Colloquium

12.09.2013

Implementation of ~~Introduction~~ to Modified Kneser-Ney Smoothing on Top of Generalized Language Models for Next Word Prediction

Martin Körner

People and Knowledge Networks

Content

- Introduction
- Language Models
- Generalized Language Models
- Smoothing
- Implementation
- Future Work
- Summary

Content

- **Introduction**
- Language Models
- Generalized Language Models
- Smoothing
- Implementation
- Future Work
- Summary

Introduction: Motivation

- Next word prediction: What is the next word a user will type?

- Use cases for next word prediction:

  - Augmentative and Alternative Communication (AAC)

  - Small keyboards (Smartphones)

- How do we predict words?

1. Rationalist approach
   - Manually encoding information about language
   - “Toy” problems only

2. Empiricist approach
   - Statistical, pattern recognition, and machine learning methods applied on corpora
   - Result: Language models

Content

- Introduction
- **Language Models**
- Generalized Language Models
- Smoothing
- Implementation
- Future Work
- Summary

Language Models in General

- Language model: How likely is a sentence s?
  → Probability distribution: $P(s)$

- Calculate $P(s)$ by multiplying conditional probabilities
  - Example:

$$P(\text{If you're going to San Francisco , be sure } \dots) =$$
$$P(\text{you're} \mid \text{If}) * P(\text{going} \mid \text{If you're}) *$$
$$P(\text{to} \mid \text{If you're going}) * P(\text{San} \mid \text{If you're going to}) *$$
$$P(\text{Francisco} \mid \text{If you're going to San}) * \cdots$$

Next Word Prediction

- Instead of $P(s)$ only one conditional probability:

$$P(\text{Next Word}|\ \text{Current Sentence})$$

→Empiricist approach fails

Conditional Probabilities Simplified

- Markov assumption [JM80]:
  - Only the last $n-1$ words are relevant for a prediction:

$$P(\text{Next Word} \mid \text{Last } n-1 \text{ Words})$$

  - Example with *n=5:*

$$P(\text{sure} \mid \text{If you're going to San Francisco , be})$$
$$\approx P(\text{sure} \mid \text{San Francisco , be})$$

Counts as a word

Definitions and Markov Assumption

- *n*-gram: Word sequence of length *n* with a count
  - E.g.: 5-gram:

    If you're going to San 4

- Sequence naming:

$$w_1^{i-1} := w_1\ w_2\ \ldots w_{i-1}$$

- Markov assumption formalized:

$$P\left(w_i \middle| \underbrace{w_1^{i-1}}\right) \approx P\left(w_i \middle| \underbrace{w_{i-n+1}^{i-1}}\right)$$

$P(w_n|w_1^{n-1})$

Current sentence *n*-1 words

→ How to calculate $P(w_n|w_1^{n-1})$?

How to Calculate $P(w_n|w_1^{n-1})$

- The easiest way:
  - Maximum likelihood:

$$P_{\mathrm{ML}}(w_n|w_1^{n-1}) = \frac{c(w_1^n)}{c(w_1^{n-1})}$$

  - Example:

$$P(\text{San} \mid \text{If you're going to}) = \frac{c(\text{If you're going to San})}{c(\text{If you're going to})}$$

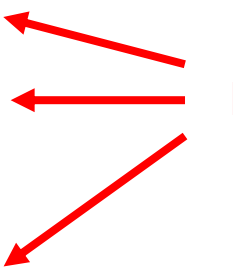
  - Still: Problem with data sparsity

Content

- Introduction
- Language Models
- **Generalized Language Models**
- Smoothing
- Implementation
- Future Work
- Summary

Intro Generalized Language Models (GLMs)

- Main idea:
  - Insert wildcard words ($*$) into sequences
- Example:
  - Instead of $P(\text{San} \mid \text{If you're going to})$:
    - $P(\text{San} \mid \text{If} \quad * \quad * \quad *)$
    - $P(\text{San} \mid \text{If} \quad * \quad * \quad \text{to})$
    - $P(\text{San} \mid \text{If} \quad * \quad \text{going} \quad *)$
    - $P(\text{San} \mid \text{If} \quad * \quad \text{going} \quad \text{to})$
    - $P(\text{San} \mid \text{If you're} \quad * \quad *)$
    - …



Length: 5, Wildcard words: 2

→ Aggregate results

  - Separate different types of GLMs based on:
    1. Sequence length
    2. Number of wildcard words

Why Generalized Language Models?

