Hi, I'm ...

Konstantin Tennhard

Ruby Developer at flinc

Natural Language Processing (NLP) with **JRuby** and **OpenNLP**

Motivation

Language and stuff ...

Language

Sharing Information

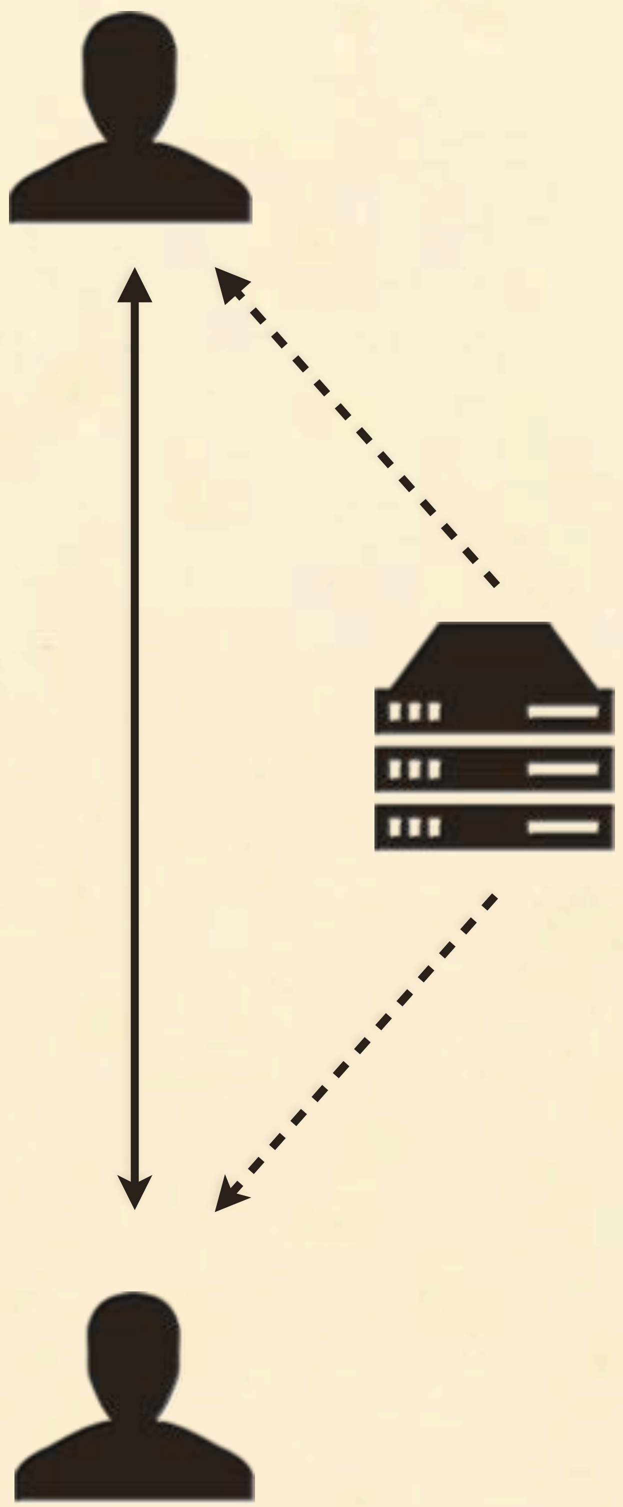
Language is the most natural way to communicate with others. It is excellent for encoding information.

Language

Flow of Information

Language

Flow of Information



Language

Representation

Natural language can be represented as a **series of sounds** or as a **series of characters**.

