Text Analysis and Retrieval

2. Basics of Natural Language Processing

Assoc. Prof. Jan Šnajder

With contributions from
Assist. Prof. Goran Glavaš
Mladen Karan, PhD

University of Zagreb
Faculty of Electrical Engineering and Computing (FER)

Academic Year 2019/2020

v2.1

Motivation: NLP as preprocessing

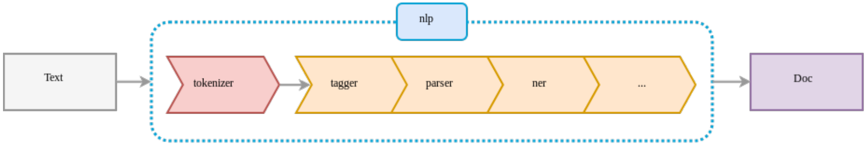
- Most text analysis subtasks addressed by natural language processing are useful for TAR
- For basic IR, you don't need much: **tokenization** and a bit of **morphology processing** suffices
- For full-blown semantic text analysis, you need a lot: proper **morphology, syntax, and semantic processing**
- There are many tools available for these tasks (unfortunately, in most cases the best tools available work only for English)

Outline

Learning outcomes 1

1. Describe the components of the basic NLP pipeline
2. Describe what POS tagging is and why we need it
3. Explain stemming and lemmatization, why we need it, and the difference between them
4. List the main NLP tools available

Typical NLP pipeline



Typical NLP pipeline

(1) Language detection
(2) Text cleanup (boilerplate removal / normalization / OCR-error correction, . . .)
(3) Sentence segmentation
(4) Tokenization
(5) Morphological processing: stemming
(6) POS tagging
(7) Morphological processing: lemmatization
(8) Syntactic processing: parsing

$\vdots$

Higher-level tasks (semantics, information extraction, . . .)

Basic NLP pipeline

(1) Language detection
(2) Text cleanup (boilerplate removal / normalization / OCR-error correction, ...)
(3) **Sentence segmentation**
(4) **Tokenization**
(5) **Morphological processing: stemming**
(6) **POS tagging**
(7) **Morphological processing: lemmatization**
(8) Syntactic processing: parsing

⋮

Higher-level tasks (semantics, information extraction, ...)

Sentence segmentation and tokenization

- **Sentence segmentation**: finding boundaries of sentences in text
  - Often done heuristically, using regular expressions
  - Best performance with supervised machine learning models
    - predict for each full stop whether it denotes the end of the sentence
- **Tokenization**: breaking a text up into tokens – words and other meaningful elements
  - tokens are words, punctuation marks, and special characters
  - rule-based (i.e., heuristic) vs. supervised approaches
- Elementary preprocessing tasks, hence supported by all major NLP libraries (e.g., StanfordCoreNLP, NLTK, OpenNLP)

Morphological processing

- The same word can appear in text in different **morphological variants**. Is this a problem?
- Most of the time, yes!

- **In IR:** The query house should match against document talking about houses and maybe housing (but probably not about housewifes)
- **In information extraction:** If money lounderies or money laundry appear in the text, we'd like to extract a keyphrase money laundering
- **Many other IR/TM tasks:** We simply want to count house, houses and housing as being the same thing
- **For syntax and semantics:**
  We need to know the grammatical features of a word: Ana voli Ivana is not the same as Anu voli Ivan

Basics of morphology

Morphology

Branch of linguistics concerned with the internal structure of words. Words are made up of morphemes (= smallest linguistic pieces with a grammatical function).

1. **Inflectional morphology**: creating word-forms that express grammatical features
   - fish → fishes, Haus → Häuser, skup → najskupljoj
2. **Derivational morphology**: creating new words from existing ones
   - fish → fishery, Haus → Gehäuse, voće → voćnjak
3. **Compounding**: combine two or more existing words
   - sky + scraper, Kraft + fahr + zeug, vatro + gasac

Quick test

Inflection, derivation, or compounding?