- Data sparsity of *n*-grams
  - "If you're going to San" is seen less often than for example "If * * to San"

→ Question: Does that really improve the prediction?

  - Result of evaluation: Yes

… but we should use smoothing for language models

Content

- Introduction
- Language Models
- Generalized Language Models
- **Smoothing**
- Implementation
- Future Work
- Summary

Smoothing

- Problem: Unseen sequences

  → Try to estimate probabilities of unseen sequences

  - Probabilities of seen sequences need to be reduced

- Two approaches:

  1. Backoff smoothing
  2. Interpolation smoothing

Backoff Smoothing

- If sequence unseen: use shorter sequence
  - E.g.: if $P(\text{San} \mid \text{going to}) = 0$ use $P(\text{San} \mid \text{to})$

Higher order probability

$$P_{back}(w_n|w_i^{n-1}) = \begin{cases} \tau(w_n|w_i^{n-1}) & if\ c(w_i^n) > 0 \\ \gamma * P_{back}(w_n|w_{i+1}^{n-1}) & if\ c(w_i^n) = 0 \end{cases}$$

Weight

Lower order probability (recursive)

Interpolated Smoothing

- Always use shorter sequence for calculation

$$P_{inter}\left(w_n \middle| w_i^{n-1}\right) = \underbrace{\tau\left(w_n \middle| w_i^{n-1}\right)}_{\text{Higher order probability}} + \underbrace{\gamma}_{\text{Weight}} * \underbrace{P_{inter}\left(w_n \middle| w_{i+1}^{n-1}\right)}_{\text{Lower order probability (recursive)}}$$

- Seems to work better than backoff smoothing

Kneser-Ney Smoothing [KN95] Intro

- Originally backoff smoothing
- Idea: Improve lower order calculation
- Example: Word visiting unseen in corpus

$P(\text{Francisco} \mid \text{visiting}) = 0$

→ Normal backoff: $\gamma * P(\text{Francisco})$

$P(\text{San} \mid \text{visiting}) = 0$

→ Normal backoff: $\gamma * P(\text{San})$

Also: San and Francisco have the same counts

Result: Francisco is as likely as San at that position

Is that correct?

→ Difference between Francisco and San?

Answer: Number of different contexts

Kneser-Ney Smoothing Idea

- For lower order calculation:
  - Don't use $c(w_n)$
  - Instead: Number of different bigrams the word completes:

$$N_{1+}(\bullet\, w_n) := |\{w_{n-1} : c(w_{n-1}^n) > 0\}|$$

Count

  - Or in general:

$$N_{1+}(\bullet\, w_{i+1}^n) = |\{w_i : c(w_i^n) > 0\}|$$

- Modified Kneser-Ney [CG98]: Different calculation of higher order discounts

Smoothing of GLMs

- We can use all smoothing techniques on GLMs as well!

- Small modification:

  E.g: $P(\text{San} \mid \text{If} \quad * \quad \text{going} \quad *)$

  Lower order sequence :

