Text preprocessing

X. Hilaire

ESIEE Paris, IT department
x.hilaire@esiee.fr

AIC-5102B Natural Language Processing

Outline

1. Data collection and preprocessing
2. Tokenization
3. Tokenization using regular expressions
4. Byte pair encoding
5. Unigram language model

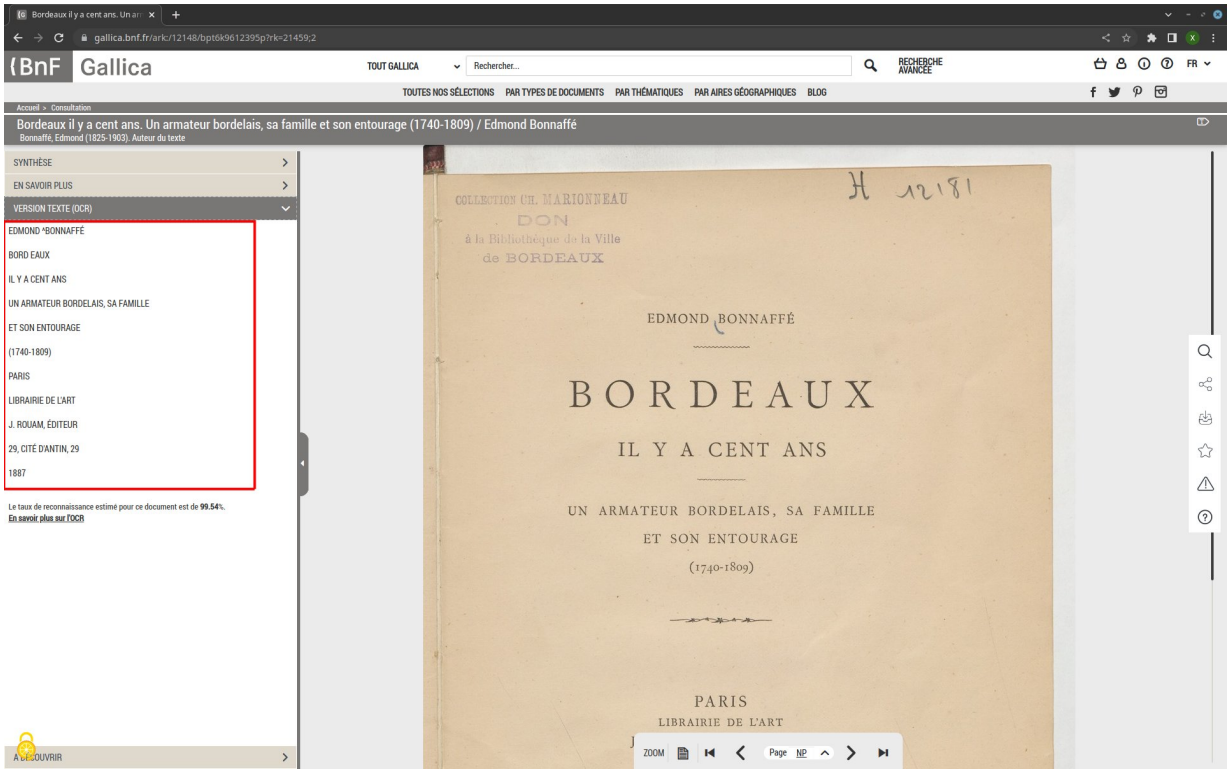
Outline

1. Data collection and preprocessing
2. Tokenization
3. Tokenization using regular expressions
4. Byte pair encoding
5. Unigram language model

