Introduction to Natural Language Processing

Part IX: Practical Issues

Henning Wachsmuth

https://ai.uni-hannover.de

Practical Issues: Learning Objectives

Concepts

- NLP processes and algorithm pipelines
- Available libraries and frameworks
- Issues related to effectiveness and efficiency

Methods

- Joint inference and pipeline extensions
- General effectiveness tweaks
- Pipeline efficiency optimization
- Parallelization

Outline of the Course

Introduction

Introduction

Going to the real world

- How to develop an NLP approach for a real application?
- How to build up an NLP application?
- What issues to take care of?

From single tasks to processes

- Often, pipelines of NLP algorithms realize complex analysis and/or synthesis processes with multiple (sub-) tasks.
- NLP algorithms for specific tasks can be reused in different processes.
- Frameworks exist to control the processes.

Main issues in NLP

- Low effectiveness, due to data or approach limitations
- Low efficiency, due to high run-time or memory consumption
- Low robustness, due to domain-specific development

  Robustness is not covered here, but in *Statistical NLP*.

Introduction

Development of an NLP Approach (Recap)

Input (typical)

- Task. An NLP task to be tackled
- Text corpus. A corpus, split into training, validation, and test set

A typical development process

1. Analyze on training set how to best tackle the task.
2. Develop approach based on some technique that tackles the task.
3. Evaluate the effectiveness of the approach on the validation set.
4. Repeat steps 1–3 until effectiveness cannot be improved anymore.
5. Evaluate the effectiveness of the final approach on the test set.

Output

- Approach. An NLP approach that serves as a method for the given task
- Results. Empirical effectiveness results of the approach

NLP Processes

NLP algorithm

- A single NLP algorithm usually realizes a method that infers one type of information from text—or generates one type of text

  Some also deal with a few related types at the same time.

  A sentence splitter outputs sentences

  A language model outputs one text

Why NLP *processes?*

- Many algorithms require as input the output of other methods, which in turn depend on further methods, and so forth.

  An entity recognizer may need part-of-speech tagging, which needs tokenization, ...

- Even a single type of output may require several methods.
- Most real-world NLP tasks aim at combinations of different types, such as those from information extraction.
- Due to the interdepencies, the standard approach to realize a process is in the form of an *algorithm pipeline*.

NLP Processes

Example: Information Extraction (Recap)

Information extraction

- The extraction of entities and their attributes, relations between entities, and events the entities participate in.
- Input. Unstructured natural language texts
- Output. Structured information that can, e.g., be stored in databases

Example task: Extraction of companies' founding dates

Time entity (2014), **Organization entity** (Google):

“ 2014 ad revenues of Google are going to reach

Reference (search company), **Time entity** ('98):

$20B . The search company was founded in ’98 .

Founded relation

Reference (Its), **Time entity** (in 2004):

Its IPO followed in 2004 . [...] “

Output: Founded("Google", 1998)

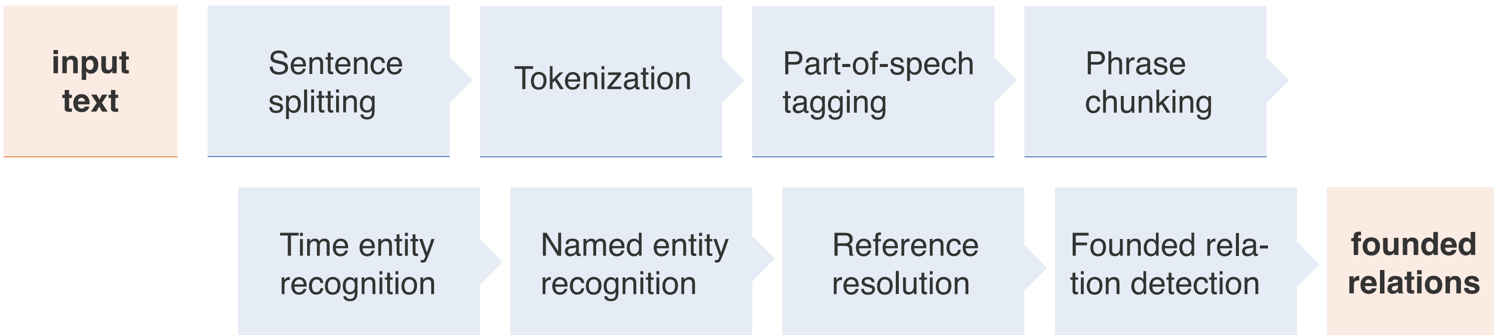
NLP Processes

Algorithm Pipelines

Algorithm pipeline

- A set of algorithms along with a schedule that defines the order of algorithm application.
- Each algorithm takes as input a text and the output of all preceding algorithms, and it produces further output.

Example pipeline for companies' founding dates



Pipeline scheduling

- The input requirements of each algorithm need to be fulfilled.
- Some algorithms are independent, i.e., they have no defined ordering.

NLP Processes

Algorithm Libraries

Problem?

- Tens of algorithms may be needed in an NLP approach.
- Implementing all of them from scratch would take forever.

Solution: Algorithm libraries

- Usually, only (or mainly) those algorithms are developed newly that infer the desired output information types in a given task.
- Other algorithms are taken from available algorithm libraries.
  This includes one's own algorithms from previous NLP approaches.
- The decomposition of a process into several steps is a main advantage of the pipeline approach in this regard.

Selected libraries

- Python. Stanford NLP, NLTK, spaCy, Huggingface, PyTorch, polyglot
- Java. Stanford CoreNLP, OpenNLP, mate-tools, TT4j

NLP Processes

NLP Frameworks

Problem?

- The data and control flow may be complex in an NLP approach.
- Implementing it from scratch is error-prone and time-intensive.

Solution: NLP frameworks

- Frameworks that define a standardized way of realizing NLP processes.
- Algorithms need to match a specific interface.
- Data and control flow are handled automatically (few lines of code).
- Known frameworks include Apache UIMA, GATE, and sklearn.

Example: Apache UIMA (for Java but with Python integration)

- Each algorithm implements a `process` method.
- XMI files specify input and output annotation types of algorithms.
- A pipeline is simply defined as a list of such XMI files.
- The framework calls `process` for each algorithm in a pipeline.

Effectiveness Issues

Effectiveness Issues

Reasons for limited effectiveness

- Ambiguity of natural language

  "Death penalty — why not?" → Stance on death penalty?

- Missing context and world knowledge

  "I hope Biden will keep his attitude towards capital punishment." → And here?

- Process-related reasons: Lack of training data, domain transfer, error accumulation (see next slide)

Perfect effectiveness?

- Noisy texts, errors in test data, subjective tasks, etc. prevent this.

  "i have mixed feelings about the death penalty." → Con stance?

- Only trivial tasks can generally be solved perfectly.

  "Capital punishment KILLS INNOCENT people." → # Capitalized tokens?

Effectiveness Issues

Process-related Reasons for Limited Effectiveness

Lack of training data

- Training data may often not suffice to make a given approach effective.
- If more data cannot be acquired, one may resort to simpler techniques.

  How to choose the right technique is discussed in *Statistical NLP*.

Domain transfer of an approach

- Approaches may fail on data very different from the training data.
- Ways out include heterogeneous training data and domain adaptation.

  How to deal with domain dependency is also discussed in *Statistical NLP*.

Error accumulation

- Errors propagate through an algorithm pipeline, since the output of one algorithm serves as input to subsequent ones.
- In standard pipelines, algorithms cannot fix errors of predecessors.
- Even when each algorithm works well, overall effectiveness may be low.

Effectiveness Issues

Strategies to Counter Error Accumulation

Joint inference algorithms

- Infer multiple information types simultaneously, in order to find the optimal solution over all types.

  In neural contexts, a related idea is called *multi-task learning.*
- Knowledge from each task can be exploited for the others.

  > Named entity recognition. Avoid confusion between different entity types.
  >
  > Argument mining. Segment and classify argument units in one step.
- This increases run-time notably and limits reusability.

Pipeline extensions

- Iterative pipelines. Repeat pipeline execution and use the output of later algorithms to improve the output of earlier ones.
- Probabilistic pipelines. Optimize a probability model based on different possible outputs and/or confidence values of each algorithm.
- Both require modifications of algorithms and notably reduce efficiency.

Effectiveness Issues

Practical Effectiveness Tweaks

Exploiting domain knowledge

- Rule of thumb. The narrower the domain, the higher the effectiveness
- Encoding domain-specific knowledge is important in practice.
- In-domain training is often a must for high effectiveness.

Combining statistics and rules

- Real-world NLP applications mostly combine statistical learning with hand-crafted rules.
- Rules are derived from a manual review of uncertain and difficult cases.

Scaling up

- At large scale, precision can be preferred over recall, assuming that the information sought for appears multiple times.
- A smart use of redundancy increases confidence.

"In 1998, he founded Google."	"Google exists since '98."	"Google, estd. 1998."

Efficiency Issues

Efficiency Issues

Reasons for limited efficiency

- NLP pipelines often include several time-intensive algorithms.
- Large amounts of data may need to be processed, possibly repeatedly.
- Much information may be stored during processing.

Ways to improve memory efficiency

- Scaling up is the natural solution to higher memory needs.
- Also, debugging (and minimizing) what information is stored may help.

Memory efficiency is often *not* the main problem.

Ways to improve run-time efficiency

- Indexing of relevant information
- Resort to simpler NLP algorithms
- Filtering and scheduling in pipelines
- Parallelization of NLP processes

Details on all of them below.

Efficiency Issues

Potential Memory Efficiency Issues

Memory consumption in NLP

- Permanent and temporal storage of input texts and output information
- Storage of algorithms and models during execution

Storage of inputs and outputs

- Single input texts are usually small in NLP
- Output information is negligible compared to input.
- The main problem may be the permanent storage of full text corpora.
  Some take only a few MB's, but large-scale corpora have hundreds of GB's or more.

Storage of algorithms

- Memory consumption may add up in longer text analysis pipelines.
- Machine learning brings up further challenges due to huge models.
- In both cases, powerful machines are needed — and/or parallelization.

Efficiency Issues

Indexing and Simpler Algorithms

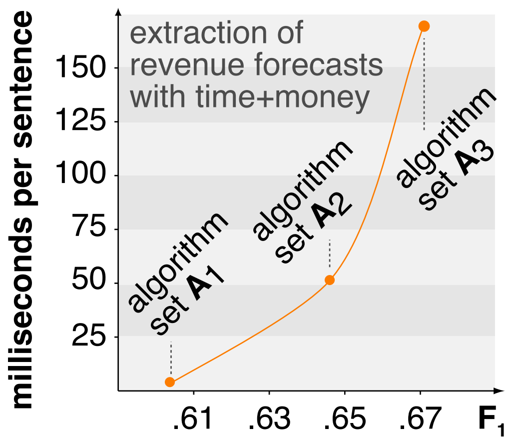
Indexing of relevant information

- In applications such as web search, the same information may have to be obtained multiple times from a text.
- By storing and indexing information beforehand, the need for *ad-hoc* NLP can be avoided.
- Naturally, this is restricted to anticipated information needs.
- Also, it implies a trade-off between run-time and memory efficiency.

Simpler algorithms

- A natural way to improve run-time is to use simpler but faster algorithms.
- Large efficiency gains possible.
- At large scale, high effectiveness is possible via redundancy and precision focus.

  See effectiveness issues above.



Efficiency Issues

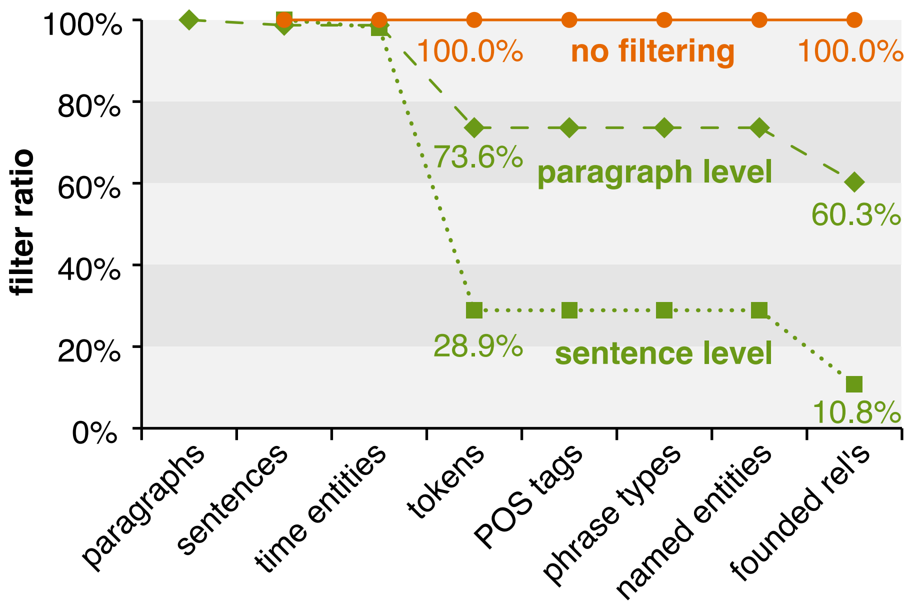
Filtering in Pipelines

Filtering relevant portions of text

- Standard pipelines apply each algorithm to the whole input text.
- For a given NLP task, not all portions of a text are relevant.
- After each step, irrelevant portions can be filtered out.
- The size of the portions trades efficiency for effectiveness.

Example: Extraction of founding dates

- Data. CoNLL'03 test set with 231 news articles
- Results. Tokenization on less than 30%; relation extraction only on every 9th sentence



Efficiency Issues

Pipeline Scheduling

Optimal scheduling of pipelines

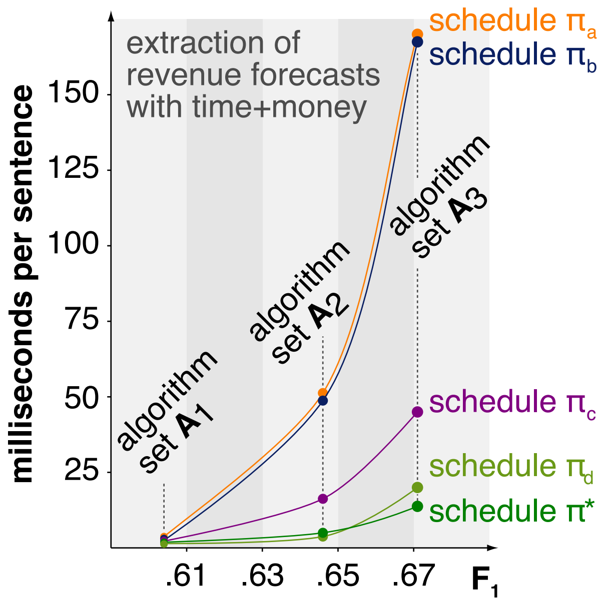
- With filtering, the schedule of a pipeline's algorithms affects efficiency.
- Schedule optimization is a dynamic programming problem based on the run-times and “filter rates” of the algorithms.

Intuition

- Filter out many portions of text early.
- Schedule expensive algorithms late.

Effects

- Efficiency may be improved by more than an order of magnitude.
- If filtering matches the level on which the algorithms operate, effectiveness is maintained.



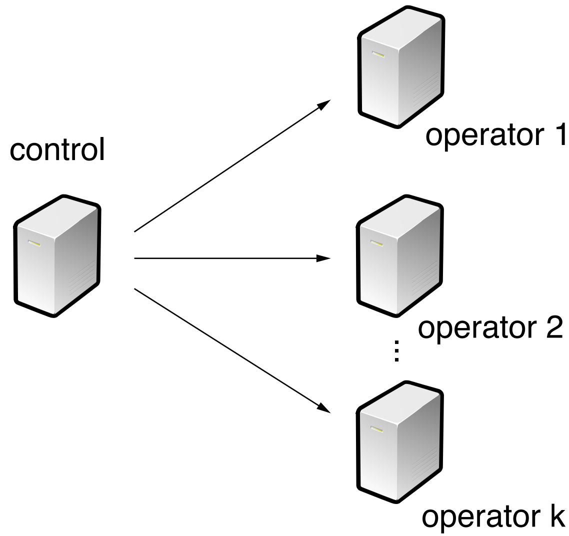
Efficiency Issues
Parallelization

Text analysis entails "natural" parallelization

- Input texts are usually analyzed in isolation, allowing their distribution.

  Here, we focus on basic scenarios and homogeneous architectures.

Basic parallelization scenario



- One control machine, many operators
- Control sends input to operators
- Operators process input and produce output
- Control aggregates output

Homogeneous parallel system architecture

- All machines comparable in terms of speed etc.
- No specialized hardware

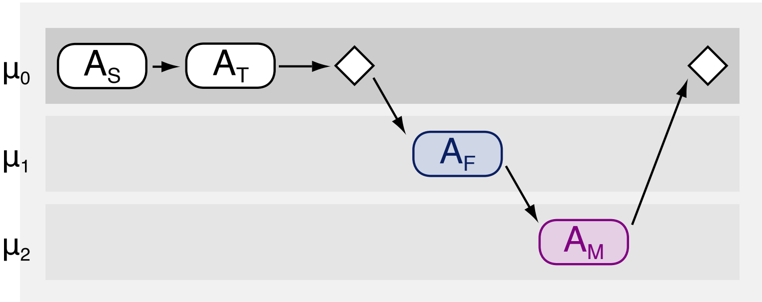
  Heterogenous architectures would require more tailored optimizations.

