assignment1_part1-Copy1

February 4, 2022

[ ]:
```
version = "REPLACE_PACKAGE_VERSION"
```

---

1 Assignment 1 Part 1: N-gram Language Models (40 pts)

In this assignment, we're going to train an n-gram language model that is able to "imitate" William Shakespeare's writing.

[1]:
```
# Configure nltk

import nltk

nltk_data_path = "assets/nltk_data"
if nltk_data_path not in nltk.data.path:
    nltk.data.path.append(nltk_data_path)
```

1.1 Question 1: Load the dataset (10 pts)

As the first step towards imitating Shakespeare's writing, you will create a function called `load_data` that loads his original *Sonnets* from `assets/gutenberg/THE_SONNETS.txt`. This function should accomplish the following:

- **Extract sentences from the data file.** Of course, depending on the nature of the task at hand, what constitutes a *sentence* can vary. In the context of this assignment, we will define a sentence as just a line of a sonnet, regardless of the punctuation at end. In addition, we will ignore the boundaries of the sonnets — that is, we are not dealing with 154 individual *sonnets* but rather $154 \times 14 = 2156$ *sentences* (actually only 2155 sentences, as *Sonnet 99* has 15 lines but *Sonnet 126* has only 12). We encourage you to explore alternative definitions of a sentence on your own; for example, an entire sonnet could be modelled as a sentence. Finally, make sure that the newline character `\n` at the end of each line is dropped.
- **Tokenise each extracted sentence.** While it's ambiguous what a sentence is, what constitutes a "*word*" is even more task-dependent. Do punctuations count as "words"? Are two "words" with the same spelling but different casing considered identical? Since what a text file contains is nothing more than a squence of characters, there are many possible ways of grouping these characters to form "words" that are subsequently taken as input by a program. To distinguish what's actually taken as input by a program from a linguistic word, we call the former a *token*. The process of producing a list of tokens out of a sentence is then called

tokenisation. At this step, you will first lower-case each sentence extracted from the previous step entirely and then tokenise each lower-cased sentence. You may use any tokeniser of your choice, such as `word_tokenize` from the `nltk` library. The grading of the assignment doesn't depend on your choice of the tokeniser.

This function should return a `list` of length 2155, where each element is a `list` of `str` representing the tokens of each sentence as produced by the tokeniser of your choice. An example output would be:

```
[['from', 'fairest', 'creatures', 'we', 'desire', 'increase', ','],
 ['that', 'thereby', 'beauty', '’', 's', 'rose', 'might', 'never', 'die', ','],
 ...
 ['came', 'there', 'for', 'cure', 'and', 'this', 'by', 'that', 'i', 'prove', ','],
 ['love', '’', 's', 'fire', 'heats', 'water', ',', 'water', 'cools', 'not', 'love', '.']]
```

[1]:
```
def load_data():
    """
    Load text data from a file and produce a list of token lists
    """

    from nltk.tokenize import sent_tokenize, TweetTokenizer, word_tokenize
    import re
    sentences_raw = []
    sentences_processed = []
    sentences_split_into_words = []

    # YOUR CODE HERE
    # read file, tokenize into sentences without processing


    return sentences_split_into_words
stu_ans = load_data()
```

[2]:
```
# Autograder test

stu_ans = load_data()

assert isinstance(stu_ans, list), "Q1: Your function should return a list. "
assert len(stu_ans) == 2155, "Q1: There should be 2155 sentences. "

for index, tokens in enumerate(stu_ans):
    assert isinstance(tokens, list), f"Q1: The element at index {index} of your␣
 ↪answer list should be a list. "
    for token in tokens:
        if token.isalpha():
            assert token.islower(), f'Q1: Token "{token}" in the sentence at␣
 ↪index {index} is not lower-cased. '
```

```
        assert token != "\n", f'Q1: You should drop the "\\n" character in the␣
↪sentence at index {index}. '

del stu_ans
```

```
      ␣
↪---------------------------------------------------------------------------

       AssertionError                            Traceback (most recent call␣
↪last)

       /tmp/ipykernel_189/3019820287.py in <module>
         4
         5 assert isinstance(stu_ans, list), "Q1: Your function should return a␣
↪list. "
    ----> 6 assert len(stu_ans) == 2155, "Q1: There should be 2155 sentences. "
         7
         8 for index, tokens in enumerate(stu_ans):


       AssertionError: Q1: There should be 2155 sentences.
```