- EN: show → showed
- EN: big → bigger
- HR: novac → novčanik
- HR: kupiti → otkupljivanje
- EN: house → housing
- EN: run → runs
- DE: kaufen → verkaufen
- DE: kaufen → verkauft
- EN: house → housewife
- EN: tour → detoured

Morphological normalization

- Transform each word to its **normalized form** (whatever it is)
- Two approaches:
  - **Stemming** – quick and dirty
  - **Lemmatization** – linguistically proper way of normalization

Stemming

- Reduction of word-forms to **stems**
  - adjustments $\rightarrow$ adjust
  - defensible $\rightarrow$ defens
  - revivals $\rightarrow$ reviv
- Typically by **suffix stripping** plus some extra steps and checks
- Pros: simple and efficient
- Cons:
  - prone to **overstemming** and **understemming** errors
  - difficult to design for morphologically complex languages
  - imprecise (don't differentiate between inflection and derivation)

Porter stemmer

- Popular suffix-stripping stemmer
  - Initial algorithm designed for English
- Each word can be represented as $[C](VC)^m[V]$, where C is a sequence of consonants and V is a sequence of vowels
- Each word has a measure m:
  - $m = 0$ tr, ee, tree, by
  - $m = 1$ trouble, oats, trees, ivy
  - $m = 2$ troubles, troubles, private
- Suffix stripping rules: `(condition) S1 -> S2`
  - `(m > 1) EMENT ->`
  - `(m>0) ALIZE -> AL`
  - `(m>0) TIONAL -> TION`
  - `(m>1 and (*S or *T)) ION ->`

Porter stemmer

- A cascade of 5 suffix removal steps:
  - Step 1 deals with plurals and past participles
  - Step 2–4: derivation
  - Step 5: tidying up
- Porter stemmer occasionally overstems (university/universe) and understems
- Still it works fine in most cases and is very useful in IR applications
- Porter-like stemmers for many other languages exists as part of the **Snowball project**
  - http://snowball.tartarus.org/

Lemmatization

- Transformation of a word-form into a linguistically valid base form, called the **lemma** (the dictionary form)
  - nouns → singular nominative form
  - verbs → infinitive form
  - adjectives → singular, nominative, masculine, indefinitive, positive form
- A much more difficult task than stemming, especially for morphologically complex languages, for which you basically need:
  - a **morphological dictionary** that maps word-forms to lemmas
  - a **machine learning model**, trained on a large number of word-lemma pairs
- Example of a machine learning-based lemmatizer: CST lemmatizer
  - http://cst.dk/online/lemmatiser/uk/

Parts-of-speech

- **Part of speech** is the grammatical category of a word
- Some parts of speech are universal across languages:
  - **Verbs** assert something about the subject of the sentence and express actions, events, or states of being
  - **Nouns** are words that we used to name a person, an animal, a place, a thing, or an abstract idea
  - **Adjectives** modify nouns and pronouns by describing, identifying, or quantifying them.
  - **Pronouns** replace nouns or another pronouns and are essentially used to make sentences less cumbersome and less repetitive
  - **Adverbs** modify a verb, an adjective, another adverb, a phrase, or a clause. An adverb indicates manner, time, place, cause, ...
  - **Prepositions**, **Conjunctions**, **Interjections** ...

POS tagging

- **POS tagging** (grammatical tagging, word-category disambiguation) is the process of marking up a word in a text as corresponding to a particular part of speech

POS-tagged text

A/DT Part-Of-Speech/NNP Tagger/NNP is/VBZ a/DT piece/NN of/IN software/NN that/WDT reads/VBZ text/NN in/IN some/DT language/NN and/CC assigns/VBZ parts/NNS of/IN speech/NN to/TO each/DT word/NN ,/, such/JJ as/IN noun/NN ,/, verb/NN ,/, adjective/NN ,/, etc./FW./.

- POS taggers assign tags from a finite predefined **tagset**
  - For English, the most commonly used tagset is **Penn Treebank POS tagset**
- State-of-the-art POS taggers are supervised machine learning models

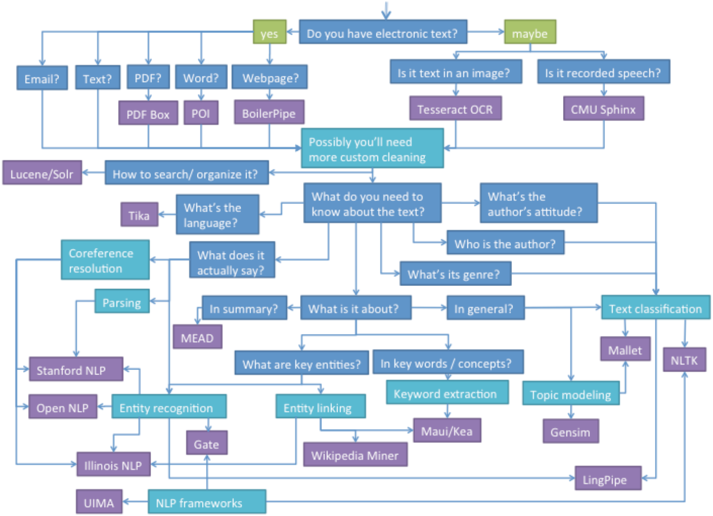
POS tagsets

- English: Penn Treebank POS tagset
  https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
- German: SSTS and relatives
  http://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/GermanTagsets.html
- Slavic languages: MULTEXT-East Morphosyntactic Specifications
  http://nl.ijs.si/ME/
  - Croatian (MULTEXT-East Version 4):
    http://nlp.ffzg.hr/data/tagging/msd-hr.html
  - Slovene (MULTEXT-East Version 5):
    https://www.sketchengine.co.uk/slovene-tagset-multext-east-v5/
  - ...

Available taggers

- For English
  - Stanford POS tagger
    http://nlp.stanford.edu/software/tagger.shtml
  - Center for Sprogteknologi (CST) tagger
    http://cst.dk/tools/index.php#output
  - CCG University of Illinois tagger
    http://cogcomp.cs.illinois.edu/demo/pos
  - Xerox POS tagger
    https://open.xerox.com/Services/fst-nlp-tools
  - ...
- For Croatian
  - ReLDI project
    https://reldi.spur.uzh.ch/resources-and-tools/

Tools jungle



http://entopix.com/so-you-need-to-understand-language-data-open-source-nlp-software-can-help.html

Apache UIMA

- UIMA = Unstructured Information Management Architecture
- Java framework for developing NLP pipelines
  http://uima.apache.org
- Standardized API between pipeline components
  - CAS = Common Analysis System
- Provides Eclipse plugins
- Wrappers for a wide variety of C++ and Java-based component libraries
- Focus is on integrating various NLP tools and scaling up to huge amounts of data
- Various NLP pipeline modules as UIMA components:
  https://uima.apache.org/external-resources.html

GATE

- Another Java framework for developing NLP pipelines
  https://gate.ac.uk
- Includes a number of rule-based NLP components
- Provides wrappers to a huge amount of NLP libraries (including UIMA, OpenNLP and Stanford parser)

Stanford CoreNLP

- http://nlp.stanford.edu/software/corenlp.shtml
- Java NLP library, integrates all Stanford NLP tools:
  - POS tagger
  - named entity recognizer
  - parser
  - coreference resolution system
  - sentiment analysis tools
- Offers very good out-of-the-box models for advanced NLP tasks
- Trained models are also available for languages other than English

NLTK

- Python library for NLP
  http://www.nltk.org/
- Accompanied by a book, available online and excellent for beginners!
  http://www.nltk.org/book
- Pros:
  - Good out-of-box models and algorithms for basic NLP tasks (tokenization, sentence segmentation, POS, WordNet, corpus analysis)
  - Training new models with user defined features is very easy

  Cons:
  - Lacks good pretrained models for more advanced tasks (e.g. parsing)
  - Can be slow compared to other tools

Other libraries

- **OpenNLP** – Java machine learning based text processing library
  https://opennlp.apache.org/
- **LingPipe** – Java toolkit for processing text
  http://alias-i.com/lingpipe/
- **Mallet** – Java MAchine Learning for LanguagE Toolkit
  http://mallet.cs.umass.edu/
- **spaCy** – "Industrial-Strength" NLP in Python ⇐ **recommended!**
  https://spacy.io/
- **AllenNLP** – NLP research library built on PyTorch and spacy
  https://allennlp.org/

⇒ Library comparison (one out of many):
  https://spacy.io/usage/facts-figures

- (There are also text processing plugins for some general purpose machine learning tools such as RapidMiner, R, KNIME, etc.)

Basic NLP pipeline

(1) Language detection
(2) Text cleanup (boilerplate removal / normalization / OCR-error correction, ...)
(3) **Sentence segmentation**
(4) **Tokenization**
(5) **Morphological processing: stemming**
(6) **POS tagging**
(7) **Morphological processing: lemmatization**
(8) Syntactic processing: parsing

$\vdots$

Higher-level tasks (semantics, extraction...)

Discussion points

1. Why do we need POS tagging?
2. Why does lemmatization typically come after POS tagging?
3. How detailed (fine-grained) should a POS tagset be?
4. What's the state of the art in POS tagging?
   ⇒ `https://github.com/sebastianruder/NLP-progress`
   ⇒ `https://paperswithcode.com/task/part-of-speech-tagging`
5. What if the text is noisy? (And what does that mean?)
6. Is an NLP pipeline robust to errors in early stages? How could this be remedied?

Learning outcomes 1 – CHECK!

1. Describe the components of the basic NLP pipeline
2. Describe what POS tagging is and why we need it
3. Explain stemming and lemmatization, why we need it, and the difference between them
4. List the main NLP tools available

Outline

Learning outcomes 2

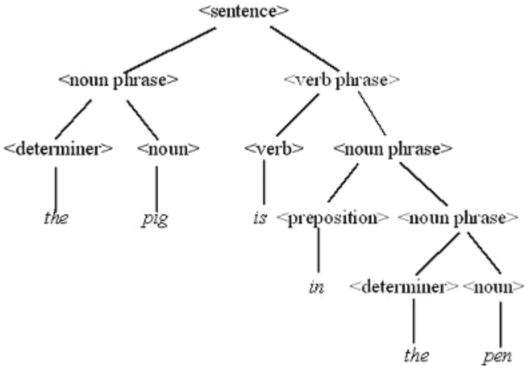
1. Describe what parsing is and why we need it
2. Differentiate between phrase-based and dependency-based parsing
3. Describe what chunking is and why we need it
4. List the main tools available for parsing/chunking

Grammars and parsers

- **Parsing** is the task of analyzing the grammatical structure of a sentence, which results in a syntax tree of the sentence
- Given a sequence of words, a parser forms units like subject, verb, object and determines the relations between them according to some grammar formalism
- Two types of parsers
  - **Constituency parsers/phrase structure tree (PST) parsers** – based on constituency/PS grammars
  - **Dependency parsers** – based on dependency grammars

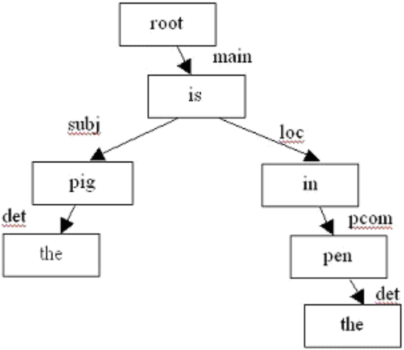
Parsers

- Constituency parser produces a tree that represents the syntactic structure of a sentence (i.e., a break down of the sentence)
  - Words appear only as leaves of the tree



Parsers

- Dependency parsing represents the structure of the sentence as the tree of syntactic dependencies between pairs of words
  - Each dependency relation has a governing word and a dependent word
  - Verb is the syntactic center of the clause, all other words directly or indirectly dependent on the verb



Parsers

- For English
  - Stanford parsers (both constituency and dependency)
    http://nlp.stanford.edu/software/lex-parser.shtml
  - Berkeley parser (constituency)
    https://code.google.com/p/berkeleyparser
  - Collins parser (constituency)
    http://people.csail.mit.edu/mcollins/PARSER.tar.gz
  - ...
- For Croatian
  - Maximum spanning tree (MST) parser (FFZG)
    http://nlp.ffzg.hr/resources/models/dependency-parsing
- For German:
  https://nlp.stanford.edu/software/
  http://pub.cl.uzh.ch/users/siclemat/lehre/ecl1/ud-de-hunpos-maltparser/html/
  https://github.com/rsennrich/ParZu

Universal dependencies (UD)

- Cross-linguistically consistent labels for multilingual parsing
  http://universaldependencies.org/#universal-dependencies-v2
- Universal POS Tags
  http://universaldependencies.org/u/pos/
- Universal Dependency Relations
  http://universaldependencies.org/u/dep/

