Natural Language Processing

Last update: 24 September 2024

Academic Year 2024 – 2025

Alma Mater Studiorum · University of Bologna

Contents

1 Basic text processing

Text normalization Operations such as:

Tokenization Split a sentence in tokens. Tokenization

| **Remark.** Depending on the approach, a token is not always a word.

Lemmatization/stemming Convert words to their canonical form. Lemmatization/stemming

| **Example.** $\{\texttt{sang}, \texttt{sung}, \texttt{sings}\} \mapsto \texttt{sing}$

Sentence segmentation Split a text in sentences. Sentence segmentation

| **Remark.** A period does not always signal the end of a sentence.

1.1 Regular expressions

Regular expression (regex) Formal language to describe string patterns. Regular expression (regex)

1.1.1 Basic operators

Disjunction (brackets) Match a single character between square brackets `[]`.

| **Example.** `/[wW]oodchuck/` matches `Woodchuck` and `woodchuck`.

Range Match a single character from a range of characters or digits.

| **Example.**
- `/[A-Z]/` matches a single upper case letter.
- `/[a-z]/` matches a single lower case letter.
- `/[0-9]/` matches a single digit.

Negation Match the negation of a pattern.

| **Example.** `/[^A-Z]/` matches a single character that is not an upper case letter.

Disjunction (pipe) Disjunction of regular expressions separated by `|`.

| **Example.** `/groundhog|woodchuck/` matches `groundhog` and `woodchuck`.

Wildcards

Optional A character followed by `?` can be matched optionally.

| **Example.** `/woodchucks?/` matches `woodchuck` and `woodchucks`.

Any `.` matches any character.

Kleene `*` A character followed by `*` can be matched zero or more times.

Kleene `+` A character followed by `+` must be matched at least once.

Counting A character followed by `{n,m}` must be matched from n to m times.

Example.

- `{n}` matches exactly n instances of the previous character.
- `{n,m}` matches from n to m instances of the previous character.
- `{n,}` matches at least n instances of the previous character.
- `{,m}` matches at most m instances of the previous character.

Anchors

Start of line `^` matches only at the start of line.

Example. `/^a/` matches a but not `ba`.

End of line `$` matches only at the end of line.

Example. `/a$/` matches a but not `ab`.

Word boundary `\b` matches a word boundary character.

Word non-boundary `\B` matches a word non-boundary character.

Aliases

- `\d` matches a single digit (same as `[0-9]`).
- `\D` matches a single non-digit (same as `[^\d]`).
- `\w` matches a single alphanumeric or underscore character (same as `[a-zA-Z0-9_]`).
- `\w` matches a single non-alphanumeric and non-underscore character (same as `[^\w]`).
- `\s` matches a single whitespace (space or tab).
- `\S` matches a single non-whitespace.

Capture group Operator to refer to previously matched substrings.

Example. In the regex `/the (.*)er they were, the \1er they will be/`, `\1` should match the same content matched by `(.*)`.

1.2 Tokenization

Lemma Words with the same stem and roughly the same semantic meaning. Lemma

Example. `cat` and `cats` are the same lemma.

Wordform Orthographic appearance of a word. Wordform

Example. `cat` and `cats` do not have the same wordform.

Vocabulary Collection of text elements, each indexed by an integer. Vocabulary

Remark. To reduce the size of a vocabulary, words can be reduced to lemmas.

Type / Wordtype Element of a vocabulary (i.e., wordforms in the vocabulary). Type / Wordtype

Token Instance of a type in a text. Token

Genre Topic of a text corpus (e.g., short social media comments, books, Wikipedia pages, ...). Genre

Remark (Herdan's law)**.** Given a corpus with N tokens, a vocabulary V over that corpus roughly have size:

$$|V| = kN^{\beta}$$

where the typical values are $10 \leq k \leq 100$ and $0.4 \leq \beta \leq 0.6$.

Stopwords Frequent words that can be dropped. Stopwords

> **Remark.** If semantics is important, stopwords should be kept. LLMs keep stopwords.

Rule-based tokenization Hand-defined rules for tokenization. Rule-based tokenization

> **Remark.** For speed, simple tokenizers use regex.