1.2 Question 2: Build vocabulary (15 pts)

Next, we need a "vocabulary" that contains all the unique tokens. Moreover, as mentioned in the lecture, we often pad a sentence with <s> and </s> to indicate its start and end when working with n-gram language models; therefore, these two special tokens should also be included in our vocabulary. Complete the function below to build a vocabulary. The order in which the tokens are stored doesn't matter.

This function should return a `list` of unique tokens, including <s> and </s>. An example output would be:

`['refuse', 'enjoyed', ..., '<s>', '</s>']`

[365]:
```
def build_vocab(sentences):
    """
    Take a list of sentences and return a vocab
    """

    vocab = []

    # YOUR CODE HERE
    #print(len(sentences), sentences[:10])
    from itertools import chain
```

```
    # Make list of list into flat list
    #flat_sentences = []
    #for sublist in sentences:
    #    for word in sublist:
    #        flat_sentences.append(word)
    #print(flat_sentences)

    #Method 2 for flattening
    flat_sentences = list(chain.from_iterable(sentences))
    #print(flat_sentences)

    # Convert to set
    setofwords = set(flat_sentences)

    # Convert back to list and adding 2 padding tokens
    vocab = list(setofwords)
    vocab.extend(('<s>', '</s>'))
    #print(vocab)

    return vocab

stu_sents = load_data()
stu_vocab = build_vocab(stu_sents)
```

```
2147 [['from fairest creatures we desire increase,'], ['that thereby beauty's
rose might never die,'], ['but as the riper should by time decease,'], ['his
tender heir might bear his memory:'], ['but thou contracted to thine own bright
eyes,']]
2155 [['from', 'fairest', 'creatures', 'we', 'desire', 'increase', ','],
['that', 'thereby', 'beauty', ''', 's', 'rose', 'might', 'never', 'die', ','],
['but', 'as', 'the', 'riper', 'should', 'by', 'time', 'decease', ','], ['his',
'tender', 'heir', 'might', 'bear', 'his', 'memory', ':'], ['but', 'thou',
'contracted', 'to', 'thine', 'own', 'bright', 'eyes', ','']]
```

[366]:
```
# Autograder tests

stu_sents = load_data()
stu_vocab = build_vocab(stu_sents)

assert isinstance(stu_vocab, list), "Q2: Your function should return a list. "
assert stu_vocab, "Q2: Your vocab is empty. "
assert "<s>" in stu_vocab, "Q2: Remember to include special token <s>. "
assert "</s>" in stu_vocab, "Q2: Remember to include special token </s>. "
assert len(set(stu_vocab)) == len(stu_vocab), "Q2: Your vocab contains␣
 ↪duplicated tokens. "
```

```
# Some hidden tests

del stu_sents, stu_vocab
```

```
2147 [['from fairest creatures we desire increase,'], ['that thereby beauty's
rose might never die,'], ['but as the riper should by time decease,'], ['his
tender heir might bear his memory:'], ['but thou contracted to thine own bright
eyes,']]
2155 [['from', 'fairest', 'creatures', 'we', 'desire', 'increase', ','],
['that', 'thereby', 'beauty', ''', 's', 'rose', 'might', 'never', 'die', ','],
['but', 'as', 'the', 'riper', 'should', 'by', 'time', 'decease', ','], ['his',
'tender', 'heir', 'might', 'bear', 'his', 'memory', ':'], ['but', 'thou',
'contracted', 'to', 'thine', 'own', 'bright', 'eyes', ','']]
```

1.3 Question 3: Generate all n-grams (15 pts)

Now let's write a function to generate all n-grams for each sentence. This can be accomplished in two steps: * **Pad each sentence with <s> and </s> for** $n \geq 2$. You need $n-1$ paddings on both ends of a sentence, so that there are two n-grams that model the first and the last token, respectively. You may implement the padding function yourself or use the `pad_both_ends` function from the `nltk` library.