Collecting text data

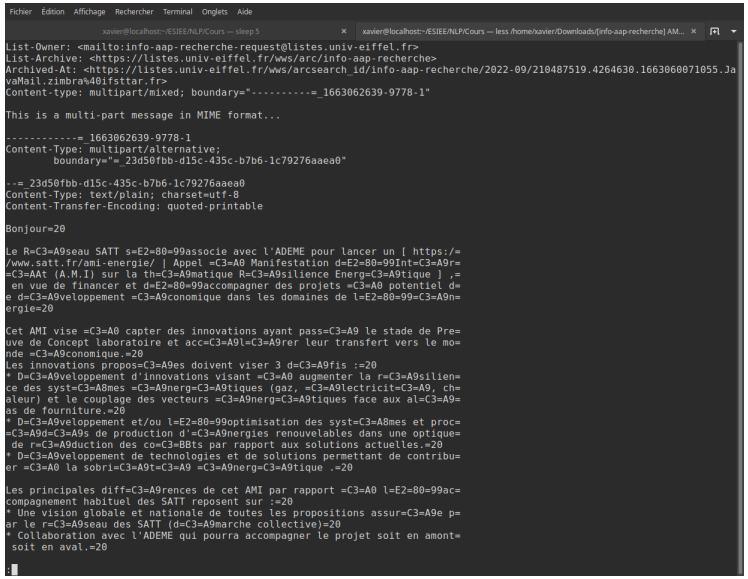
Text data can be gathered from various sources:

- Fully OCR'ized corpora. Example: Gallica (Bibliothèque nationale de France), ≈ 9 millions documents – `https://gallica.bnf.fr`



- OCR is not perfect : "EDMOND ^BONNAFFÉ" "BORD EAUX"...
- Yet $>$ 99.54% in this example