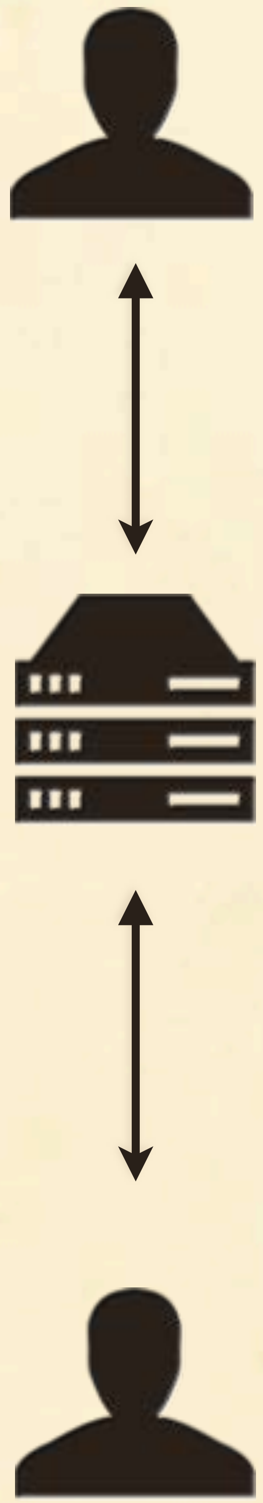
Natural Language Processing

Intelligent Machines

With the help natural language processing methods, we **enable machines to understand and process language**.

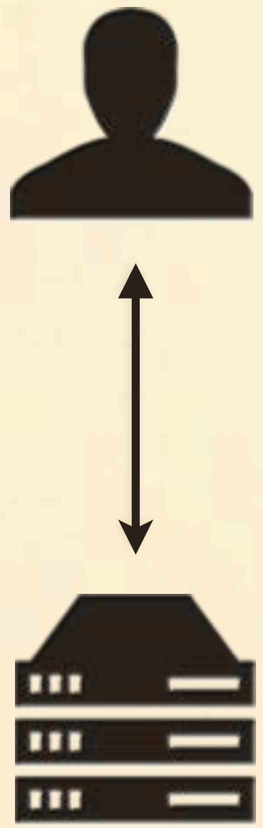
Natural Language Processing

Intermediate Processing,
e.g., Automatic Translation



Natural Language Processing

Human-to-Machine Communication



Examples

… we won’t talk about.

Machine Translation
Text Summarization
Opinion Mining

Examples

… we will talk about!

