Text Normalization

MMAI 5400 – lecture 2
Fall 2024

Today's lecture

NLP tasks

- What, why & when

Text normalization

- Tokenization
- Word normalization
- Stop word removal

NLP tasks

What is NLP?

- A collection of tasks concerned with machine processing of natural language.

Tasks are what you have to do to solve NLP-related business problems.

It’s often your job to recognize what task to apply given a business problem. The problem itself might not mention the task.

You need to know what the NLP tasks are & in what situations they apply.

NLP tasks

- Not a complete list

Tasks

- *Data acquisition* (lecture 1)
- *Text normalization* (lecture 2)
  - Preprocessing in NLP
- *Language modelling* (lecture 3)
  - Many uses
- *Text classification* (lecture 4)
  - Classification for texts

Tasks

- *Topic modelling* (lecture 5)
  - Clustering for texts
- *Parsing* (lecture 6)
  - Extracting grammatical structure & relationships within sentences.
- *Named entity recognition* (lecture 8)
  - Locate & classify named entities in text into categories like person name, locations, etc.

Tasks continued

- Still not a complete list

Tasks

- *Summarization*
- *Question answering* (lecture 12)
  - Chatbots
- *Machine translation*
- *Text generation* (lecture 11)
- *Aspect-based sentiment analysis* (lecture 10)
  - Parsing + sentiment analysis
- *Text comparison*

What do you need to know about a task?

- What type business problems can it be applied to?
- What are the inputs (i.e. what type of data)?
- What is the output?
- What other tasks are involved?
- How good is state of the art?
    - What model/algorithms are involved?
- What resources are required during deployment (estimate)?
    - Compute, data privacy, etc.
- What resources are required for development (estimate)?
    - Knowledge, time, compute, data, etc.

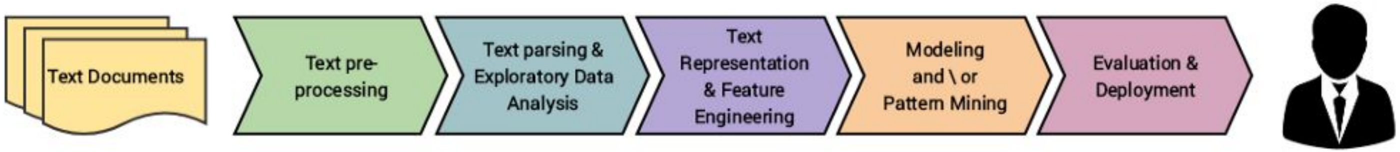
What can you do with good knowledge of tasks?

- Your job

1. Identify the NLP task(s) that a particular business problem depends on.
2. Given the NLP task(s), identify their requisites.
3. Decide whether it's feasible or not.
4. If feasible, execute the tasks.

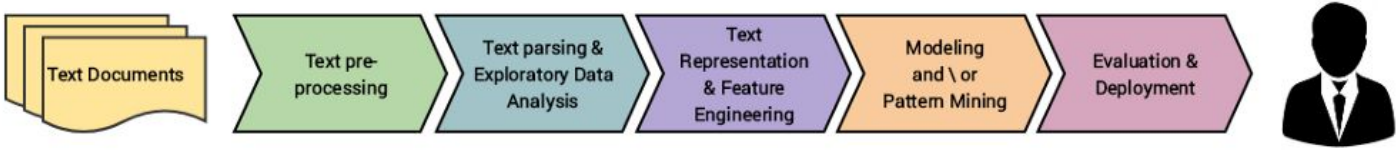
Text normalization

Common NLP workflow



1. **Text normalization (pre-processing)**
   - Tokenization
   - Stemming & lemmatization
2. **Parsing (structuring)**
   - Named Entity Recognition (NER)
   - Grammatical function of words – PoS tagging
   - Syntactic role of elements – dependency parsing
3. **Representation & feature engineering**
   - Bag-of-words; embeddings
4. **Modelling/mining**
   - ML
5. **Evaluation**

Common NLP workflow



TODAY

1. **Text normalization (pre-processing)**
   - Tokenization
   - Stemming & lemmatization
2. **Parsing (structuring)**
   - Named Entity Recognition (NER)
   - Grammatical function of words – PoS tagging
   - Syntactic role of elements – dependency parsing
3. **Representation & feature engineering**
   - Bag-of-words; embeddings
4. **Modelling/mining**
   - ML
5. **Evaluation**

LATER

What is text normalization?

- NLP speak for preprocessing

Case normalization: E.g., turn all characters in the text to lowercase.

Tokenization: Convert running text (a long string) into a sequence of tokens.

Part-of-speech tagging (lecture 5): Assign a grammatical category (such as noun, verb, adjective, etc.) to each word in a sentence.

Stop word removal: Remove uninformative words (i.e. stop words) such as articles (the, an) and conjunctions (and, but).

Word normalization: Convert words to a standardized format or choosing a single, normal form for words with multiple variations.