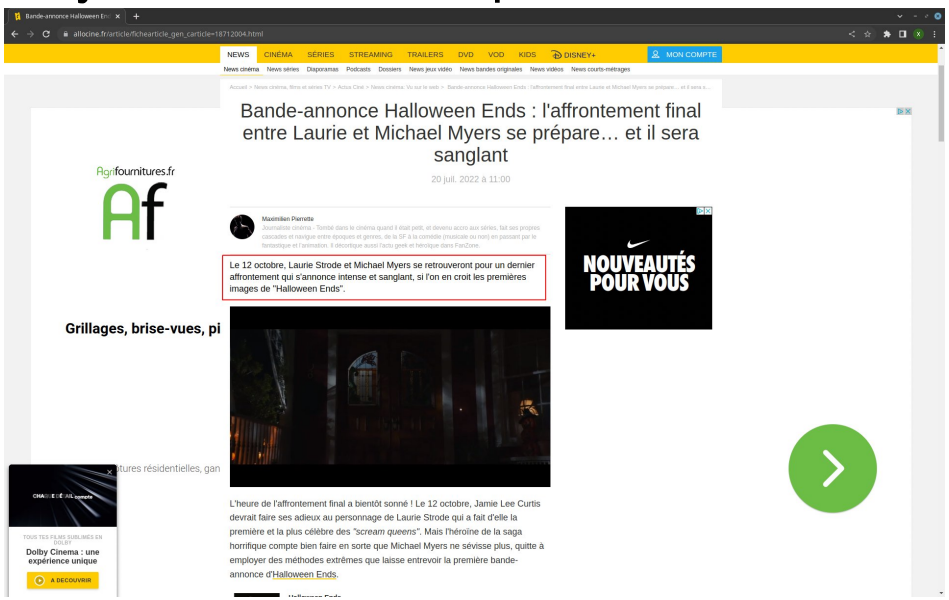
Collecting text data

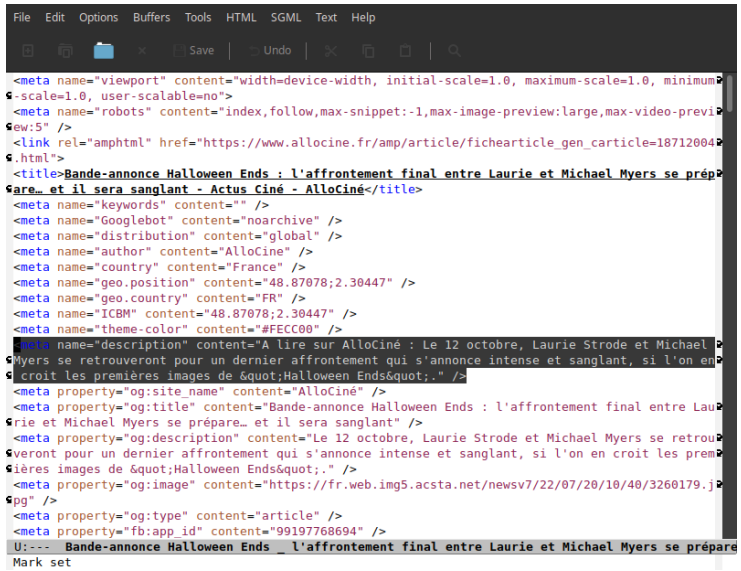


- Email
- MIME attachments must be decoded
- May consist of plain text / MS-Word / XLS / PDF / XML... documents
- All are convertible to plain text

Collecting text data

- Any website. Example: AlloCiné

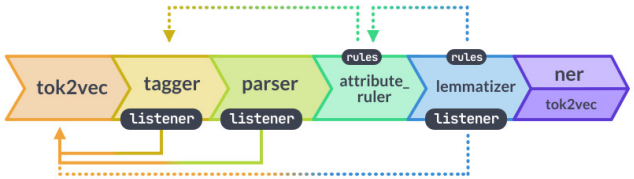




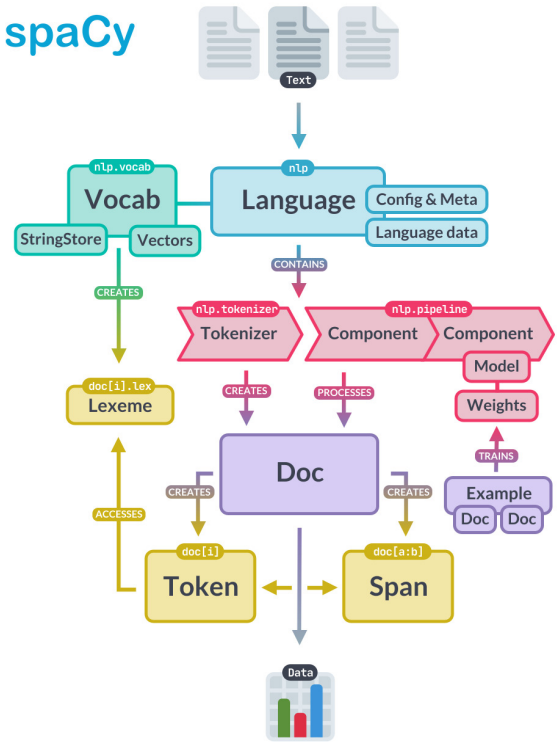
- We have to get rid of HTML tags
- But useful text might reside within an HTML tag ! (`<meta content=...>` in the example)

NLP pipelines (examples)

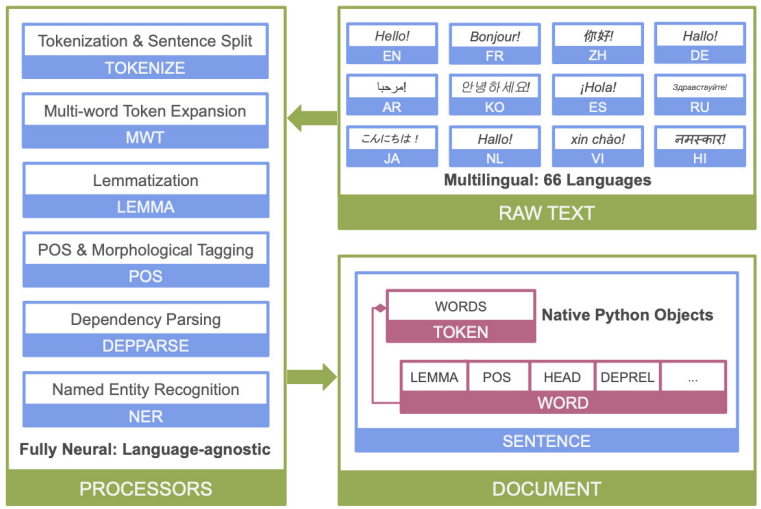
spaCy – `https://spacy.io/api`



API:



Stanza – `https://stanfordnlp.github.io/stanza`



Data preprocessing

- Unless very comfortable assumptions (case of fully OCRized corpora above), data must be preprocessed to be exploitable.
- Preprocessing usually involves 3 steps:
  1. Tokenization / normalization :
     - segments data into "tokens", or lexical units. Those include subwords, words, or sentences.
     - transforms certain sequences into others, e.g. *Cooooool* $\mapsto$ *cool*; *HELLO* $\mapsto$ *hello*; *March 3, 2021* $\mapsto$ *3 march 2021*
     - throws away everything else (data assumed non usable)
  2. Removal of stop words : removes all words which bear no or little meaning. Examples: *the*, *of* , *a*, *an*, *at*, *on*, *in*, *which*.
  3. Lemmatization and stemming :
     - lemmatization replaces conjugated forms of verbs, adjectives, or substantives by their lemmas, or glossed words. For instance, *giving* $\mapsto$ *give*, *drew* $\mapsto$ *draw*. The lemma is a word of the source language.
     - Stemming cuts the inflected part of words to keep only their radicals, or root. For instance, *giving*, *gives*, *give* $\mapsto$ *giv*. The root is not necessarily a word of the source language.

Outline

1. Data collection and preprocessing
2. Tokenization
3. Tokenization using regular expressions
4. Byte pair encoding
5. Unigram language model

Tokenization

Definition

Tokenization is to split a chunk of input data into useful, atomic items called lexical units, while discarding everything else.

- Tokenization can be heard at word, subword, or sentence level
- At sentence level, a (bad) idea is to segment according to punctuation alone:
  - semi-colons (:) or half-stops usually do not raise significant problems
  - commas (,) are already more ambiguous
  - and dots (.) are extremely ambiguous
- See next slide

Sentence tokenization

Example[1]:

```
TORONTO | North American markets ended the trading day in
the red, with Canada's main stock index down more than 150
points and U.S. stock markets also lower.

The S&P/TSX composite index was down 165.98 points at
19,560.16, driven by losses in the energy sector as the
price of oil fell.In New York, the Dow Jones industrial
average was down 173.27 points at 30,961.82. The S&P 500
index was down 44.66 points at 3,901.35, while the Nasdaq
composite was down 167.32 points at 11,552.36.
```

- *U.S.*, *S&P*, *165.98*, *19,560.16*
- Missing space inserted "...of oil fell.In New York..."
- Many others are possible : *Ph.D*, *J.R. Ewing*, `C.Civ.` ("code civil", in French), hundreds of abbreviations.

[1] Source https://ca.finance.yahoo.com/news/p-tsx-composite-down-nearly-153431890.html

Word tokenization

Word tokens may be (non limiting list):

- All common words: articles, verbs, adjectives, adverbs, etc.
- Proper nouns (Paris, London, Jean-Pierre, Varenne-Saint-Hilaire)
- Dates : 23/03/2022 (FR), 03.23.2022 (US), 03 March 2022 (US), ...
- Time : 13:47:20, 01:47:20 pm (UK), 13h 47mn 20s (FR)
- Numbers : 110,40 (FR), 110.40 (US), 110245300.40, 110,245,300.40 (US), 110.245.300,40 (FR)
- Amounts of money : 110,40€; 110,40€; 110.40£; but $110.40
- URLs; email addresses; LAN addresses : https://www.esiee.fr, www.esiee.fr ; x.hilaire@esiee.fr ; 147.215.50.88/32
- Hashtags, or authors handles : #DirectAN, @BrunoLeMaire

Word tokenization

- In theory, tokens depend on language, writer, document, reason for writing the document (context)
- In practice, tokenization must be fast ☞tokenization rules should be universal, and should not bother about meaning of tokens. Example: 10.01.22 might refer to
  - 10th of January 2022
  - or subnetwork with IP 10.01.22.0/24

  but xx.yy.zz should always be accepted
- Rules for tokenization may be :
  - specified formally ☞one solution is to use regular expressions (regex)
  - inferred, or learned from data ☞BPE, machine learning