Named Entity Recognition

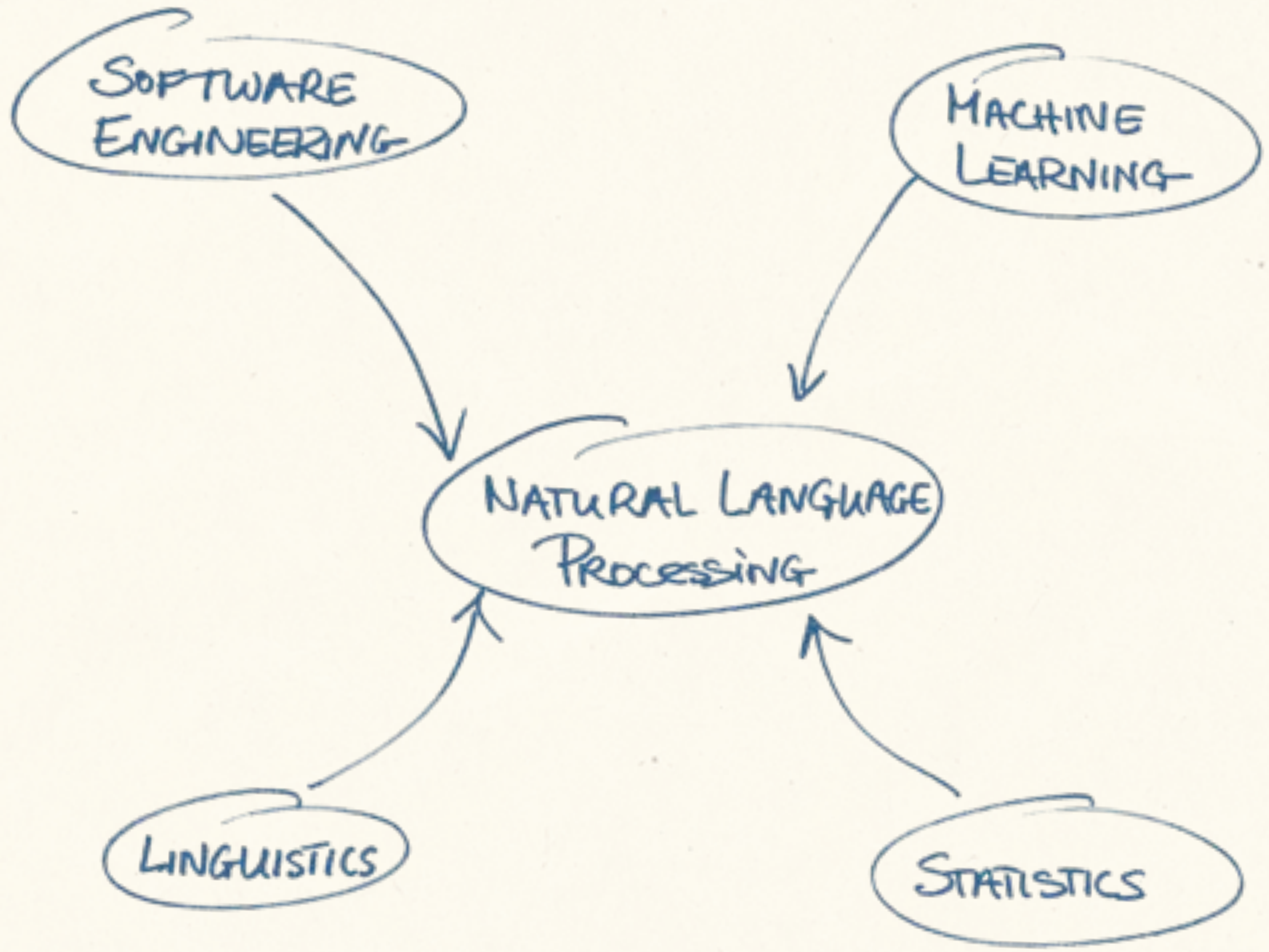
Keyword Extraction

Natural Language Processing

A Combination of Many Subjects

Natural Language Processing

A Combination of Many Subjects



Linguistic Basics

No Ruby, yet. Hang in there.

Part of Speech

The part of speech or word class of a word denotes its **syntactic function**.

Words can have multiple classes, e.g., ‘to fly’ (Verb) and ‘a fly’ (Noun).

Word Stem

The stem of a word is the part of the word that is common to all its derived variants.

The stem of a word can be an artificial construct.

Technology

y u no MRI ...

JRuby

Ruby is a very **expressive** language with **excellent string processing** capabilities.

The JVM is a **high performance** platform with **true multi-threading** capabilities.

Excellent java libraries for natural language processing exist.

OpenNLP

Machine Learning Based NLP Toolkit

OpenNLP is solely based on **machine learning** methods. It uses **maximum entropy classification** to perform natural language processing tasks.

http://opennlp.apache.org/

OpenNLP

Pre-Trained Models

Maximum entropy classifiers **have to be trained** before they can be utilized.

Pre-trained models can be downloaded from SourceForge: http://opennlp.sourceforge.net/models-1.5/

OpenNLP

Three Steps

1. Load an existing model or create a new one from your own training data.
2. Initialize the classifier using this model as input.
3. Perform the actual classification task.

OpenNLP

The Gems

Minimal wrapper around the original OpenNLP implementation:

- Automatic conversion between Ruby and Java datatypes
- Unified Interface

Separate Gems for English and German model files.

NLP Basics

Automating linguistic analyses ...

Segmentation

String → Sequence of Logical Units

The problem of segmentation is concerned with **splitting a text** into a **sequence of logical units**.

Different instances of this problem exist.

Sentence Detection

Text → Sentences

Sentence detection is the process of **segmenting a text into sentences**.

The problem is harder than it looks:

- Ruby is awesome. Ruby is great!
- “Stop it!”, Mr. Smith shouted across the yard. He was clearly angry.

Sentence Detection

Text → Sentences

```
m = OpenNLP::English.sentence_detection_model
d = OpenNLP::SentenceDetector.new(m)
r = d.process <<-TEXT
Ruby is awesome. Ruby is great!
TEXT

r[0] # => "Ruby is awesome."
r[1] # => "Ruby is great!"
```

Tokenization

Sentence → Words

Tokenization is the task of **detecting word boundaries**.

Challenges:

- Languages with **no visual representation** of **word boundaries**
- Multiple separators

Tokenization

String → Linguistic Units

```
m = OpenNLP::English.tokenization_model
t = OpenNLP::Tokenizer.new(m)
r = t.process("I shot an elephant in my pajamas.")

r # => ["I", "shot", "an", "elephant", "in", "my",
"pajamas", "."]
```

Part-of-Speech Tagging

Tokens → Tags

Part-of-Speech tagging is concerned with **identifying a word's class** in a given context.