  – Normally: $P(\text{San} \mid * \quad \text{going} \quad *)$

  – Instead use $P(\text{San} \mid \text{going} \quad *)$

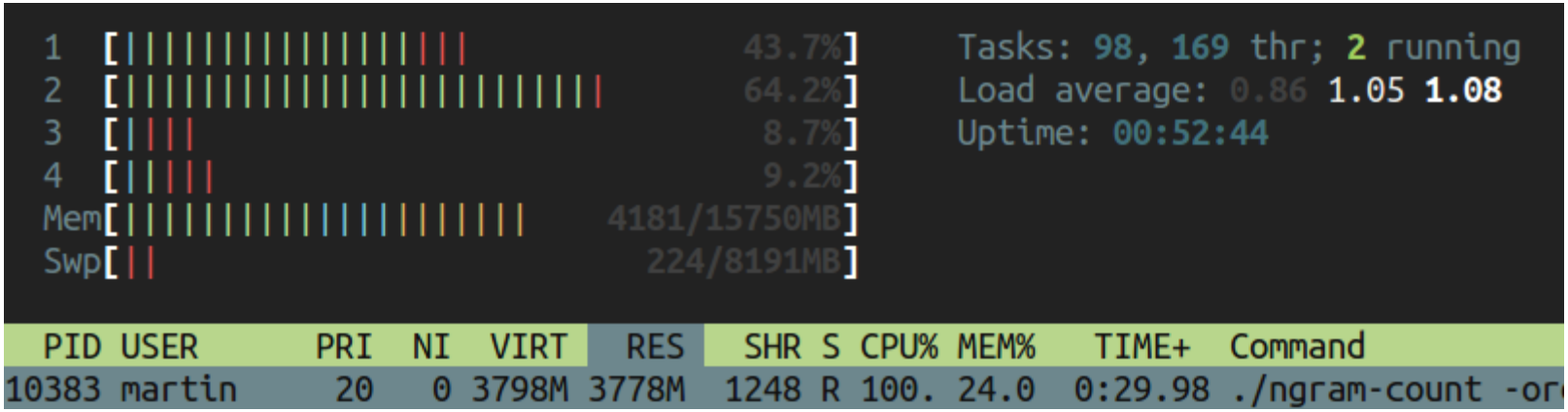
Content

- Introduction
- Language Models
- Generalized Language Models
- Smoothing
- **Implementation**
- Future Work
- Summary

Implementation: Alternatives

- SRI Language Modeling Toolkit, German Wiki (2.6 GB text)

Total: 30 Seconds

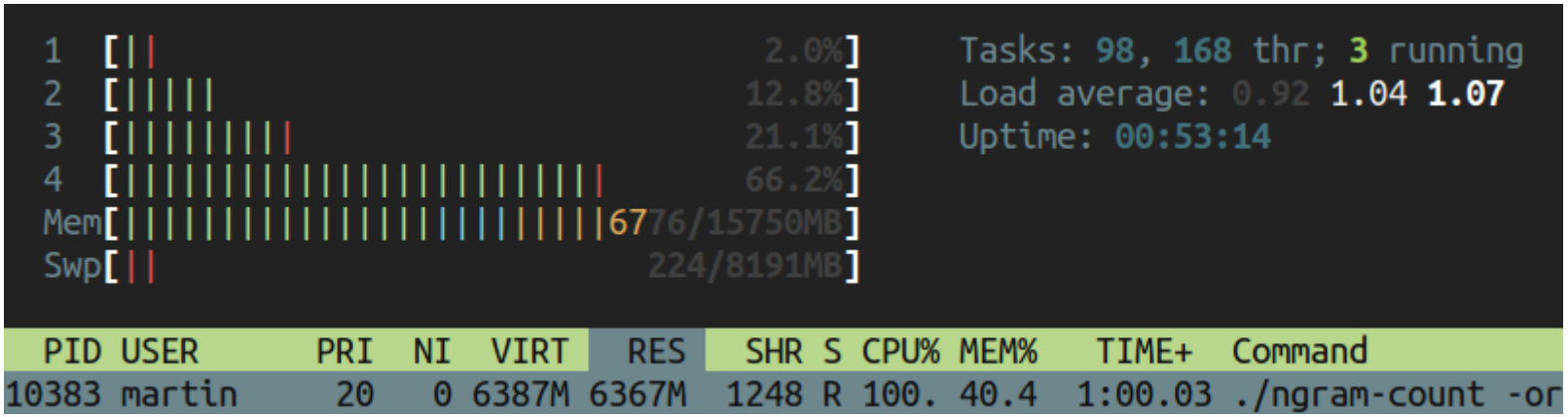


Process: 30 Seconds

Implementation: Alternatives

- SRI Language Modeling Toolkit, German Wiki (2.6 GB text)

