
- Dense with embedding: Acc: 0.85
- Added Conv layer: Acc: 0.90
- Log Reg: Acc: 0.90 (wait time)

2 Thank you for listening