A common format for representing Part-of-Speech tags is the **Penn Treebank tag set**.

Part-of-Speech Tagging

Tokens → Tags

```
m = OpenNLP::English.pos_tagging_model
t = OpenNLP::POSTagger.new(m)
r = t.process(%w[Ruby is awesome])

r[0] # => NNP
r[1] # => VBZ
r[2] # => JJ
```

Stemming

Inflected word → Word stem

Stemming is the processes of applying a set of rules to **remove morphological suffixes**.

Porter's stemmer is probably the most popular stemmer.

Stemming

Inflected word → Word stem

```
# https://github.com/raypereda/stemmify
require 'stemmify'

"programming".stem # => "program"
```

Named Entity Recognition

Tokens → Names | Locations | . . .

Named entities are noun phrases that refer to individuals, organizations, locations, etc.

Named Entity Recognition is concerned with identifying named entities in a given text.

Named Entity Recognition

Tokens → Names | Locations | . . .

```
tokens = %w[This summer EuRuKo comes to Athens
for two days on the 28th and 29th of June .]

m = OpenNLP::Models.
  named_entity_recognition_model(:location)
f = OpenNLP::NameFinder.new(m)
ranges = f.process(tokens)
ranges.map { |r| tokens[r] } # => ["Athens"]
```

Software Engineering

Bringing it all together …

Properties of NLP Task

NLP tasks can often be expressed as a **sequence of steps** that is **executed linearly**.

Hence, we can use **processing pipelines** to solve NLP problems.

Processing Pipelines

A processing pipeline is a set software **components connected in series.**

The **output of one component** is the **input of the next one**.