Total: 60 Seconds

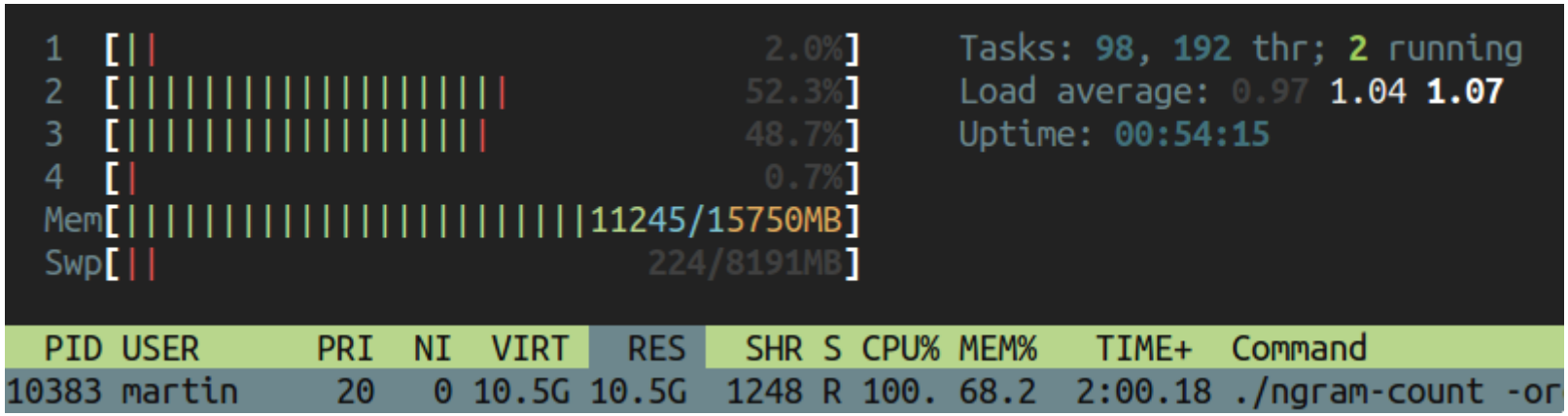


Process: 60 Seconds

Implementation: Alternatives

- SRI Language Modeling Toolkit, German Wiki (2.6 GB text)

Total: 120 Seconds

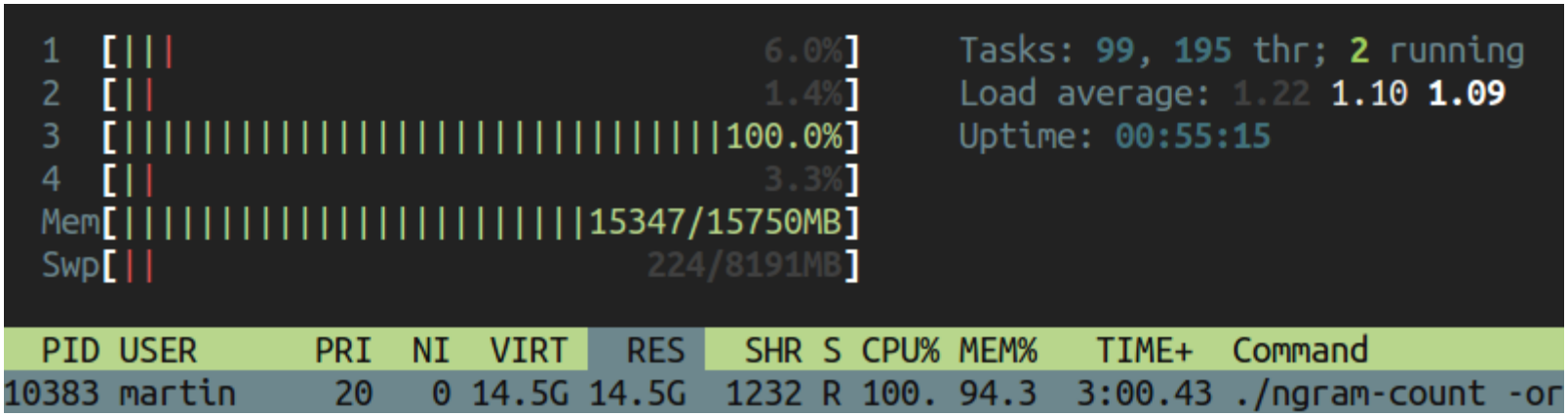


Process: 120 Seconds

Implementation: Alternatives

- SRI Language Modeling Toolkit, German Wiki (2.6 GB text)

