

Deep Learning for Text

2021.7.13

11.1 Natural Language Processing

Using machine learning and large datasets to give computers the ability to ingest a piece of language as input and return something useful, like predicting:

What's the topic of this text: text classification

Does this text contain abuse: content filtering

Does this text sound positive or negative: sentiment analysis

What would be the next word in this incomplete sentence: language modeling

How would you say this in German: translation

How would you summarize this article in one paragraph: summarization

11.2 Preparing the text data

Vectorizing text is the process of transforming text into numeric tensors. Text vectorization processes come in many shapes and form, but they all follow the same template:

First, standardizing the text to make it easier to process, for instance by converting it to lowercase or removing punctuation.

Second, splitting the text into units(called tokens), such as characters, words, or groups of words. This is called tokenization.

Third, converting each such token into a numerical vector. This will involve first **indexing** all tokens present in the data.

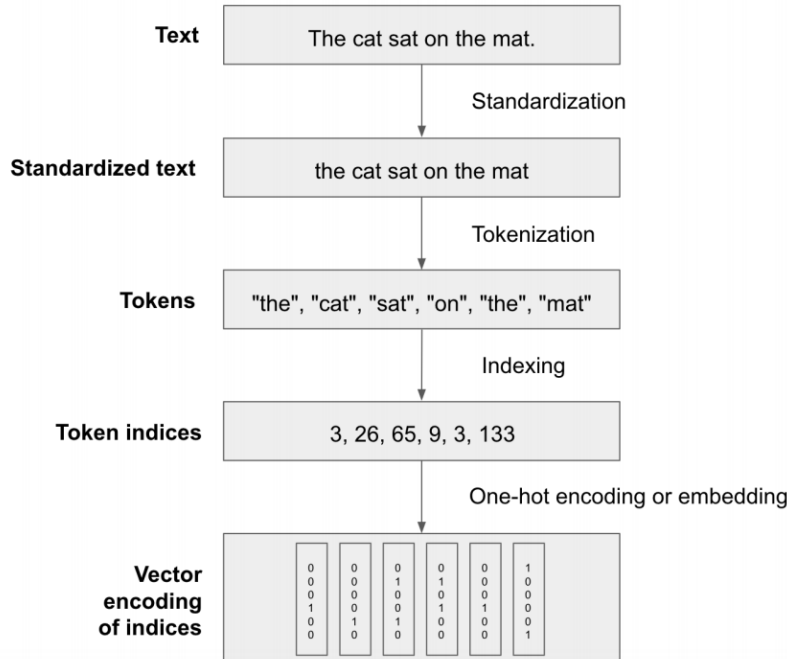


Figure 11.1 From raw text to vectors

11.2.1 Text Standardization

Text standardization is a basic form of feature engineering that aims to erase encoding differences that you don't want model to have to deal with.

One of the simplest and most widespread standardization scheme is "convert to lowercase and remove punctuation characters".

Another common transfer is to convert special characters to a standard form.

A much more advanced standardization pattern, that is more rarely used is **stemming**: converting variations of a term into a single shared representations, like turning caught and been catching into [catch].

Of course, standardization may also be erasing some amount of information. For instance, when writing a model that extracts questions from interview articles, it should definitely treat ? as a separate token instead of dropping it.

11.2.2 Text Spliting (tokenization)

Tokenization in two different ways:

Word-level tokenization: where tokens are space-separated substrings. A variant of this is to further split words into subword when applicable.

N-gram tokenization: where tokens are group of N consecutive words. For instance, "the cat" or "he was" would be 2-gram tokens.

There are two kinds of text-preprocessing models: sequence model and bag-of-words model. When building a sequence model, using word-level tokenization, and if building a bag-of-words models, using N-gram tokenization.

11.2.3 Vocabulary indexing

Once text is split into tokens, we need to encode each token into a numerical representation. In practice, the way we'd do about it is to build an index of all terms found in the training data, and assign a unique integer to each entry in the vocabulary.

```
1 vocabulary = {}
2 for text in dataset:
3     text = standardize(text)
4     tokens = tokenize(text)
5     for token in tokens:
6         if token not in vocabulary:
7             vocabulary[token] = len(vocabulary)
```

Then, we can convert that integer into a vector encoding that can be processed by a neural network:

```
1 def one_hot_encode_token(token):
2     vector = np.zeros((len(vocabulary),))
3     token_index = vocabulary[token]
4     vector[token_index] = 1
5     return vector
```

To index the vocabulary of a text corpus, just call the *adapt()* method of the layer with a *Dataset* object, that yield strings, or just with on list of Python strings.

The first two entries in the vocabulary are the mask token (index 0) and the OOV token (index 1). Entries in the vocabulary list are sorted by frequency.

11.3 Representing groups of words: sets and sequences

A much more problematic question is how to encode the way *words are weaved into sentences*: word order