- CoNLL format:
  https://universaldependencies.org/format.html

Universal dependencies (UD) parsers

- Developed by various research groups
  - Stanford UD parser (English)
    https://nlp.stanford.edu/software/stanford-dependencies.shtml
    demo: http://nlp.stanford.edu:8080/corenlp/
  - spaCy's dependency parser (English)
    https://spacy.io/api/dependencyparser
    very nice demo: https://explosion.ai/demos/displacy
  - ...
- Google's SyntaxNet
  https://opensource.google.com/projects/syntaxnet
  - Parsey McParseface (English)
    https://github.com/plowman/python-mcparseface
  - Parsey Universal (40 languages, including DE, HR, and SI)
    https://github.com/tensorflow/models/blob/master/research/syntaxnet/g3doc/universal.md

Shallow parsing (aka "chunking")

- **Shallow parsing** merely identifies the constituents (noun phrases, verbs phrases, prepositional phrases, etc.), but does not specify their internal structure nor their role in the sentence

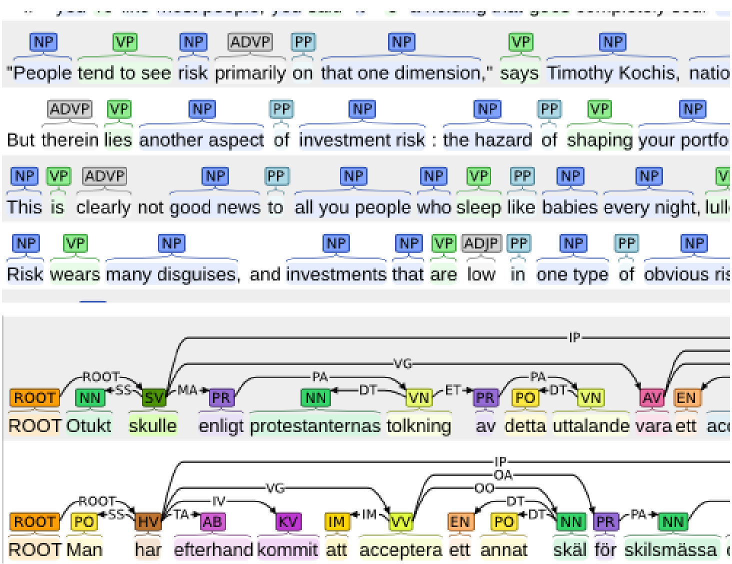
Shallow parsing example

[NP Jack and Jill] [VP went] [ADVP up] [NP the hill] [VP to fetch] [NP a pail] [PP of] [NP water].

- Some freely available shallow parsers:
  - CCG University of Illinois shallow parser
    `http://cogcomp.cs.illinois.edu/page/run_demo/ShallowParse`
  - Apache Open NLP shallow parser
    `http://opennlp.apache.org/index.html`
  - spaCy demo
    `http://textanalysisonline.com/spacy-noun-chunks-extraction`

Parsing

Chunking and parsing with Brat (http://brat.nlplab.org)



Discussion points

1. Why do we need parsing?
2. What could be a deficiency of using parsing in a pipeline?
3. Which one to prefer: constituency-based or dependency-based?
4. When to prefer chunking over parsing?
5. Can we go from a parse tree to chunks?
6. Is parsing equally difficult for all languages?
7. What's the state of the art in parsing?
   $\Rightarrow$ https://github.com/sebastianruder/NLP-progress
   $\Rightarrow$ https://paperswithcode.com/task/dependency-parsing
8. What if the text is noisy? (And what does that mean?)
9. Weird beasts: gardenpath sentences
   $\Rightarrow$ https://en.wikipedia.org/wiki/Garden-path_sentence
   $\Rightarrow$ https://www.apartmenttherapy.com/garden-sentences-262915

Learning outcomes 2 – CHECK!

1. Describe what parsing is and why we need it
2. Differentiate between phrase-based and dependency-based parsing
3. Describe what chunking is and why we need it
4. List the main tools available for parsing/chunking

Outline

Learning outcomes 3

1. Describe what a corpus is, why we need it, and name a few
2. Describe what a language model is and what it's used for
3. Write down the MLE probability for an N-gram language model
4. Differentiate between statistical and neural language models

Corpora

- **Text corpus** (plural: **corpora**): large and structured set of texts, used for **corpus linguistic analyses** and for the development of **natural language models** (primarily machine learning models)
- May be manually annotated:
  - e.g., POS-annotated or parsed corpora (**tree bank**)
  - possibly at different levels (multi-level annotated)
- Popular corpora (English):
  - Brown Corpus (1M words)
  - British National Corpus – BNC (100M words)
  - Wall Street Journal Corpus (30M words)
- Web as a Corpus (WaC): ukWaC, frWaC, deWaC, hrWaC
  - WaCky - The Web-As-Corpus Kool Yinitiative (http://wacky.sslmit.unibo.it)