Total: 180 Seconds

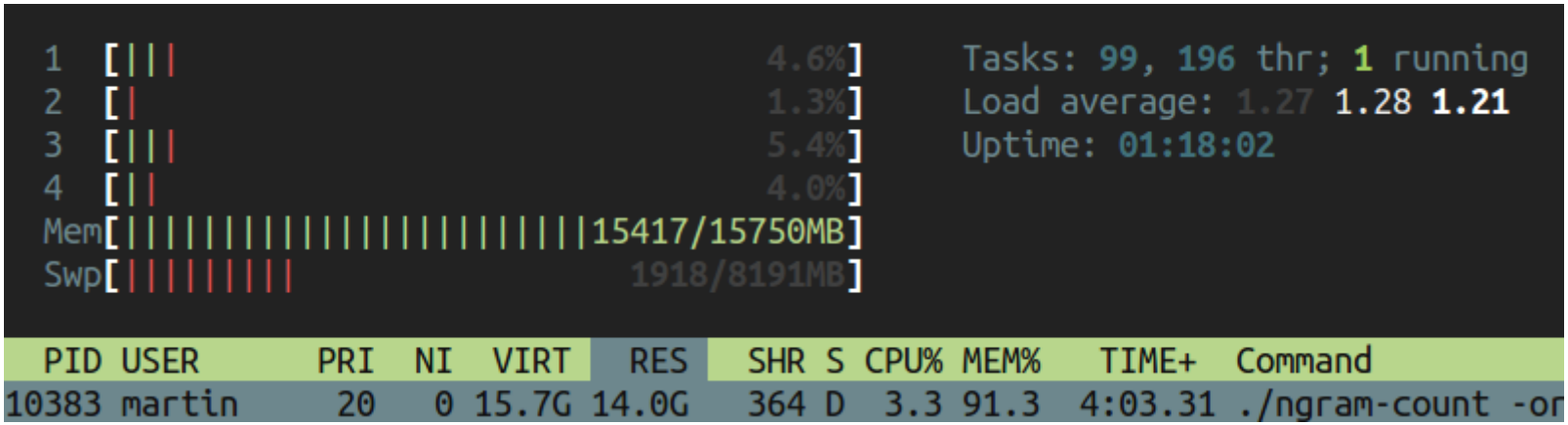


Process: 180 Seconds

Implementation: Alternatives

- SRI Language Modeling Toolkit, German Wiki (2.6 GB text)

Total: 1560 Seconds

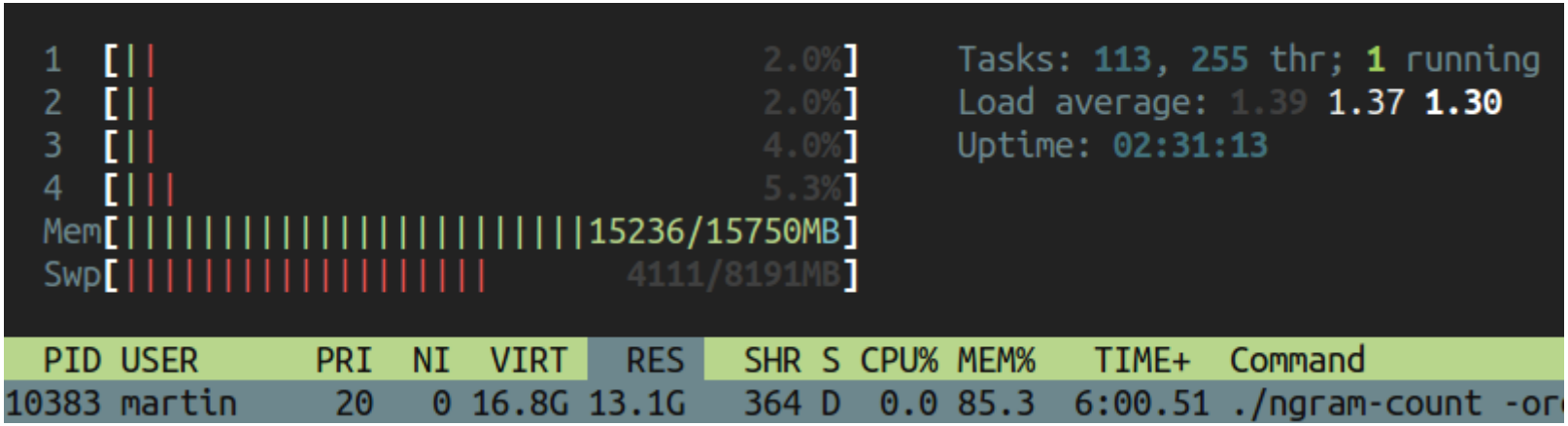


Process: 240 Seconds

Implementation: Alternatives

- SRI Language Modeling Toolkit, German Wiki (2.6 GB text)

