Introduction to preprocessing for text

DEEP LEARNING FOR TEXT WITH PYTORCH

Shubham Jain
Data Scientist

What we will learn

- Text classification
- Text generation
- Encoding
- Deep learning models for text
- Transformer architecture
- Protecting models

Use cases:

- Sentiment analysis
- Text summarization
- Machine translation

What you should know

Prerequisite course: **Intermediate Deep Learning with PyTorch**

- Deep learning models with PyTorch
- Training and evaluation loops
- Convolutional neural networks (CNNs) and recurrent neural networks (RNNs)

Text processing pipeline

Text processing pipeline

- Clean and prepare text

PyTorch and NLTK

PyTorch

NLTK

- Natural language tooklit
  - Transform raw text to processed text

Preprocessing techniques

- Tokenization
- Stop word removal
- Stemming
- Rare word removal

Tokenization

- **Tokens** or words are extracted from text
- Tokenization using `torchtext`

```
from torchtext.data.utils import get_tokenizer
tokenizer = get_tokenizer("basic_english")
tokens = tokenizer("I am reading a book now. I love to read books!")
print(tokens)
```

```
["I", "am", "reading", "a", "book", "now", ".", "I", "love", "to", "read",
"books", "!"]
```

Stop word removal

- Eliminate common words that do not contribute to the meaning
- Stop words: "a", "the", "and", "or", and more

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
tokens = ["I", "am", "reading", "a", "book", "now", ".", "I", "love", "to", "read",
"books", "!"]
filtered_tokens = [token for token in tokens if token.lower() not in stop_words]
print(filtered_tokens)
```

```
["reading", "book", ".", "love", "read", "books", "!"]
```

Stemming

- Reducing words to their base form
- For example: "running", "runs", "ran" becomes **run**

```
import nltk
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
filtered_tokens = ["reading", "book", ".", "love", "read", "books", "!"]
stemmed_tokens = [stemmer.stem(token) for token in filtered_tokens]
print(stemmed_tokens)
```

```
["read", "book", ".", "love", "read", "book", "!"]
```

Rare word removal

- Removing infrequent words that don't add value

```
from nltk.probability import FreqDist
stemmed_tokens= ["read", "book", ".", "love", "read", "book", "!"]
freq_dist = FreqDist(stemmed_tokens)
threshold = 2
common_tokens = [token for token in stemmed_tokens if freq_dist[token] > threshold]
print(common_tokens)
```

```
["read", "book", "read", "book"]
```

Preprocessing techniques

Tokenization, stopword removal, stemming, and rare word removal

- Reduce features
- Cleaner, more representative datasets
- More techniques exist

Let's practice!

DEEP LEARNING FOR TEXT WITH PYTORCH

Encoding text data

DEEP LEARNING FOR TEXT WITH PYTORCH

Shubham Jain
Data Scientist

Text encoding

- Convert text into machine-readable numbers
- Enable analysis and modeling

Human-Readable

Pet
Cat
Dog
Turtle
Fish
Cat

Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

Machine-Readable

Encoding techniques

- **One-hot encoding:** transforms words into unique numerical representations
- **Bag-of-Words (BoW):** captures word frequency, disregarding order
- **TF-IDF:** balances uniqueness and importance
- **Embedding:** converts words into vectors, capturing semantic meaning (Chapter 2)

One-hot encoding

- Mapping each word to a distinct vector
- **Binary vector:**
  - 1 for the presence of a word
  - 0 for the absence of a word
- ['cat', 'dog', 'rabbit']
  - 'cat' [1, 0, 0]
  - 'dog' [0, 1, 0]
  - 'rabbit' [0, 0, 1]

One-hot encoding with PyTorch

```
import torch
vocab = ['cat', 'dog', 'rabbit']
vocab_size = len(vocab)
one_hot_vectors = torch.eye(vocab_size)
one_hot_dict = {word: one_hot_vectors[i] for i, word in enumerate(vocab)}
print(one_hot_dict)
```

```
{'cat': tensor([1., 0., 0.]),
  'dog': tensor([0., 1., 0.]),
  'rabbit': tensor([0., 0., 1.])}
```

Bag-of-words

- **Example:** "The cat sat on the mat"
- **Bag-of-words:**
  - {'the': 2, 'cat': 1, 'sat': 1, 'on': 1, 'mat': 1}

- Treating each document as an unordered collection of words
- Focuses on **frequency**, not order

CountVectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
corpus = ['This is the first document.', 'This document is the second document.',
'And this is the third one.', 'Is this the first document?']
X = vectorizer.fit_transform(corpus)
print(X.toarray())
print(vectorizer.get_feature_names_out())
```

```
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
 ['and' 'document' 'first' 'is' 'one' 'second' 'the' 'third' 'this']
```