Word tokenization

- NLTK[2] provides a tokenization package, and two tokenization methods:
  - splitting based on words alone (with or without punctuation)
  - splitting based on regular expressions

[2] Natural Language Toolkit – `www.nltk.org`

Outline

1. Data collection and preprocessing
2. Tokenization
3. Tokenization using regular expressions
4. Byte pair encoding
5. Unigram language model

A refresher on regular expressions

- Let Σ be a set of symbols, called alphabet. A language over Σ is a subset of the powerset of Σ
- A language over Σ is rational if it is either:
  - The empty language $\emptyset$
  - A language of the form $\{a\}$, where $a \in \Sigma$
  - A language of the form $L^*, L + R$, or $L.R$

  and nothing else.
- $L + R$ is the language whose words are the union of those of L and R.
- $L.R$ is the language whose words are obtained by concatenating all possible pairs of words from L and R
- $L^n = L.L...L$ is the n^{th} power language of L
- $L^* = +_{n \geq 0} L^n$ is the union of all power languages of L. The star * is called the Kleene star.

A refresher on regular expressions

Example

- Let $L = \{a, b\}$ and $R = \{b, c\}$
- Then $L + R = \{a, b, c\}$
- $L.R = \{ab, ac, bb, bc\}$
- $L^2 = L.L = \{aa, ab, ba, bb\}$
- $L^* = \{a, b, c, aa, ab, ba, bb, aaa, baa, bba, aab, abb, bab, bbb, ...\}$

Examples

- The set of integer numbers is a rational language.
- The set of dates of the form $dd/mm/yyyy$ is a rational language.

Counter-examples

- The set of properly parenthesized algebraic expressions is not a regular language
- The set of palindroms over the Latin alphabet is not a regular language

A refresher on regular expressions

- A rational expression is a string that describes a rational language
- The + and . operators, exponentiation, and Kleene star in rational languages all have their equivalent rational expression

Regular expression	Rational language	Remarks
`(a|b|c|d)`	{a,b,c,d}	Language made of a,b,c,d
`(L|R)`	$L + R$	L and R are regular
`[abcd]`	{a,b,c,d}	same as above
`[a-d0-2t]`	{a,b,c,d,0,1,2,t}	
`[^a-ct]`	$\Sigma \setminus \{a, b, c, t\}$	any character except
`LR`	$L.R$	L and R are regular
`L{`n`}`	L^n	L is regular, $n > 0$
`L{`m, n`}`	$L^m + L^{m+1} + ... + L^n$	L is regular, $0 < m \leq n$
`L{`m`,}`	$L^m + L^{m+1} + L^{m+2} + ...$	L is regular, $m > 0$
`L*`	L^*	L is regular, Kleene star
`L?`	$\varepsilon + L$	ε = empty word
`L+`	$L + L^2 + L^3 + ...$	Equivalent to `L{1,}`

Word tokenization (cont'd)

Word-based tokenization (word_tokenize, without punctuation)

```
>> import nltk
>>> from nltk.tokenize import word_tokenize
>>> s='''The S&P 500 Volatility Index increased by 22.5% on
the same day, while the S&P/TSX Composite Index in Canada
lost more than 5% of its value in between 2:30 p.m. to 3:00
p.m.'''
>>> word_tokenize(s)
['The', 'S', '&', 'P', '500', 'Volatility', 'Index',
  'increased', 'by', '22.5', '%', 'on', 'the', 'same', 'day',
  ',', 'while', 'the', 'S', '&', 'P/TSX', 'Composite', 'Index',
  'in', 'Canada', 'lost', 'more', 'than', '5', '%', 'of', 'its',
  'value', 'in', 'between', '2:30', 'p.m.', 'to', '3:00', 'p.m',
  '.']
```

Word tokenization (cont'd)

Word-based tokenization (punctword_tokenize, with punctuation)

```
>>> from nltk.tokenize import wordpunct_tokenize
>>> wordpunct_tokenize(s)
['The', 'S', '&', 'P', '500', 'Volatility', 'Index', 'increased',
'by', '22', '.', '5', '%', 'on', 'the', 'same', 'day', ',',
'while', 'the', 'S', '&', 'P', '/', 'TSX', 'Composite', 'Index',
'in', 'Canada', 'lost', 'more', 'than', '5', '%', 'of', 'its',
'value', 'in', 'between', '2', ':', '30', 'p', '.', 'm', '.',
'to', '3', ':', '00', 'p', '.', 'm', '.']
>>>
```