- **Generate n-grams on the padded sentences**. Check out the `ngrams` function from `nltk`. For a sentence with ℓ tokens excluding paddings, the maximum number of possible n-grams generated from its padded version should be $\ell + n - 1$. Think about why.

Complete the function below to return a `list`, where each element of the `list` is a either a list or a "generator object" produced by the `ngrams` function, representing a sequence of all n-grams generated from each appropriately padded sentence. If the argument `n=2`, the autograder would accept either of the example outputs below:

```
[<generator object ngrams at 0x7f77ed778b10>,
 <generator object ngrams at 0x7f77e7d934f8>,
 ...
 <generator object ngrams at 0x7f77e7d751b0>,
 <generator object ngrams at 0x7f77e7d75228>]
```

OR

```
[
    [('<s>', 'from'), ('from', 'fairest'), ('fairest', 'creatures'), ('creatures', 'we'), ('we
     ('desire', 'increase'), ('increase', ','), (',', '</s>')],

    [('<s>', 'that'), ('that', 'thereby'), ('thereby', 'beauty'), ('beauty', ''), (''', 's'),
     ('rose', 'might'), ('might', 'never'), ('never', 'die'), ('die', ','), (',', '</s>')],

    ...

    [('<s>', 'came'), ('came', 'there'), ('there', 'for'), ('for', 'cure'), ('cure', 'and'), (
     ('this', 'by'), ('by', 'that'), ('that', 'i'), ('i', 'prove'), ('prove', ','), (',', '</s
```

```
    [('<s>', 'love'), ('love', ''), ('', 's'), ('s', 'fire'), ('fire', 'heats'), ('heats', '
     ('water', ','), (',', 'water'), ('water', 'cools'), ('cools', 'not'), ('not', 'love'), ('
     ('.', '</s>')]
]
```

[3]:
```python
def build_ngrams(n, sentences):
    """
    Take a list of unpadded sentences and create all n-grams as specified by␣
 ↪the argument "n" for each sentence
    """

    all_ngrams = []

    # YOUR CODE HERE
    from nltk.util import everygrams, pad_sequence
    from nltk.lm.preprocessing import pad_both_ends
    from itertools import chain
    from nltk import ngrams

    # Step 1 Pad each flattened sentence with <s> and </s> for  2 with
    # pad_both_ends function from the nltk library.

    return all_ngrams
stu_n = 1
stu_sents = load_data()
stu_ngrams = build_ngrams(stu_n, stu_sents)
```

[4]:
```python
# Autograder tests

stu_n = 4
stu_sents = load_data()
old_hash = hash(tuple([tuple(sent) for sent in stu_sents]))
stu_ngrams = build_ngrams(stu_n, stu_sents)

assert isinstance(stu_ngrams, list), "Q3: Your function should return a list. "
assert stu_ngrams, "Q3: Your ngrams list is empty. "

# Check that your function does not modify 'stu_sents' in place
new_hash = hash(tuple([tuple(sent) for sent in stu_sents]))
assert new_hash == old_hash, "Q3: Your function should not modify its argument␣
 ↪'sentences' in place. "

# Some hidden tests


del stu_n, stu_sents, stu_ngrams
```

```
      ␣
↪------------------------------------------------------------------------

       AssertionError                            Traceback (most recent call␣
↪last)

       /tmp/ipykernel_189/3026585918.py in <module>
         7
         8 assert isinstance(stu_ngrams, list), "Q3: Your function should␣
↪return a list. "
   ----> 9 assert stu_ngrams, "Q3: Your ngrams list is empty. "
        10
        11 # Check that your function does not modify 'stu_sents' in place


       AssertionError: Q3: Your ngrams list is empty.
```

Now that we have completed all the preparation work for imitating William Shakespeare's writing, it's time to take a break. We will resume in Assignment 1 Part 2 to finish training an n-gram language model. See you there!

[ ]:

[ ]: