Lab Class NLP:III

By June 09, 2025, solutions for the following exercises have to be submitted: 1, 2, 3, and 4.

Exercise 1 : Text Preprocessing

One of the first steps in developing a text-based model is to preprocess the text data. This step is crucial as it can significantly impact the effectiveness of the model. However, not all preprocessing steps are beneficial for every task. For example, lower-casing might be problematic for Named Entity Recognition (NER), where capitalization helps to distinguish between different meanings of words: `Bush` vs. `bush`; `Apple` vs `apple`.

For each of the following text preprocessing steps, give an example of a situation or a task where performing that step would likely reduce the effectiveness of a model trained on the preprocessed data and briefly explain why that might happen.

(a) Stemming

(b) Lemmatization

(c) Stopword removal

(d) Removing URLs

(e) Removing all non-alphanumeric characters

Exercise 2 : Regular Expressions

(a) Write regular expressions that match the following patterns:

  (a1) A sequence of alphabetic strings, i.e., strings that contain only letters, e.g., `abc`, `DEfg`, etc.

  (a2) All strings that contain the substring `abc` followed by any number of characters, e.g., `abc`, `abcd`, `abc123`, `abc!@#`, etc.

  (a3) All strings that start with exactly one uppercase letter, followed by any number of characters except uppercase letters, and end with either a period, exclamation mark, or question mark, e.g., `Abcd!`, `Efg123.`, but not `HIj12k34?`.

  (a4) All strings that start at the beginning of the line with an integer and that end at the end of the line with an alphabetic character.

(b) Consider the following regular expressions.

(b1) Which of the following inputs will be matched by the regular expression

```
^.*[aeiou].*(ist|ism|ant|er|or)$
```

- ☐ teacher
- ☐ bigger
- ☐ important
- ☐ inferior
- ☐ artist
- ☐ ant

Describe the inputs accepted by the regex in your own words.

(b2) Which of the following inputs will be matched by the regular expression

```
^.*[aeiou]ing$
```

- ☐ singing
- ☐ playing
- ☐ walking
- ☐ seeing
- ☐ driving

Describe the inputs accepted by the regex in your own words.

(b3) The following regular expression should match *email addresses*:

```
^[a-zA-Z0-9_.-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$
```

Which of the following strings will be matched by the regular expression?

- ☐ max.mustermann@uni-weimar.de
- ☐ max.mustermann@uni-weimar
- ☐ max.mustermann@uni.weimar.de
- ☐ max.mustermann@uni.weimar.
- ☐ Max.Mustermann@Uni-Weimar.DE

(c) Consider the following algorithm *Tokenize*, which uses regular expressions to segment a given sequence d into tokens:

Tokenize(d)

```
alwayssep="[?!()\"\/\\|]"]
clitic="(?:'|:|-|'s|'d|'m|'ll|'re|'ve|n't)"
```

1. Apply `s/$alwayssep/ $& /g` to d.
2. Apply `s/(\D),/\1 , /g` and `s/,(\D)/ , \1/g` to d.
3. Apply `s/\s'/$& /g` and `s/(\W)'/\1 '/g` to d.
4. Apply `s/$clitic\s/ $&/g` and `s/($clitic)(\W)/ \1 \2/g` to d.
5. Split d by whitespace (`/\s+/`) to obtain a list of tokens T.

Manually execute the tokenization algorithm *Tokenize* on the given sequence d_1 by showing the state of d_1 after each step. Fill in the table below:

- Show the exact string after each step 1-5.

d_1	Thomas' note said, that: "7:30am isn't great :("
1.	
2.	
3.	
4.	
5.	

Exercise 3 : Regular Expressions III

(a) Come up with regular expressions that match the following word patterns. Your expression must:

- Match all provided examples.
- Exclude incorrect examples (e.g., using `.*` will not be accepted).

(a1) Match one or more question marks. Examples: `?`, `???`, `?????`

(a2) Match words containing exactly 5 characters. Examples: `hello`, `world`, `input`

(a3) Match words containing exactly 4 characters, starting with `lo`. Examples: `love`, `long`, `loop`

(a4) Match words containing at least 10 characters. Examples: `independence`, `expression`, `traditional`

(a5) Match words made of lowercase letters only. Examples: `apple`, `tree`, `lowercase`

(a6) Match years from the 18th century (1700–1799). Examples: `1700`, `1725`, `1799`

(b) Which of the following inputs will be matched by the regular expression

```
^#[A-Fa-f0-9]{6}$
```

Which of the following strings will be matched?

- ☐ #FF5733
- ☐ FF5733
- ☐ #GH1234
- ☐ #a1b2c3
- ☐ #12345

Describe the inputs accepted by the regex in your own words.

Exercise 4 : Regular Expressions

Write a regular expression that matches an ISO datetime format, which consists of a date and time expressed in coordinated universal time (UTC), with the format `YYYY-MM-DDTHH:MM:SSZ`. The `T` separates the date and time, the `Z` indicates UTC time zone, and the time is in 24-hour format.

Your regular expression should:

- Match valid ISO datetime strings, but not match invalid ones
- Allow for leading zeros in the month and day fields (e.g. `2023-05-07T11:23:45Z` is a valid match)
- Only match UTC time zone (`Z`), not other time zone abbreviations (e.g. `2019-01-01T00:00:00-0800` is not a valid match)
- Be able to extract UTC datetime strings contained within arbitrary text (e.g., `"2023-05-07T11:23:45Z"` is a valid match but `12023-05-07T11:23:45Z` or `A2023-05-07T11:23:45Z` are not).

Exercise 5 : Writing a tokenizer

Write a function

```
def tokenize(text:  str) -> list[str]
```

in Python that takes in a text and outputs a list of tokens. You are only allowed to use standard Python imports (especially `re` for regular expressions). Your tokenizer should represent the following as a single token:

- Dates (e.g., `01/01/70`, `01-01-1970`, `01.01.1970`, `01.  Jan.`)
- Numbers (e.g., `0.5`, `0,5`, `100 000`, `100'000`)
- "Hashtags" (e.g., `#Hashtag`)
- Abbreviations (e.g., `U.S.A.`)

Come up with at least one additional type of token and implement it in your tokenizer. Submit your solution as a `.py` file.