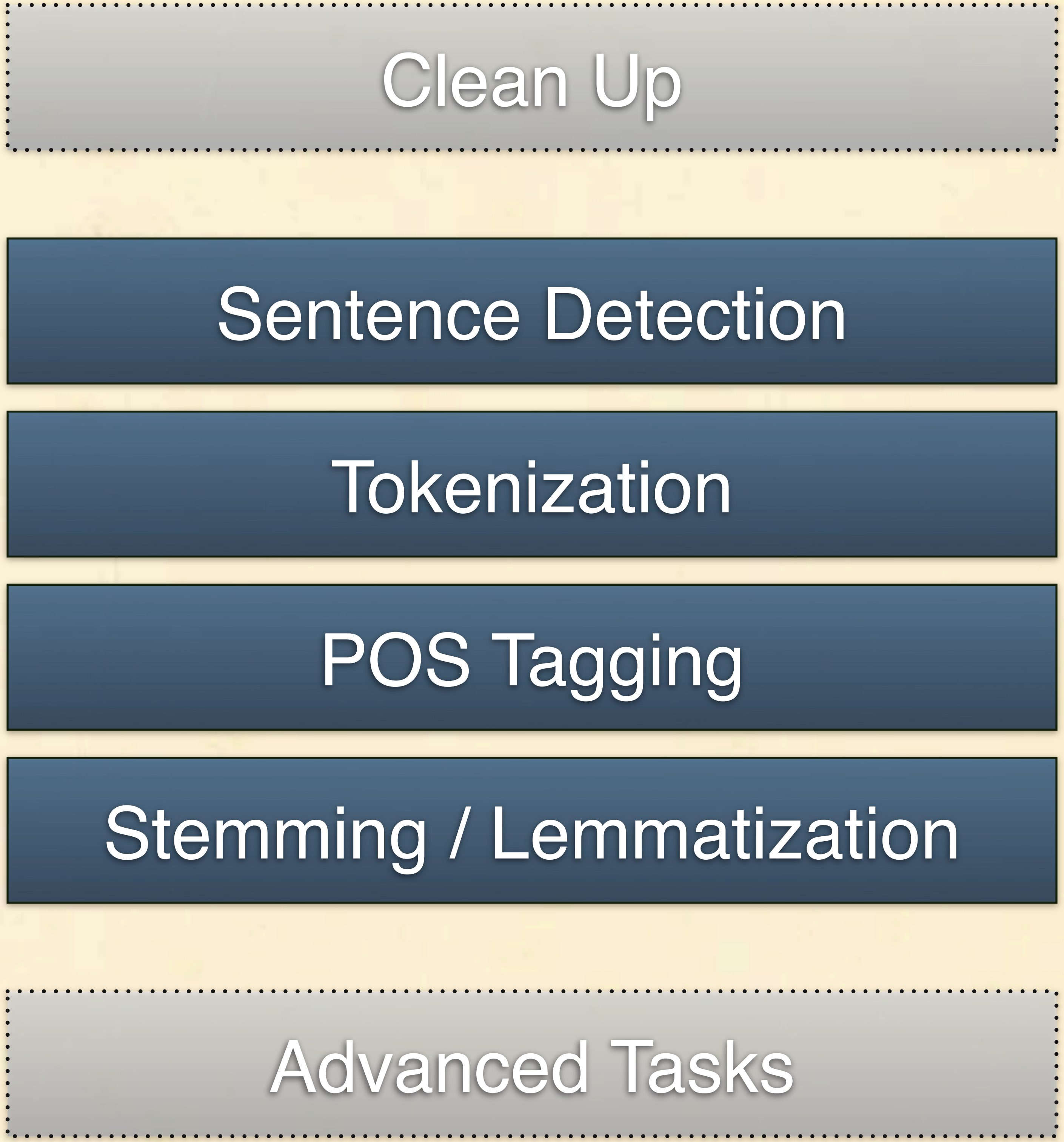
Composable Operations

t6d/composable_operations

A flexible Ruby implementation of a processing pipeline:

- `Operation` represents a single processing component.
- `ComposedOperation` represents a processing pipeline, but can also be used as a component in an other pipeline.

Pre-Processing Pipeline



Clean Up

Sentence Detection

Tokenization

POS Tagging

Stemming / Lemmatization

Advanced Tasks

Pre-Processing Pipeline

Definition

```ruby
require 'composable_operations'
include ComposableOperations

class PreProcessing < ComposedOperation
  use SentenceDetection
  use Tokenization
  use POSTagging
end
```

Pre-Processing Pipeline

Sentence Detection Component

```
require 'opennlp'
require 'opennlp-english'
require 'opennlp-german'
require 'composable_operations'
include ComposableOperations

class SentenceDetection < Operation
  processes :text
  property :language, default: :en,
                      converts: :to_sym,
                      required: true,
                      accepts: [:en, :de]

  def execute
    detector = OpenNLP::SentenceDetector.new(model)
    detector.process(text)
  end

  protected

  def model
    case language
    when :en
      OpenNLP::English.sentence_detection_model
    when :de
      OpenNLP::German.sentence_detection_model
    end
  end
end
```

Pre-Processing Pipeline

Tokenization Component

```ruby
require 'opennlp'
require 'opennlp-english'
require 'opennlp-german'
require 'composable_operations'
include ComposableOperations

class Tokenization < Operation
  processes :sentences
  property :language, default: :en,
                      converts: :to_sym,
                      required: true,
                      accepts: [:en, :de]

  def execute
    tokenizer = OpenNLP::Tokenizer.new(model)
    Array(sentences).map do |sentence|
      tokenizer.process(sentence)
    end
  end

  protected

  def model
    # ...
  end
end
```

Pre-Processing Pipeline

POS Tagging Component

```ruby
require 'opennlp'
require 'opennlp-english'
require 'opennlp-german'
require 'composable_operations'
include ComposableOperations

class POSTagging < Operation
  processes :sentences
  property :language, default: :en,
                      converts: :to_sym,
                      required: true,
                      accepts: [:en, :de]

  def execute
    tagger = OpenNLP::POSTagger.new(model)

    sentences.map.with_index do |sent, sent_idx|
      tags = tagger.process(sent)
      tags.map.with_index do |tag, tkn_idx|
        [sentences[sent_idx][tkn_idx], tag]
      end
    end
  end

  protected

  def model
    # ...
  end
end
```

Pre-Processing Pipeline

Execution

```
PreProcessing.perform("Ruby is awesome. Ruby is great!")

# Returns:
#
# [
#   [
#     ["Ruby", "NNP"],
#     ["is", "VBZ"],
#     ["awesome", "JJ"],
#     [".", "."]
#   ],
#   [
#     ["Ruby", "NNP"],
#     ["is", "VBZ"],
#     ["great", "JJ"],
#     ["!", "."]
#   ]
# ]
```

Keyword Extraction

Let's talk about the good stuff ...

