C3W1_Assignment

June 8, 2023

1 Week 1: Explore the BBC News archive

Welcome! In this assignment you will be working with a variation of the BBC News Classification Dataset, which contains 2225 examples of news articles with their respective categories (labels).

Let's get started!

[1]:
```
# IMPORTANT: This will check your notebook's metadata for grading.
# Please do not continue the lab unless the output of this cell tells you to␣
↪proceed.
!python add_metadata.py --filename C3W1_Assignment.ipynb
```

```
Grader metadata detected! You can proceed with the lab!
```

NOTE: *To prevent errors from the autograder, you are not allowed to edit or delete non-graded cells in this notebook . Please only put your solutions in between the* ***###*** *START CODE HERE and* ***###*** *END CODE HERE code comments, and also refrain from adding any new cells.* ***Once you have passed this assignment*** *and want to experiment with any of the non-graded code, you may follow the instructions at the bottom of this notebook.*

[2]:
```
# grader-required-cell

import csv
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

Begin by looking at the structure of the csv that contains the data:

[3]:
```
# grader-required-cell

with open("./data/bbc-text.csv", 'r') as csvfile:
    print(f"First line (header) looks like this:\n\n{csvfile.readline()}")
    print(f"Each data point looks like this:\n\n{csvfile.readline()}")
```

```
First line (header) looks like this:

category,text

Each data point looks like this:
```

tech,tv future in the hands of viewers with home theatre systems plasma high-definition tvs and digital video recorders moving into the living room the way people watch tv will be radically different in five years time. that is according to an expert panel which gathered at the annual consumer electronics show in las vegas to discuss how these new technologies will impact one of our favourite pastimes. with the us leading the trend programmes and other content will be delivered to viewers via home networks through cable satellite telecoms companies and broadband service providers to front rooms and portable devices. one of the most talked-about technologies of ces has been digital and personal video recorders (dvr and pvr). these set-top boxes like the us s tivo and the uk s sky+ system allow people to record store play pause and forward wind tv programmes when they want. essentially the technology allows for much more personalised tv. they are also being built-in to high-definition tv sets which are big business in japan and the us but slower to take off in europe because of the lack of high-definition programming. not only can people forward wind through adverts they can also forget about abiding by network and channel schedules putting together their own a-la-carte entertainment. but some us networks and cable and satellite companies are worried about what it means for them in terms of advertising revenues as well as brand identity and viewer loyalty to channels. although the us leads in this technology at the moment it is also a concern that is being raised in europe particularly with the growing uptake of services like sky+. what happens here today we will see in nine months to a years time in the uk   adam hume the bbc broadcast s futurologist told the bbc news website. for the likes of the bbc there are no issues of lost advertising revenue yet. it is a more pressing issue at the moment for commercial uk broadcasters but brand loyalty is important for everyone. we will be talking more about content brands rather than network brands   said tim hanlon from brand communications firm starcom mediavest. the reality is that with broadband connections anybody can be the producer of content. he added: the challenge now is that it is hard to promote a programme with so much choice.   what this means said stacey jolna senior vice president of tv guide tv group is that the way people find the content they want to watch has to be simplified for tv viewers. it means that networks in us terms or channels could take a leaf out of google s book and be the search engine of the future instead of the scheduler to help people find what they want to watch. this kind of channel model might work for the younger ipod generation which is used to taking control of their gadgets and what they play on them. but it might not suit everyone the panel recognised. older generations are more comfortable with familiar schedules and channel brands because they know what they are getting. they perhaps do not want so much of the choice put into their hands mr hanlon suggested. on the other end you have the kids just out of diapers who are pushing buttons already - everything is possible and available to them   said mr hanlon. ultimately the consumer will tell the market they want.   of the 50 000 new gadgets and technologies being showcased at ces many of them are about enhancing the tv-watching experience. high-definition tv sets are everywhere and many new models of lcd (liquid crystal display) tvs have been launched with dvr capability built into them instead of being external boxes. one such example launched at the show is humax s 26-inch lcd tv with an 80-hour tivo dvr and dvd recorder. one of

