ALeF: Active Learning Framework for Readability Prediction

Philip van Oosten — Véronique Hoste

LT³, Language and Translation Technology Team

Atila, September 23, 2010
Overview

1. Problem setting
   - In general
   - Our proposed solution for readability prediction

2. The Wolfit workflow engine
   - Wolfit API
   - Wolfit Editor screenshots

3. ALeF: active learning on top of Wolfit
   - Cascaded approach
   - With active learning: ALeF
   - Solutions provided by ALeF
Overview

1. Problem setting
   - In general
   - Our proposed solution for readability prediction

2. The Wolfit workflow engine
   - Wolfit API
   - Wolfit Editor screenshots

3. ALeF: active learning on top of Wolfit
   - Cascaded approach
   - With active learning: ALeF
   - Solutions provided by ALeF
Overhead for NLP experiments

**Infrastructure**
- setting up software infrastructure: in many cases overhead or done manually
- maintenance of complex tasks → overhead.

**Resources**
- corpus annotation often necessary
- data sets often used in isolation
  ⇒ experimental results often incomparable
- data set used in more tasks or settings
  ⇒ more valuable to compare methodologies
- more data sets per task
  ⇒ better comparison of methodologies
Possible solutions

**Infrastructure**
- applications of which small parts can be adapted.
- workflow engines
- ...

**Resources**
- share resources
- use same data in multiple settings
- ...
Readability prediction is a small domain

⇒ try to provide solutions that can be generalized to other domains
(but first focus on readability)
Our solution for Readability Prediction

Readability prediction is a small domain

⇒ try to provide solutions that can be generalized to other domains
(but first focus on readability)

- a workflow engine built to reduce overhead
- built on top of that: ALeF, a complete framework for readability research

Status: implementation of the workflow engine
Overview

1. Problem setting
   - In general
   - Our proposed solution for readability prediction

2. The Wolfit workflow engine
   - Wolfit API
   - Wolfit Editor screenshots

3. ALeF: active learning on top of Wolfit
   - Cascaded approach
   - With active learning: ALeF
   - Solutions provided by ALeF
Advantages of workflow engines?

Separation of concerns, so that users need only minimal knowledge of

- multiple programming languages
- pipelining/parallelization
- in-detail usage of resources
A workflow engine focused on rapid development

- API to construct and execute workflows
- Web application to construct workflows
  (django-wolfit)
- GNU GPLv2 on Google Code

http://code.google.com/p/wolfit
http://code.google.com/p/django-wolfit
Wolfit API

Functionality:
- construct/configure/execute workflows
- expose workflows as Python functions
- import and export
- visualization

To do:
- abstract workflows
- more task types: CLI, CObject, Java, ...
- workflows as tasks in other workflows
- more control structures
- ...
An empty editor. Left: available modules
Right: Name, description, minimap, info
Top: New, load, save, delete, ... buttons
A function connected to workflow input and output.
The currently available modules.
A python function, taking keyword arguments as input and a dictionary as return value.
A workflow input
A workflow output

![Workflow output](image)
A connection between two functions
You can connect modules by dragging from one *terminal* to another.
An example of what a more elaborate workflow can look like.
A feedback module
Feedback loops are like while loops. They enable active learning, backtracking, heuristic algorithms, ...
Creating and executing a workflow.

By means of a **Python script**:

- Use the API directly to compose a Process object
- Configure a runner. Currently the ThreadPoolRunner is preferrable.
- Execute the workflow like you would execute a Python function

By means of **django-wolfit**:

- Create a workflow using the Django web application
- Use an *importer* to create a Process object.
- Configure a runner.
- Execute the Process like a python function.

**To do**: make creating web demos based on workflows easy
Overview

1. Problem setting
   - In general
   - Our proposed solution for readability prediction

2. The Wolfit workflow engine
   - Wolfit API
   - Wolfit Editor screenshots

3. ALeF: active learning on top of Wolfit
   - Cascaded approach
   - With active learning: ALeF
   - Solutions provided by ALeF
Figure: Readability prediction as a typical cascaded CL task.
Corpus

Text selection

Subcorpus

Select texts for assessment

Text selection depends on several factors

Inconsistencies are represented to assessors

Web application for preference judgements

Find inconsistencies

Data:
User preferences about readability of documents in the subcorpus

Select the best strategy to assign initial certainties by asking feedback from assessors

Consistent and inconsistent parts
Consistent parts will get more certainty

Use a heuristic to assign initial certainties

Readability prediction systems through n-fold CV

Web demo or readability comparison, ranking, classification or score assignment

Possibilistic knowledge base
Solutions provided by ALeF

**Infrastructure**  An extensible application was presented and will be implemented. Third party plugins can be added.

**Resources**  Encourage people to make use of ALeF and donate their data sets for research (first try the framework ourselves).
Thank you!