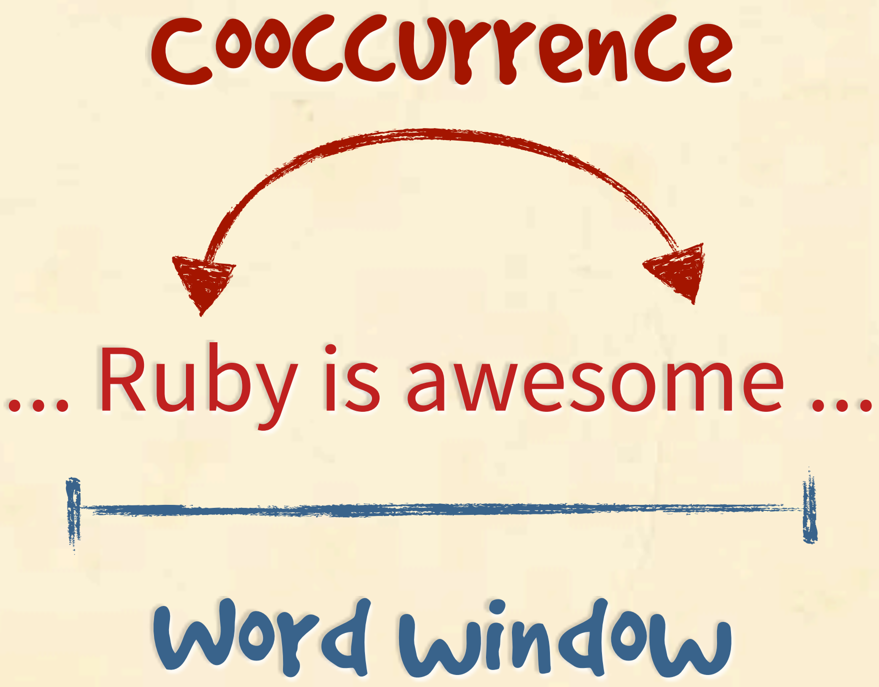
TextRank

TextRank is a **graph-based algorithm** heavily inspired by Google's PageRank algorithm.

The algorithm was published by **Rada Mihalcea and Paul Tarau**: http://acl.ldc.upenn.edu/acl2004/emnlp/pdf/Mihalcea.pdf

Cooccurrence

Linguistics ... again!



cooccurrence

... Ruby is awesome ...

word window

Keyword Extraction Pipeline

1. Preprocessing
   Sentence Detection, Tokenization, POS Tagging, Normalization through Stemming, Token Filtering
2. Cooccurrence Calculation
3. Coocurrence Graph Construction
4. Text Rank Calculation
5. Sorting and Extracting Nodes

Keyword Extraction Pipeline

```
class KeywordRanking < ComposedOperation

  use PreProcessingPipeline, filter: [/^NN/, /^JJ/]
  use CooccurrenceCalculation
  use CooccurrenceGraphConstruction
  use PageRankCalculation
  use NodeSortingAndExtraction

end

KeywordRanking.perform(...)
```

Code

The code can be found on Github:

https://github.com/t6d/keyword_extractor

Be nice, it’s just some demo code – not for use in production. ;)

Summary

Natural Language Processing with JRuby and OpenNLP
by Konstantin Tennhard

GitHub: t6d
Twitter: t6d

Code can be found on GitHub:

* http://github.com/t6d/opennlp
* http://github.com/t6d/opennlp-english
* http://github.com/t6d/opennlp-german
* http://github.com/t6d/opennlp-examples

* http://github.com/t6d/keyword_extractor

* http://github.com/t6d/composable_operations
* http://github.com/t6d/smart_properties

Any questions? Feel free to approach me anytime throughout the conference or send me a tweet, if that’s what you prefer.

Summary

```
 _____ _   _    _    _   _ _  ______  
|_   _| | | |  / \  | \ | | |/ / ___| 
  | | | |_| | / _ \ |  \| | ' /\___ \ 
  | | |  _  |/ ___ \| |\  | . \ ___) |
  |_| |_| |_/_/   \_\_| \_|_|\_\____/ 
```