- *Stemming*
- *Lemmatization*

Text normalization

1. **Tokenizing**
2. **Normalizing word formats**
    - All words to lowercase
    - Plurals to singulars
    - Stemming
    - Lemmatizing
3. **Removing stop words**

3. **Segmenting (parsing) sentences**
    - Part-of-speech tagging
    - Dependency parsing

Trade-off

Losing information in return for better generalization

Tokenization

Tokenizing

The goal is to segment running text into units (i.e. tokens) that are **useful** for downstream tasks (e.g. dependency parsing & machine learning).

Text can be tokenized at 3 levels:

1. Word tokenization
2. Sub-word tokenization - e.g. byte-pair encoding
3. Character - each character is an individual token

Word tokenization

The goal is to segment running text into words (tokens) that are **useful** for downstream tasks (e.g. dependency parsing & machine learning)

Challenge: *periods & commas*. The are often informative, but their meaning/role varies

- *"A period can mark the end of a sentence."* vs *"But a period can also mark the decimal point in 3.1416."*
- Tokenizing the period in the 1st sentence makes sense, but in the 2nd sentence it would result in breaking up the number 3.1416 into the tokens "3", "." & "1416" (not very useful for downstream tasks).

Word tokenization

Challenge: *space*. Space generally indicate word boundaries, but not always.

- E.g. "*The new York University library.*" vs "*The New York University library.*"
- In the 1st sentence "*new*" & "*York*" should be separate tokens, but in the second "*New York*" should be a single token.
- To disambiguate this, Named Entity Recognition (NER, class 8) often have to be part of tokenization

See *Speech & Language Processing*, by Dan Jurafsky & James H. Martin (2019), pages 15 & 16 (chapter 2) for more examples.

Word tokenization

Tokenization has to be ***fast***, it is just a pre-processing step.

Common practice is to use an implementation based on regular expressions.

Question: What happens when tokens occur in the test set that didn't appear in the training set?

Character tokenization only make sense for sequence models (e.g. LSTMs & Transformers).

Character-level tokenization

Defining tokens as words can result in many unknown words, i.e. words that occur very rarely. This is a problem for ML.

A way to address this problem is character-level tokenization

- Simple: each character is a token
- E.g.: "My name is not Zumbi." is broken up into "M", "y", " ", "n", "a", "m", "e", " ", "i", "s", " ", "n", "o", "t", " ", "Z", "u", "m", "b", "i", "."

Character-level tokenization

Defining tokens as words can result in many unknown words, i.e. words that occur very rarely. This is a problem for ML.

Another way to address this problem is sub-word tokenization

- The goal is to find the largest chunks (tokens) without any unknown/very rare tokens.
- Large chunks/tokens are important to decrease the number of combinations of tokens (limit combinatorial explosion).
- The result is often that common words are tokenized to the entire words, while rare words are tokenized into sub-words/part-of-words.
- Common algorithms are Byte-Pair Encoding & WordPiece (described in section 2.4.3 of *Speech & Language Processing*).

Sub-word tokens

Sub-word tokens can be space-delimited words (like `bicycle`), word combinations (like `York University`), & parts of words (e.g. the morphemes* `-est` or `-er`).

Byte-pair-encoding (BPE)

Used for GPT-[/d]

- Vocabulary size: 50 000

Wordpiece

Used for BERT

- Vocabulary size: ≈30 000

* a morpheme is the smallest meaning-bearing unit of a language. E.g. the word `unlikeliest` has the morphemes `un-`, `likely`, & `-est`. See pages 17 & 21 in Speech and Language Processing by Jurafsky & Martin.

Question: What would properties should a reference corpus have?

BPE algorithm

The BPE algorithm was originally proposed as a compression algorithm.

But, it is also very well suited as a tokenizer for language models.

The idea is to divide up words into a sequence of sub-word units that appear frequently in the *reference corpus*.

The reference corpus is the corpus used to "train" the BPE tokenizer.

BPE algorithm

1. Define a vocabulary size.
2. Get a big reference corpus.
3. Tokenize the text into words.
4. Add an end token ("_") to each word.
5. Split words into characters.
6. Initial vocabulary will be the unique characters including _. These are called symbols.
7. Merge the most frequent symbol pair.
8. The merged symbol pair becomes a new symbol & is added to the vocabulary.
9. Repeat 7 & 8 until the vocabulary size has reached the size determined in step 1.

Question: Does this example start from step 1 (previous slide)?
Question: At which step does the example start?

BPE example*

- 1st iteration

From a corpus with 18 words.

Count	Dictionary	Vocabulary
5	`l o w _`	_, d, e, i, l, n, o, r, s, t, w
2	`l o w e s t _`	
6	`n e w e r _`	
3	`w i d e r _`	
2	`n e w _`	

* example from pages 18-19 in Speech and Language Processing by Jurafsky & Martin.

BPE example*

- 2nd iteration

Count	Dictionary	Vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_
2	l o w e s t _	
6	n e w e r_	
3	w i d e r_	
2	n e w _	

* example from pages 18-19 in Speech and Language Processing by Jurafsky & Martin.

BPE example*

- 3rd iteration

Count	Dictionary	Vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_
2	l o w e s t _	
6	n e w er_	
3	w i d er_	
2	n e w _	

* example from pages 18-19 in Speech and Language Processing by Jurafsky & Martin.