```
the us s biggest satellite tv companies  directtv  has even launched its own
branded dvr at the show with 100-hours of recording capability  instant replay
and a search function. the set can pause and rewind tv for up to 90 hours. and
microsoft chief bill gates announced in his pre-show keynote speech a
partnership with tivo  called tivotogo  which means people can play recorded
programmes on windows pcs and mobile devices. all these reflect the increasing
trend of freeing up multimedia so that people can watch what they want  when
they want.
```

As you can see, each data point is composed of the category of the news article followed by a comma and then the actual text of the article.

1.1 Removing Stopwords

One important step when working with text data is to remove the **stopwords** from it. These are the most common words in the language and they rarely provide useful information for the classification process.

Complete the `remove_stopwords` below. This function should receive a string and return another string that excludes all of the stopwords provided. You only need to account for whitespace as the separation mechanism between words in the sentence.

[8]:
```python
# grader-required-cell

# GRADED FUNCTION: remove_stopwords
def remove_stopwords(sentence):
    """
    Removes a list of stopwords

    Args:
        sentence (string): sentence to remove the stopwords from

    Returns:
        sentence (string): lowercase sentence without the stopwords
    """
    # List of stopwords
    stopwords = ["a", "about", "above", "after", "again", "against", "all",␣
 ↪"am", "an", "and", "any", "are", "as", "at", "be", "because", "been",␣
 ↪"before", "being", "below", "between", "both", "but", "by", "could", "did",␣
 ↪"do", "does", "doing", "down", "during", "each", "few", "for", "from",␣
 ↪"further", "had", "has", "have", "having", "he", "he'd", "he'll", "he's",␣
 ↪"her", "here", "here's", "hers", "herself", "him", "himself", "his", "how",␣
 ↪"how's", "i", "i'd", "i'll", "i'm", "i've", "if", "in", "into", "is", "it",␣
 ↪"it's", "its", "itself", "let's", "me", "more", "most", "my", "myself",␣
 ↪"nor", "of", "on", "once", "only", "or", "other", "ought", "our", "ours",␣
 ↪"ourselves", "out", "over", "own", "same", "she", "she'd", "she'll",␣
 ↪"she's", "should", "so", "some", "such", "than", "that", "that's", "the",␣
 ↪"their", "theirs", "them", "themselves", "then", "there", "there's",␣
 ↪"these", "they", "they'd", "they'll", "they're", "they've", "this", "those",␣
 ↪"through", "to", "too", "under", "until", "up", "very", "was", "we", "we'd",␣
 ↪"we'll", "we're", "we've", "were", "what", "what's", "when", "when's",␣
 ↪"where", "where's", "which", "while", "who", "who's", "whom", "why",␣
 ↪"why's", "with", "would", "you", "you'd", "you'll", "you're", "you've",␣
 ↪"your", "yours", "yourself", "yourselves" ]

    # Sentence converted to lowercase-only
    sentence = sentence.lower()

    ### START CODE HERE
    words=sentence.split()
    words=[word for word in words if word not in stopwords]
    sentence=' '.join(words)


    ### END CODE HERE
    return sentence
```

[9]:
```
# grader-required-cell

# Test your function
remove_stopwords("I am about to go to the store and get any snack")
```

[9]:
```
'go store get snack'
```

Expected Output:

```
'go store get snack'
```

1.2 Reading the raw data

Now you need to read the data from the csv file. To do so, complete the `parse_data_from_file` function.

A couple of things to note: - You should omit the first line as it contains the headers and not data points. - There is no need to save the data points as numpy arrays, regular lists is fine. - To read from csv files use `csv.reader` by passing the appropriate arguments. - `csv.reader` returns an iterable that returns each row in every iteration. So the label can be accessed via row[0] and the text via row[1]. - Use the `remove_stopwords` function in each sentence.

[18]:
```
# grader-required-cell

# GRADED FUNCTION: parse_data_from_file
def parse_data_from_file(filename):
    """
    Extracts sentences and labels from a CSV file

    Args:
        filename (string): path to the CSV file

    Returns:
        sentences, labels (list of string, list of string): tuple containing␣
 ↪lists of sentences and labels
    """
    sentences = []
    labels = []
    with open(filename, 'r') as csvfile:
        ### START CODE HERE
        reader = csv.reader(csvfile, delimiter=',')
        next(reader)
        for row in reader:
            labels.append(remove_stopwords(row[0]))
            sentences.append(remove_stopwords(row[1]))