Language modeling

- Probabilistic models of text, used for two purposes:
  1. determine the probability of the next word in a sequence
  2. determine the probability of a word sequence
- We'd like to compute the probability

$$P(w_1, w_2, \cdots, w_{n-1}, w_n) = P(w_1^n)$$

- This can be rewritten using the chain rule

$$\begin{aligned} P(w_1^n) =& P(w_1)P(w_2|w_1)P(w_3|w_1^2)\cdots P(w_n|w_1^{n-1}) \\ =& \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \end{aligned}$$

- All we need now is to estimate these probabilities. . .

Language modeling

- Naive solution: **maximum likelihood estimates (MSE)** from corpus

$$P(w_k|w_1^{k-1}) = \frac{C(w_1^k)}{C(w_1^{k-1})}$$

  where $C(\cdot)$ is the number of occurrences in the corpus
- This would fail because of **sparsity**: even short sequences of 5–6 words would barely ever appear in a corpus, no matter how large
- Solution: **approximate** the conditional by considering only N preceding words

$$\begin{aligned} P(w_k|w_1^{k-1}) \approx & P(w_k|w_{k-N+1}^{k-1}) \\ P(w_1^n) = & \prod_{k=1}^{n} P(w_k|w_{k-N+1}^{k-1}) \end{aligned}$$

Language modeling

- MLE:

$$P(w_k|w_{k-N+1}^{k-1}) = \frac{C(w_{k-N+1}^{k})}{C(w_{k-N+1}^{k-1})}$$

$\Rightarrow$ easier to estimate (less sparse) – but see next slide!

- $N = 2$ is a **bigram LM**, $N = 3$ is a **trigram LM**, etc.

Language model MLE

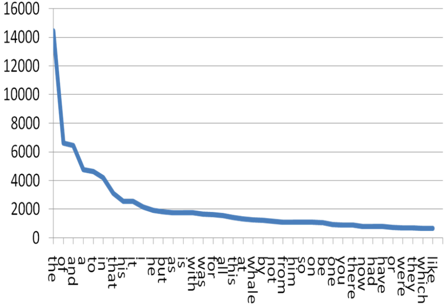
I saw a white fluffy...

- Bigram model ($N = 2$):
  $P(\text{rabbit}|\text{I saw a white fluffy}) \approx \frac{C(\text{fluffy rabbit})}{C(\text{fluffy})}$
- Trigram model ($N = 3$):
  $P(\text{rabbit}|\text{I saw a white fluffy}) \approx \frac{C(\text{white fluffy rabbit})}{C(\text{white fluffy})}$

- Increasing N increases the accuracy, but also memory usage!

A problem with MLE: Zipf's law

- Zipf's law (Zipf, 1949) states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table
- Example: sorted word counts in Herman Melville's "Moby Dick"



- **Happax legomena** account for $\sim 50\%$ of the words in corpus

Smoothing

- Due to Zipf's law, some word combinations will never occur in the corpus, zeroing the whole joint probability
- To further reduce the sparsity problem one can use smoothing:
  - Add one
  - Witten-Bell
  - Good-Turing
  - Kneser-Ney
- Or combining models of different order in various ways:
  - Backoff
  - Deleted interpolation

Neural language models (NLMs)

- Dan Jurafsky (2018). **Neural Networks and Neural Language Models**. (SLP draft chapter)
  https://web.stanford.edu/~jurafsky/slp3/7.pdf
- Yoshua Bengio et al. (2003). **A neural probabilistic language model**. Journal of machine learning research.
  http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

Discussion points

1. How to evaluate language models?
2. What's a disadvantage of SLM?
3. What's the state of the art in LM?

   ⇒ https://github.com/sebastianruder/NLP-progress

   ⇒ https://paperswithcode.com/task/language-modelling
4. Which one works better: SLM or NLM?
5. Can NLM be useful for tasks other than for predicting the next word or sentence probability?

Learning outcomes 3 – CHECK!

1. Describe what a corpus is, why we need it, and name a few
2. Describe what a language model is and what it's used for
3. Write down the MLE probability for an N-gram language model
4. Differentiate between statistical and neural language models