Total: 6540 Seconds



Process: 360 Seconds

- English Wikipedia: 6.6 GB

Implementation: Alternatives

- Kyoto Language Modeling Toolkit, German Wiki (2.6 GB)

```
martin@martin-ThinkPad:~/glm/kylm$ java -cp kylm-0.0.7.jar kylm.main.CountNgrams de-traini
ng.txt de-ml5.arpa -n 5 -ml
```

Guess what is going to happen
(Hint: It's written in Java)

Implementation: Alternatives

- Kyoto Language Modeling Toolkit, German Wiki (2.6 GB)

```
martin@martin-ThinkPad:~/glm/kylm$ java -cp kylm-0.0.7.jar kylm.main.CountNgrams de-traini
ng.txt de-ml5.arpa -n 5 -ml
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at java.util.Arrays.copyOf(Arrays.java:2219)
        at java.util.Vector.grow(Vector.java:262)
        at java.util.Vector.ensureCapacityHelper(Vector.java:242)
        at java.util.Vector.insertElementAt(Vector.java:597)
        at kylm.model.ngram.BranchNode.getChild(Unknown Source)
        at kylm.model.ngram.NgramLM.countNgrams(Unknown Source)
        at kylm.model.ngram.NgramLM.trainModel(Unknown Source)
        at kylm.main.CountNgrams.main(Unknown Source)
martin@martin-ThinkPad:~/glm/kylm$
```

java.lang.OutOfMemoryError: Java heap space

☹

Implementation: Requirements

- We need …
  - … to work on large corpora (6 GB +)
  - … to work on many different corpora
  - … Generalized Language Models
  - … Modified Kneser-Ney Smoothing

Implementation: *n*-grams

- How we build *n*-grams and generalized *n*-grams:
  1. Store different sequences in many (small) files
  2. Sort the files
  3. Count the sequences (easy, because they're sorted)
  4. Store counts

- Some numbers for English Wikipedia:
  - Size of text file: 6.6 GB

Type	Different sequences	Size
5-grams	862.489.285	25 GB
(1..5)-grams	2.004.173.723	51 GB
Generalized (1..5)-grams	10.057.876.269	217 GB

Implementation: Mod. Kneser-Ney

1. Calculate different counts: $N_{1+}(\bullet\, w_{i+1}^{n})$, $N_{1+}(w_{i+1}^{n-1}\, \bullet)$ …
2. Calculate lowest order probability
3. Calculate second-lowest order probability
4. Aggregate lowest and second-lowest order probabilities

   → Problem: Sorting of those sequences

   Example: Higher order: to San

   Lower order: San

   Solution: Sort higher order seq. starting at second word
5. Repeat steps 3 and 4 with higher order sequences

- Introduction
- Language Models
- Generalized Language Models
- Smoothing
- Implementation
- **Future Work**
- Summary

Future Work

- Evaluation
  - Evaluating next word prediction
  - More general metrics like perplexity and cross-entropy