            ### END CODE HERE
    return sentences, labels
```

[19]:
```
# grader-required-cell

# Test your function

# With original dataset
sentences, labels = parse_data_from_file("./data/bbc-text.csv")

print("ORIGINAL DATASET:\n")
print(f"There are {len(sentences)} sentences in the dataset.\n")
print(f"First sentence has {len(sentences[0].split())} words (after removing␣
 ↪stopwords).\n")
print(f"There are {len(labels)} labels in the dataset.\n")
print(f"The first 5 labels are {labels[:5]}\n\n")
```

```python
# With a miniature version of the dataset that contains only first 5 rows
mini_sentences, mini_labels = parse_data_from_file("./data/bbc-text-minimal.
 ↪csv")

print("MINIATURE DATASET:\n")
print(f"There are {len(mini_sentences)} sentences in the miniature dataset.\n")
print(f"First sentence has {len(mini_sentences[0].split())} words (after␣
 ↪removing stopwords).\n")
print(f"There are {len(mini_labels)} labels in the miniature dataset.\n")
print(f"The first 5 labels are {mini_labels[:5]}")
```

```
ORIGINAL DATASET:

There are 2225 sentences in the dataset.

First sentence has 436 words (after removing stopwords).

There are 2225 labels in the dataset.

The first 5 labels are ['tech', 'business', 'sport', 'sport', 'entertainment']


MINIATURE DATASET:

There are 5 sentences in the miniature dataset.

First sentence has 436 words (after removing stopwords).

There are 5 labels in the miniature dataset.

The first 5 labels are ['tech', 'business', 'sport', 'sport', 'entertainment']
```

Expected Output:

```
ORIGINAL DATASET:

There are 2225 sentences in the dataset.

First sentence has 436 words (after removing stopwords).

There are 2225 labels in the dataset.

The first 5 labels are ['tech', 'business', 'sport', 'sport', 'entertainment']


MINIATURE DATASET:

There are 5 sentences in the miniature dataset.
```

```
First sentence has 436 words (after removing stopwords).

There are 5 labels in the miniature dataset.

The first 5 labels are ['tech', 'business', 'sport', 'sport', 'entertainment']
```

Note: The test above should take less than 5 seconds to run. If your implementation runs longer than that, the grader might time out when you submit. Try to improve it and if it still takes a while to execute, you can click the text below for some hints.

Click for hints

2 Read the CSV file and specify the comma character as the delimiter

reader = csv.reader(None, delimiter=None)

```
# skip the first line using the next() function (1 line of code)
    None

    # start for loop and iterate over each row of the reader
    for row in reader:

        # append the first element of the row to the `labels` list (1 line of code)
        None

        # use the remove_stopwords() function on the second element of the
        # row then append to the `sentences` list (1 to 3 lines of code)
        None
```

2.1 Using the Tokenizer

Now it is time to tokenize the sentences of the dataset.

Complete the `fit_tokenizer` below.

This function should receive the list of sentences as input and return a Tokenizer that has been fitted to those sentences. You should also define the "Out of Vocabulary" token as `<OOV>`.

[22]:
```python
# grader-required-cell

# GRADED FUNCTION: fit_tokenizer
def fit_tokenizer(sentences):
    """
    Instantiates the Tokenizer class

    Args:
        sentences (list): lower-cased sentences without stopwords

    Returns:
        tokenizer (object): an instance of the Tokenizer class containing the␣
 ↪word-index dictionary
    """
    ### START CODE HERE
    # Instantiate the Tokenizer class by passing in the oov_token argument
    tokenizer = Tokenizer(oov_token='<OOV>')

    # Fit on the sentences
    tokenizer.fit_on_texts(sentences)
    ### END CODE HERE
    return tokenizer
```

[23]:
```
# grader-required-cell

tokenizer = fit_tokenizer(sentences)
word_index = tokenizer.word_index

print(f"Vocabulary contains {len(word_index)} words\n")
print("<OOV> token included in vocabulary" if "<OOV>" in word_index else "<OOV>␣
 ↪token NOT included in vocabulary")
```