Word tokenization (cont'd)

Regex-based tokenization (regex_tokenize)

```
>>> nltk.regexp_tokenize(s, r'''[0-9\.]+%?|[A-Z]+\&[A-Z]+|\w+''')
['The', 'S&P', '500', 'Volatility', 'Index', 'increased', 'by',
'22.5%', 'on', 'the', 'same', 'day', 'while', 'the', 'S&P', 'TSX',
'Composite', 'Index', 'in', 'Canada', 'lost', 'more', 'than', '5%',
'of', 'its', 'value', 'in', 'between', '2', '30', 'p', '.', 'm',
'.', 'to', '3', '00', 'p', '.', 'm', '.']
>>>
```

Exercise: give the regular expressions (extended form) that recognize:

1. Email addresses
2. Signed decimal numbers US style (e.g. 12.345, or -110245300.45 or -110,245,300.40)
3. Dates, European style (dd/mm/yyyy). You may assume February always has 28 days.

Word tokenization (cont'd)

Exercise (solution):

Subword tokenization

- Sometimes, it can wise to split some word into subwords to ease indexation and querying
- Example, in German: *Donaudampfschiffahrtselek-trizitätenhauptbetriebswerkbauunterbeamtengesellschaft*
- One word, 79 letters
- Meaning: *society of subordinate officials of the main maintenance building of the Danube steamboat electrical services*
- Preferred in split form: Donau $\rightarrow$ Danube, dampf $\rightarrow$ steam, schiffahrts $\rightarrow$ boat (+ interfix), elektrizitäten $\rightarrow$ electricities / electrical services, haupt $\rightarrow$ head/main, betriebs $\rightarrow$ maintenance, werkbau $\rightarrow$ factory building, unter $\rightarrow$ under, beamten $\rightarrow$ officials, gesellschaft $\rightarrow$ society
- If we don't split, and the query is "stemboat, electrical services, association", it is unlikely the related document be retrieved.

Deriving subwords from data

- It is common that affixes (prefixes, infixes or suffixes) existing words are mixed to create a new word :
  - *picture* + *element* $\rightarrow$ *pixel*
  - *involuntary* + *celibate* $\rightarrow$ *incel*
  -
- We can either input everything manually, or let an algorithm derive subwords from subsequences of words
- Popular algorithms to achieve this include:
  - Byte pair encoding (BPE)
  - Unigram language model
  - WordPiece
  - SentencePiece

Outline

1. Data collection and preprocessing
2. Tokenization
3. Tokenization using regular expressions
4. Byte pair encoding
5. Unigram language model

Byte pair encoding

- Byte pair encoding (BPE) is an old, yet still used tokenizer from Philip Gage, 1994. *P. Gage, "A new algorithm for data compression", The C Users Journal, Volume 12, Issue 2, Feb. 1994, pp 23–38*
- "The algorithm compresses data by finding the most frequently occurring pairs of adjacent bytes in the data and replacing all instances of the pair with a byte that was not in the original data. The algorithm repeats this process until no further compression is possible, either because there are no more frequently occurring pairs or there are no more unused bytes to represent pairs."
- High resemblance with Huffman encoding
- Adapted by Sennrich et al. in 2016 [6]

Byte pair encoding

- Consider corpus = { newer, low, lower, newest, widest }.
- Every word is thought as a sequence of characters.
- Step 1 :

corpus	vocabulary
n,e,w,e,r	n,e,w,r,l,o,s,t,i,d
l,o,w	
l,o,w,e,r	
n,e,w,e,s,t	
w,i,d,e,s,t	

☞Competing pairs : (w,e), score=3. No other competitor. Consider (w,e) $\mapsto$ we