Data-driven tokenization Determine frequent tokens from a large text corpus. Data-driven tokenization

1.2.1 Data-driven tokenization

Tokenization is done by two components:

Token learner Learns a vocabulary from a given corpus (i.e., training). Token learner

Token segmenter Segments a given input into tokens based on a vocabulary (i.e., inference). Token segmenter

Byte-pair encoding (BPE) Based on the most frequent n-grams. Byte-pair encoding (BPE)

Token learner Given a training corpus C, BPE determines the vocabulary as follows:

1. Start with a vocabulary V containing all the 1-grams of C and an empty set of merge rules M.
2. While the desired size of the vocabulary has not been reached:
   a) Determine the pair of tokens $t_1 \in V$ and $t_2 \in V$ such that, among all the possible pairs, the n-gram $t_1 + t_2 = t_1 t_2$ obtained by merging them is the most frequent in the corpus C.
   b) Add $t_1 t_2$ to V and the merge rule $t_1 + t_2$ to M.

> **Example.** Given the following corpus:
>
> | Occurrences | Tokens |
> |---|---|
> | 5 | `l o w $` |
> | 2 | `l o w e r $` |
> | 6 | `n e w e s t $` |
> | 6 | `w i d e s t $` |
>
> The initial vocabulary is: $V = \{\texttt{\$}, \texttt{l}, \texttt{o}, \texttt{w}, \texttt{e}, \texttt{r}, \texttt{n}, \texttt{w}, \texttt{s}, \texttt{t}, \texttt{i}, \texttt{d}\}$.
> At the first iteration, $\texttt{e} + \texttt{s} = \texttt{es}$ is the most frequent n-gram. Corpus and vocabulary are updated as:

Occurrences	Tokens
5	`l o w $`
2	`l o w e r $`
6	`n e w es t $`
6	`w i d es t $`

$$V = \{\texttt{\$}, \texttt{l}, \texttt{o}, \texttt{w}, \texttt{e}, \texttt{r}, \texttt{n}, \texttt{w}, \texttt{s}, \texttt{t}, \texttt{i}, \texttt{d}\} \cup \{\texttt{es}\}$$

At the second iteration, $\texttt{es} + \texttt{t} = \texttt{est}$ is the most frequent n-gram:

Occurrences	Tokens
5	`l o w $`
2	`l o w e r $`
6	`n e w est $`
6	`w i d est $`

$$V = \{\texttt{\$}, \texttt{l}, \texttt{o}, \texttt{w}, \texttt{e}, \texttt{r}, \texttt{n}, \texttt{w}, \texttt{s}, \texttt{t}, \texttt{i}, \texttt{d}, \texttt{es}\} \cup \{\texttt{est}\}$$

And so on...

Token segmenter Given the vocabulary V and the merge rules M, the BPE segmenter does the following:

1. Split the input into 1-grams.
2. Iteratively scan the input and do the following:
   a) Apply a merge rule if possible.
   b) If no merge rules can be applied, lookup the (sub)word in the vocabulary. Tokens out-of-vocabulary are marked with a special unknown token `[UNK]`.

WordPiece Similar to BPE with the addition of merge rules ranking and a special leading/tailing set of characters (usually `##`) to identify subwords (e.g., `new##`, `##est` are possible tokens). WordPiece

Unigram Starts with a big vocabulary and remove tokens following a loss function. Unigram

1.3 Normalization

Normalization Convert tokens into a standard form. Normalization

Example. `U.S.A.` and `USA` should be encoded using the same index.

Case folding Map every token to upper/lower case. Case folding

Remark. Depending on the task, casing might be important (e.g., `US` vs `us`).

Lemmatization Reduce inflections and variant forms to their base form. Lemmatization

Example. $\{\texttt{am}, \texttt{are}, \texttt{is}\} \mapsto \texttt{be}$

Remark. Accurate lemmatization requires complete morphological parsing.

Stemming Reduce terms to their stem. Stemming

Remark. Stemming is a simpler approach to lemmatization.

Porter stemmer Simple stemmer based on cascading rewrite rules.

> **Example.** `ational` $\mapsto$ `ate`, `ing` $\mapsto \varepsilon$, `sses` $\mapsto$ `ss`.

1.4 Edit distance

Minimum edit distance Minimum number of edit operations (insertions, deletions, and substitutions) needed to transform a string into another one.

Minimum edit distance

> **Remark.** Dynamic programming can be used to efficiently determine the minimum edit distance.

Levenshtein distance Edit distance where:

Levenshtein distance

- Insertions cost 1;
- Deletions cost 1;
- Substitutions cost 2.

> **Example.** The Levenshtein distance between `intention` and `execution` is 8.

```
I  N  T  E  *  N  T  I  O  N
|  |  |  |  |  |  |  |  |  |
*  E  X  E  C  U  T  I  O  N
−  ±  ±     +  ±
1  2  2     1  2
```