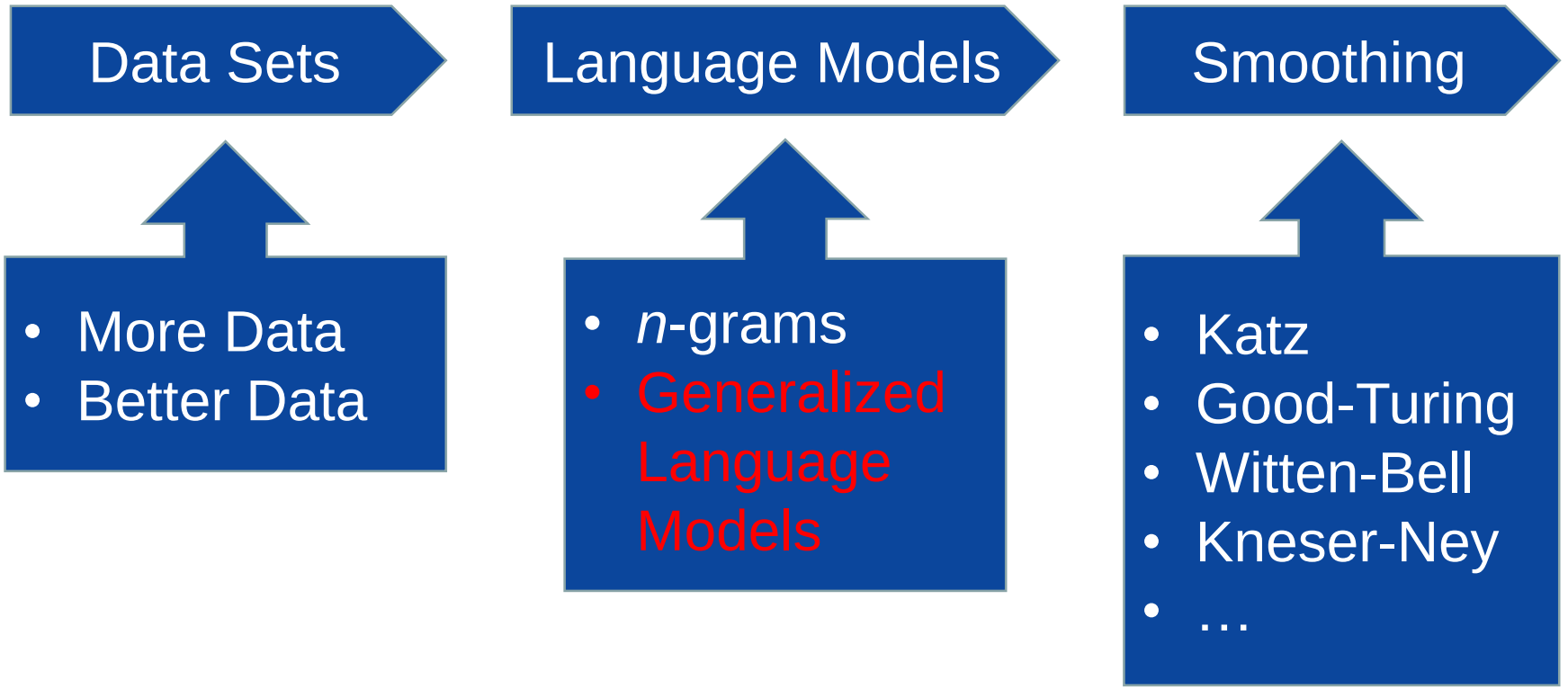
- Different way of backing off at Kneser-Ney for Generalized Language Models (idea from Thomas Gottron)

- Code cleanup
- Better documentation
- …

Content

- Introduction
- Language Models
- Generalized Language Models
- Smoothing
- Implementation
- Future Work
- **Summary**

Summary


Data Sets
Language Models
Smoothing
More Data
Better Data
n-grams
Generalized Language Models
Katz
Good-Turing
Witten-Bell
Kneser-Ney
…

Find my bachelor thesis online at:

http://mkrnr.de/bt

That's MKRNR

Also, the slides from …

… last Oberseminar talk: http://mkrnr.de/c

… today: http://mkrnr.de/d

Thank you for your attention!

Questions?

- **Images:**
  - Wheelchair Joystick (Slide 4): http://i01.i.aliimg.com/img/pb/741/422/527/527422741_355.jpg
  - Smartphone Keyboard (Slide 4): https://activecaptain.com/articles/mobilePhones/iPhone/iPhone_Keyboard.jpg
- **References:**

  - **[CG98]:** Stanley Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. Technical report, Technical Report TR-10-98, Harvard University, August, 1998.
  - **[JM80]:** F. Jelinek and R.L. Mercer. Interpolated estimation of markov source parameters from sparse data. In Proceedings of the Workshop on Pattern Recognition in Practice, pages 381–397, 1980.
  - **[KN95]:** Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on, volume 1, pages 181–184. IEEE, 1995.