- Step 2 :

corpus	vocabulary
n,e,<u>we</u>,r	<u>we</u>,n,e,w,r,l,o,s,t,i,d
l,o,w	
l,o,<u>we</u>,r	
n,e,<u>we</u>,s,t	
w,i,d,e,s,t	

☞Competing pairs : (e,we), (l,o), (n,e), (s,t), (we,r), score=2. Consider (e,we) $\mapsto$ ewe

Byte pair encoding

- Step 3 :

corpus	vocabulary
n,ewe,r l,o,w l,o,we,r n,ewe,s,t w,i,d,e,s,t	ewe,we,n,e,w,r,l,o,s,t,i,d

☞Competing pairs :

(l,o), (n,ewe), (s,t), score=2. Consider (l,o) $\mapsto$ lo

- Step 4 :

corpus	vocabulary
n,ewe,r lo,w lo,we,r n,ewe,s,t w,i,d,e,s,t	lo, ewe,we,n,e,w,r,s,t,i,d

☞Competing pairs :

(n,ewe), (s,t), score=2. Consider (n,e) $\mapsto$ newe

Byte pair encoding

- Step 5 :

corpus	vocabulary
newe,r lo,w lo,we,r newe,s,t w,i,d,e,s,t	lo, newe,we,e,w,r,s,t,i,d

☞Competing pairs :

(s, t), score=2. Consider (s,t) $\mapsto$ st

- Step 6 :

corpus	vocabulary
newe,r lo,w lo,we,r newe,st w,i,d,e, st	st, lo, newe,we,e,w,r,i,d

☞Competing pairs :

none of score ≥ 2. Stop (for the sake of example)

Byte pair encoding

Concluding remarks:

- The algorithm has achieved rather poor results here
- It did not detect `new`, `low`, `-est`, `-er`
- This is mainly because (w, e) was the dominating pair but not the most useful.
- As with all greedy algorithm, it is highly dependent on the order competing pairs are processed, and introduces confusion when they become many
- An heuristic in such a case is to prefer pairs which begin words (**prefix**) or end them (**postfix**)

Byte pair encoding

Python code by Sennrich et al. [6]

Algorithm 1 Learn BPE operations

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

Outline

1. Data collection and preprocessing
2. Tokenization
3. Tokenization using regular expressions
4. Byte pair encoding
5. Unigram language model

Unigram language model

- See Kudo, 2018 [2]
- Let $\mathcal{V}$ be a vocabulary.
- Probabilities $p(x)$ are unknown for every $x \in \mathcal{V}$, but assumed independent
- As a result, the probability of any sequence $\mathbf{x} = (x_1, ..., x_M)$ is just the product of all $p(x_i)$'s:

$$P(\mathbf{x}) = \prod_{i=1}^{M} p(x_i)$$

- If X is an input sequence, then the most probable segmentation $\mathbf{x}^*$ of X is then given by

$$\mathbf{x}^* = \arg_{\mathbf{x} \in \mathcal{S}(X)} \max P(\mathbf{x}) \tag{1}$$

where $\mathcal{S}(X)$ is the set of all possible segmentations.

Unigram language model

- A document D consists of sentences $X^{(1)}, X^{(2)}...$
- Given (1), the $p(x_i)$'s are those which maximize the likelihood

$$\mathcal{L} = \sum_{s=1}^{|D|} \log P(X^{(s)}) = \sum_{s=1}^{|D|} \log \left(\sum_{\mathbf{x} \in \mathcal{S}(X^{(s)})} P(\mathbf{x}) \right) \tag{2}$$

- They can be estimated using the EM algorithm, provided $\mathcal{V}$ is given
- Problem: in real life, $\mathcal{V}$ is not known too, which makes the problem intractable
- Kudo suggests the following steps:

Unigram language model

1. Choose an reasonably large enough seed vocabulary $\mathcal{V}$ from the corpus
2. Repeat the following until $\mathcal{V}$ reaches a desired size:
   1. Estimate the $p(x_i)$'s by maximizing (2)
   2. Compute $loss_i$ = reduction of $\mathcal{L}$ induced by removing sequence x_i from $\mathcal{V}$
   3. Sort the x_i's by $loss_i$, and keep the top η % as new $\mathcal{V}$