BPE example*

- 4th iteration

Count	Dictionary	Vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew
2	l o w e s t _	
6	n ew er_	
3	w i d er_	
2	n ew _	

* example from pages 18-19 in Speech and Language Processing by Jurafsky & Martin.

BPE example*

- 10th iteration

Vocabulary

`_`, `d`, `e`, `i`, `l`, `n`, `o`, `r`, `s`, `t`, `w`, `r_`, `er_`, `ew`, `new`, `lo`, `low`, `newer_`, `low_`

WordPiece

- Similar to BPE, but based on LM likelihood & start of word token instead of end of word token (i.e. `_token` vs `token_`)

* example from pages 18-19 in Speech and Language Processing by Jurafsky & Martin.

Levels of tokenization

Word-level tokens

- Problems with rare or unseen words, i.e. out-of-vocabulary words (OOV)

Character-level tokens

- No problems with OOV
- Too fine-grained & missing important information
- Require sequence models (e.g. doesn’t work with BoW)

Sub-word tokens

- No problems with OOV
- Not too fine-grained

Word normalization

Question: What is printed?

Counting words

What do we want to capture when we count words?

The meaning of words

- A trivial example:
    - In the sentences "*I live in Canada.*" & "*I live in canada.*", "Canada" & "canada" refer to the same entity (the country Canada) & should thus be counted as the ***same*** word.
    - Try this: `print('Canada' == 'canada')`.

How can we address this?

- ***Text normalization***!!

Word normalization

Put words/tokens in a standard format. I.e. choosing a single normal form for words with multiple forms like Canada & Ca.

This can be valuable, despite losing some information. For example, for information retrieval we want the same information whether we search with “Canada” or “Ca”.

There are 2 ways to deal with this: **stemming** & **lemmatization**

Question: Of the 2 examples below, which is/are stemmed vs lemmatized?

Normalization

- stemming & lemmatizing

The goal of both stemming & lemmatization is to reduce a word’s inflectional & derivational (sometimes) forms to a common base form.

Examples:

1. My dog**’s** fur **is** dark → My dog fur **be** dark
2. The girl**s**’ bicycl**es** are differ**ent** colour**s** → the girl ’ bicycl are differ colour

Stemming

Chopping off ends of words

Quick & dirty

Produces non-english words

The Porter Stemmer is a commonly used algorithm

Lemmatization

Like stemming, but “better”

Slower

Using a vocabulary & morphological analysis to get the base/dictionary form of a word (the *lemma*)

Stemming

Stemming

- Often defined as a crude *heuristic* procedure where the ends of words are chopped off in the hope of getting to the words base form.
- Frequently it includes the removal of derivational affixes.

Sometimes this works well:

- Stemming "interesting" & "interested" return "interest" (using the NLTK Porter Stemmer).

Sometimes this doesn't work well:

- Stemming the word "see" & "saw" return "see" & "saw" (using the NLTK Porter Stemmer).

There are multiple different stemmers, relying on different rules for chopping off the ends of words, & thus they produce different results.

Lemmatization

Lemmatization

- Obtaining a word's base form by using a vocabulary & morphological analysis, often excluding only inflectional endings & return the base or dictionary form of a word – the lemma.

Lemmatization algorithms often require the word's part-of-speech as an additional input.

Example:

My dog**'s** fur **is** dark → My dog fur **be** dark

Stop word removal

Stop words

Common words that don’t add (enough) information to the NLP task.

Remove them before further processing.

Example

```
“I was walking under the power lines & I slipped & hit my knee.
There was no visible injury but it hurt the next day.”
```

Stop words

Common words that don’t add (enough) information to the NLP task

Remove them before further processing

Example

“I ~~was~~ walking under ~~the~~ power lines ~~&~~ I slipped ~~&~~ hit my knee. ~~There was~~ no visible injury ~~but it~~ hurt ~~the~~ next day.”

Text processing
- terminology

Word – a single distinct meaningful element of speech or writing; usually separated by spaces

- Depending on the task, words can include period (“.”), comma (“,”), question marks (“?”) & similar

Token or term – individual components of a text; can be a character, sequence of characters, word or a sequence of words

Sentence – a sequence of words that conveys a meaning

Document – an individual piece of text organized into sentences (& paragraphs)

Corpus – the text (or speech) dataset; a collection of documents (or speech recordings)

- E.g. the Brown corpus: a million-word collection of samples from 500 English texts from different genres (e.g. newspaper, fiction, non-fiction, & academic), assembled at Brown University in 1960s.

Text processing
- terminology

Lemma – the canonical/dictionary or citation form of a set of words having the same stem, the same major part-of-speech, & the same word sense

- E.g. *run* is the lemma among *run*, *runs*, *ran*, & *running*

Wordform – the full inflected or derived form of the word

- E.g. *run*, *runs*, *ran*, & *running* are all wordforms

Lemmatization

Stemming

Text normalization exercise

Open `MMAI5400_class02_TextNormalization.ipynb`

Python Debugging?