Efficiency Issues

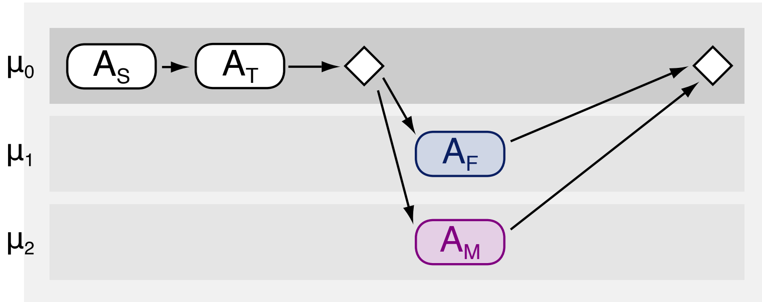
Four Approaches to Parallelizing NLP Processes

parallelization of NLP algorithms

(a) Process pipelining

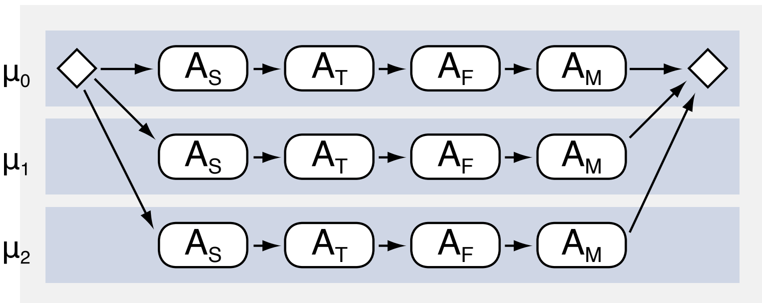


(b) Process parallelization

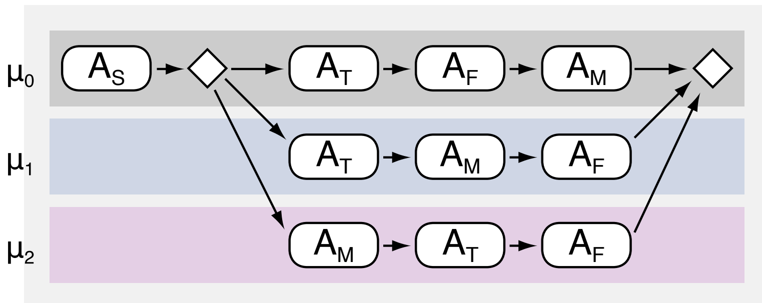


parallelization of NLP pipelines

(c) Pipeline duplication



(d) Schedule parallelization



(machines μ_0, μ_1, μ_2 and NLP algorithms A_S, A_T, A_F, A_M — A_F and A_M independent)

Efficiency Issues

Discussion of the Parallelization Approaches

Process pipelining

- Pro. Low memory consumption, lower execution time
- Con. Not fault-tolerant, high network overhead, machine idle times

Process parallelization

- Pro. Low memory consumption, possibly lower execution time
- Con. Not fault-tolerant, network overhead, high machine idle times

Pipeline duplication

- Pro. Very fault-tolerant, no idle times, much lower execution time
- Con. Full memory consumption on every operator

Schedule parallelization

- Pro. Fault-tolerant, few idle times, lower memory consumption, much lower execution time
- Con. Some network overhead, more complex process control

Conclusion

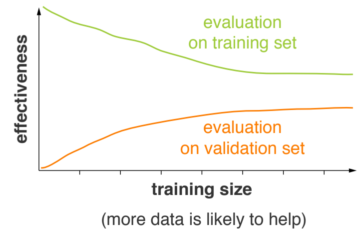
Summary

Practical issues

- NLP processes are often complex in practice.
- Algorithm libraries and frameworks help.
- Obtaining high effectiveness *and* efficiency challenging

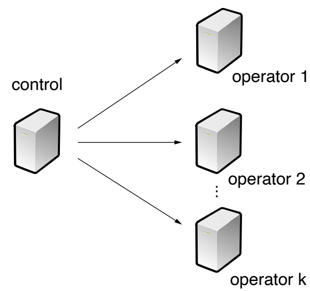
Effectiveness issues

- Error accumulation in the process must be faced.
- Available data size governs algorithm choices.
- Ambiguity and lack of context remain the main issues.



Efficiency issues

- Space efficiency may simply require much hardware.
- Run-time efficiency of NLP processes can be optimized.
- NLP can be parallelized very well.



References

Some content and examples taken from

- Daniel Jurafsky and Christopher D. Manning (2016). Natural Language Processing. Lecture slides from the Stanford Coursera course. `https://web.stanford.edu/~jurafsky/NLPCourseraSlides.html`.
- Henning Wachsmuth (2015): Text Analysis Pipelines — Towards Ad-hoc Large-scale Text Mining. LNCS 9383, Springer.