- Kudo suggests to choose the initial $\mathcal{V}$ as a subset of most frequently found subsequences by BPE
- Highly relies on this initial choice

Unigram language model

- Little improvement over BPE, according to the BLEU score[3] after translation from English to German

Model	BLEU
Word	23.12
Character (512 nodes)	22.62
Mixed Word/Character	24.17
BPE	24.53
Unigram w/o SR ($l = 1$)	24.50
Unigram w/ SR ($l = 64,\ \alpha = 0.1$)	25.04

Table 5: Comparison of different segmentation algorithms (WMT14 en→de)

- See full results here – `https://arxiv.org/abs/1804.10959`

[3] Bilingual evaluation understudy – see, e.g. `https://en.wikipedia.org/wiki/BLEU`

WordPiece

- Originally proposed by Schuster and Nakajima in 2012 [5], in the context of speech processing
- Used in Google's BERT
- Highly obscure, as described by only 4 enumerated items in [5]:

> tally the same metric that we use during decoding. The algorithm to find the automatically learned word inventory efficiently works in summary as follows:
>
> 1. Initialize the word unit inventory with the basic Unicode characters (Kanji, Hiragana, Katakana for Japanese, Hangul for Korean) and including all ASCII, ending up with about 22000 total for Japanese and 11000 for Korean.

WordPiece

2. Build a language model on the training data using the inventory from 1.
3. Generate a new word unit by combining two units out of the current word inventory to increment the word unit inventory by one. Choose the new word unit out of all possible ones that increases the likelihood on the training data the most when added to the model.
4. Goto 2 until a predefined limit of word units is reached or the likelihood increase falls below a certain threshold.

Modern tokenizers

- Though simple, using BPE for word tokenization is subject to criticism:
  - Proceeds greedily ☞ensures minimal-length encoding (MLE). But does MLE implies maximal relevancy ?
  - Highly dependent on corpus order ☞solution is not unique, and may destroy useful subwords – consider $(low, e) \mapsto lowe$ in the example
- Modern tokenizers include ELMo [3], BERT [1], and GPT [4] ☞all are highly contextual dependent, and involve LSTM. Not presented this year.

Stemming

- An old stemming algorithm is due to Martin Porter (1980)
- Works only on English language
- The Porter stemmer algorithm has 5 steps, and replaces each termination of words found at each step by others, according to some hard-coded rules. Excerpts:

```
Step 1  SSES -> SS                          caresses  ->  caress
        IES  -> I                           ponies    ->  poni
                                            ties      ->  ti
        SS   -> SS                          caress    ->  caress
        S    ->                             cats      ->  cat
Step 2  (m>0) ATIONAL ->  ATE       relational     ->  relate
        (m>0) TIONAL  ->  TION      conditional    ->  condition
                                    rational       ->  rational
        (m>0) ENCI    ->  ENCE      valenci        ->  valence
        (m>0) ANCI    ->  ANCE      hesitanci      ->  hesitance
        ...
```

Stemming

- Can be implemented very easily with UNIX commands like `sed`
- Many implementations available at `https://tartarus.org/martin/PorterStemmer/`
- A trainable


Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova.
BERT: pre-training of deep bidirectional transformers for language understanding.
In Jill Burstein, Christy Doran, and Thamar Solorio, editors, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 4171–4186. Association for Computational Linguistics, 2019.


Taku Kudo.
Subword regularization: Improving neural network translation models with multiple subword candidates.
In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 66–75, Melbourne, Australia, July 2018. Association for Computational Linguistics.


Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer.

Deep contextualized word representations, 2018.
cite arxiv:1802.05365Comment: NAACL 2018. Originally posted to openreview 27 Oct 2017. v2 updated for NAACL camera ready.


Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever.
Language models are unsupervised multitask learners.
2018.


Mike Schuster and Kaisuke Nakajima.
Japanese and korean voice search.
In 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 5149–5152, 2012.


Rico Sennrich, Barry Haddow, and Alexandra Birch.
Neural machine translation of rare words with subword units.
In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.