TF-IDF

- Term Frequency-Inverse Document Frequency
  - Scores the importance of words in a document
  - Rare words have a higher score
  - Common ones have a lower score
  - Emphasizes informative words

TfidfVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
corpus = ['This is the first document.','This document is the second document.',
'And this is the third one.','Is this the first document?']
X = vectorizer.fit_transform(corpus)
print(X.toarray())
print(vectorizer.get_feature_names_out())
```

```
[[0.         0.         0.68091856 0.51785612 0.51785612 0.        ]
 [0.         0.         0.         0.51785612 0.51785612 0.68091856]
 [0.85151335 0.42575668 0.         0.32274454 0.32274454 0.        ]
 [0.         0.         0.68091856 0.51785612 0.51785612 0.        ]]
['and' 'document' 'first' 'is' 'one' 'second']
```

TfidfVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
corpus = ['This is the first document.','This document is the second document.',
'And this is the third one.','Is this the first document?']
X = vectorizer.fit_transform(corpus)
print(X.toarray())
print(vectorizer.get_feature_names_out())
```

```
[[0.         0.         0.68091856 0.51785612 0.51785612 0.        ]
 [0.         0.         0.          0.51785612 0.51785612 0.68091856]
 [0.85151335 0.42575668 0.         0.32274454 0.32274454 0.        ]
 [0.         0.         0.68091856 0.51785612 0.51785612 0.        ]]
['and' 'document' 'first' 'is' 'one' 'second']
```

Encoding techniques

Techniques: One-hot encoding, bag-of-words, and TF-IDF

- Allows models to understand and process text
- Choose one technique to avoid redudancy
- More techniques exist

Let's practice!

DEEP LEARNING FOR TEXT WITH PYTORCH

Introduction to building a text processing pipeline

DEEP LEARNING FOR TEXT WITH PYTORCH

Shubham Jain
Data Scientist

Recap: preprocessing

- Preprocessing:
  - Tokenization
  - Stopword removal
  - Stemming
  - Rare word removal

Text processing pipeline

- Encoding:
  - One-hot encoding
  - Bag-of-words
  - TF-IDF
- Embedding

Text processing pipeline

- Dataset as a container for processed and encoded text
- DataLoader: batching, shuffling and multiprocessing

Recap: implementing Dataset and DataLoader

```python
# Import libraries
from torch.utils.data import Dataset, DataLoader
# Create a class
class TextDataset(Dataset):
    def __init__(self, text):
        self.text = text
    def __len__(self):
        return len(self.text)
    def __getitem__(self, idx):
        return self.text[idx]
```

Recap: integrating Dataset and DataLoader

```
dataset = TextDataset(encoded_text)
dataloader = DataLoader(dataset, batch_size=2, shuffle=True)
```

Using helper functions

```python
def preprocess_sentences(sentences):
    processed_sentences = []
    for sentence in sentences:
        sentence = sentence.lower()
        tokens = tokenizer(sentence)
        tokens = [token for token in tokens
                  if token not in stop_words]
        tokens = [stemmer.stem(token)
                  for token in tokens]
        freq_dist = FreqDist(tokens)
        threshold = 2
        tokens = [token for token in tokens if
        freq_dist[token] > threshold]
        processed_sentences.append(
                   ' '.join(tokens))
    return processed_sentences
```

```python
def encode_sentences(sentences):
    vectorizer = CountVectorizer()
    X = vectorizer.fit_transform(sentences)
    encoded_sentences = X.toarray()
    return encoded_sentences, vectorizer
```

```python
def extract_sentences(data):
    sentences = re.findall(r'[A-Z][^.!?]*[.!?]',
                           data)
    return sentences
```

Constructing the text processing pipeline

```
def text_processing_pipeline(text):
    tokens = preprocess_sentences(text)
    encoded_sentences, vectorizer = encode_sentences(tokens)
    dataset = TextDataset(encoded_sentences)
    dataloader = DataLoader(dataset, batch_size=2, shuffle=True)
    return dataloader, vectorizer
```

Applying the text processing pipeline

```
text_data = "This is the first text data. And here is another one."
sentences = extract_sentences(text_data)
dataloaders, vectorizer = [text_processing_pipeline(text) for text in sentences]
print(next(iter(dataloader))[0, :10])
```

```
[[1, 1, 1, 1, 1], [0, 0, 0, 1, 1]]
```

Text processing pipeline: it's a wrap!

Let's practice!

DEEP LEARNING FOR TEXT WITH PYTORCH