```
Vocabulary contains 29714 words

<OOV> token included in vocabulary
```

Expected Output:

```
Vocabulary contains 29714 words

<OOV> token included in vocabulary
```

[24]:
```
# grader-required-cell

# GRADED FUNCTION: get_padded_sequences
def get_padded_sequences(tokenizer, sentences):
    """
    Generates an array of token sequences and pads them to the same length

    Args:
        tokenizer (object): Tokenizer instance containing the word-index␣
 ↪dictionary
        sentences (list of string): list of sentences to tokenize and pad

    Returns:
        padded_sequences (array of int): tokenized sentences padded to the same␣
 ↪length
    """
    ### START CODE HERE
    # Convert sentences to sequences
    sequences = tokenizer.texts_to_sequences(sentences)

    # Pad the sequences using the post padding strategy
    padded_sequences = pad_sequences(sequences,padding='post')
    ### END CODE HERE

    return padded_sequences
```

[25]:
```
# grader-required-cell

padded_sequences = get_padded_sequences(tokenizer, sentences)
print(f"First padded sequence looks like this: \n\n{padded_sequences[0]}\n")
print(f"Numpy array of all sequences has shape: {padded_sequences.shape}\n")
print(f"This means there are {padded_sequences.shape[0]} sequences in total and␣
↪each one has a size of {padded_sequences.shape[1]}")
```

```
First padded sequence looks like this:

[  96  176 1157 …    0    0    0]

Numpy array of all sequences has shape: (2225, 2438)

This means there are 2225 sequences in total and each one has a size of 2438
```

Expected Output:

```
First padded sequence looks like this:

[  96  176 1157 ...    0    0    0]

Numpy array of all sequences has shape: (2225, 2438)

This means there are 2225 sequences in total and each one has a size of 2438
```

[26]:
```
# grader-required-cell

# GRADED FUNCTION: tokenize_labels
def tokenize_labels(labels):
    """
    Tokenizes the labels

    Args:
        labels (list of string): labels to tokenize

    Returns:
        label_sequences, label_word_index (list of string, dictionary):␣
↪tokenized labels and the word-index
    """
    ### START CODE HERE

    # Instantiate the Tokenizer class
    # No need to pass additional arguments since you will be tokenizing the␣
↪labels
    label_tokenizer = Tokenizer()

    # Fit the tokenizer to the labels
    label_tokenizer.fit_on_texts(labels)

    # Save the word index
    label_word_index = label_tokenizer.word_index

    # Save the sequences
    label_sequences = label_tokenizer.texts_to_sequences(labels)

    ### END CODE HERE

    return label_sequences, label_word_index
```

[27]:
```
# grader-required-cell

label_sequences, label_word_index = tokenize_labels(labels)
print(f"Vocabulary of labels looks like this {label_word_index}\n")
print(f"First ten sequences {label_sequences[:10]}\n")
```

```
Vocabulary of labels looks like this {'sport': 1, 'business': 2, 'politics': 3,
'tech': 4, 'entertainment': 5}

First ten sequences [[4], [2], [1], [1], [5], [3], [3], [1], [1], [5]]
```

Expected Output:

```
Vocabulary of labels looks like this {'sport': 1, 'business': 2, 'politics': 3, 'tech': 4, 'en

First ten sequences [[4], [2], [1], [1], [5], [3], [3], [1], [1], [5]]
```

Congratulations on finishing this week's assignment!

You have successfully implemented functions to process various text data processing ranging from pre-processing, reading from raw files and tokenizing text.

Keep it up!

Please click here if you want to experiment with any of the non-graded code.

Important Note: Please only do this when you've already passed the assignment to avoid problems with the autograder.

On the notebook's menu, click “View” > “Cell Toolbar” > “Edit Metadata”

Hit the “Edit Metadata” button next to the code cell which you want to lock/unlock

Set the attribute value for “editable” to:

“true” if you want to unlock it

“false” if you want to lock it

```
    </li>
    <li> On the notebook’s menu, click "View" > "Cell Toolbar" > "None" </li>
</ol>
<p> Here's a short demo of how to do the steps above:
    <br>
    <img src="https://drive.google.com/uc?export=view&id=14Xy_Mb17CZVgzVAgq7NCjMVBvSae3xO1" al
```