Building trainable taggers in a web-based, UIMA-supported NLP workbench

Rafal Rak, BalaKrishna Kolluru and Sophia Ananiadou
National Centre for Text Mining
School of Computer Science, University of Manchester
Manchester Interdisciplinary Biocentre
131 Princess St, M1 7DN, Manchester, UK
{rafal.rak,balakrishna.kolluru,sophia.ananiadou}@manchester.ac.uk

Abstract

Argo is a web-based NLP and text mining workbench with a convenient graphical user interface for designing and executing processing workflows of various complexity. The workbench is intended for specialists and non-technical audiences alike, and provides the ever expanding library of analytics compliant with the Unstructured Information Management Architecture, a widely adopted interoperability framework. We explore the flexibility of this framework by demonstrating workflows involving three processing components capable of performing self-contained machine learning-based tagging. The three components are responsible for the three distinct tasks of 1) generating observations or features, 2) training a statistical model based on the generated features, and 3) tagging unlabelled data with the model. The learning and tagging components are based on an implementation of conditional random fields (CRF); whereas the feature generation component is an analytic capable of extending basic token information to a comprehensive set of features. Users define the features of their choice directly from Argo's graphical interface, without resorting to programming (a commonly used approach to feature engineering). The experimental results performed on two tagging tasks, chunking and named entity recognition, showed that a tagger with a generic set of features built in Argo is capable of competing with task-specific solutions.

1 Introduction

The applications of automatic recognition of categories, or tagging, in natural language processing (NLP), range from part of speech tagging to chunking to named entity recognition and complex scientific discourse analyses. Currently, there is a variety of tools capable of performing these tasks. A commonly used approach involves the use of machine learning to first build a statistical model based on a manually or semi-automatically tagged sample data and then to tag new data using this model. Since the machine learning algorithms for building models are well established, the challenge shifted to *feature engineering*, i.e., developing task-specific features that form the basis of these statistical models. This task is usually accomplished programmatically which pose an obstacle to a non-technically inclined audience. We alleviate this problem by demonstrating Argo[1], a web-based platform that allows the user to build NLP and other text analysis workflows via a graphical user interface (GUI) available in a web browser. The system is equipped with an ever growing library of text processing components ranging from low-level syntactic analysers to semantic annotators. It also allows for including user-interactive components, such as an annotation editor, into otherwise fully automatic workflows. The interoperability of processing components is ensured in Argo by adopting Unstructured Information Management Architecture (UIMA) (Ferrucci and Lally, 2004) as the system's framework. In this work we explore the capabilities of this framework to support machine

[1] http://nactem.ac.uk/Argo

Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, pages 121–126,
Jeju, Republic of Korea, 8-14 July 2012.

learning components for tagging textual content.

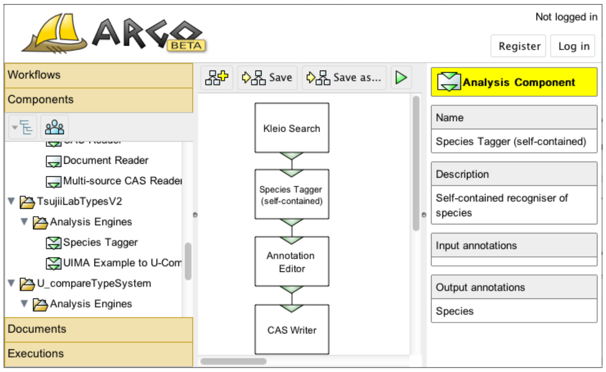
In the following section we present related work. Section 3 provides background information on Argo and its relationship to UIMA. The details of the three machine learning components are discussed in Section 4. Section 5 provides evaluation, whereas Section 6 concludes the paper.

2 Related work

Language processing tools with machine learning capabilities for tagging textual content have been distributed by various groups in form of either standalone applications or application programming interfaces (API). Packages such as Lingpipe[2], Mallet[3], Stanford NLP tools[4] and OpenNLP[5] have been extensively used by the NLP and text mining communities (Kolluru et al., 2011; Corbett and Murray-Rust, 2006). However, such tools inherently impose inconveniences on users, such as a lack of GUI, often arduous manual installation procedures, proficiency in programming or familiarity with the details of machine learning algorithms.

These limitations are overcome by GUI-equipped, workflow-supporting platforms that often directly use the solutions provided by the former tools. The notable examples of such platforms designed specifically for NLP and text mining tasks are GATE (Cunningham et al., 2002), a suite of text processing and annotation tools, and U-Compare (Kano et al., 2010), a standalone application supporting the UIMA framework that formed the inspiration for Argo.

Although the GUI platforms provide machine learning solutions, these are usually limited to using pre-trained models and providing a rich set of features for training requires resorting to programming. Argo, on the other hand, allows the users to train their own models with either a generic set of features or customisable features without having to write a single line of code. This capability is provided in Argo entirely through its GUI.



Name: Species Tagger (self-contained)

Description: Self-contained recogniser of species

Input annotations:

Output annotations: Species

Figure 1: Screen capture of Argo's web-based interface.

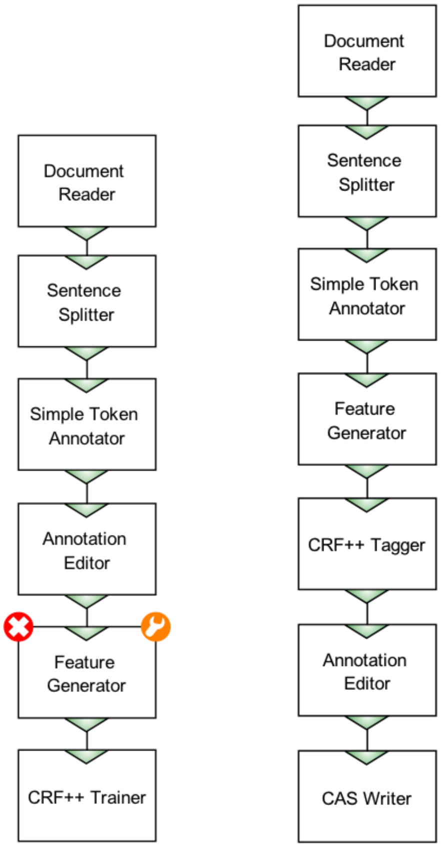
3 Argo and UIMA

Argo's main user interface consists of three panels as shown in Figure 1. The left-hand panel includes user-owned or shared storable objects; the middle panel is a drawing space for constructing workflows and the right-hand panel displays context-dependent information. The storable objects are categorised into *workflows*, represented as block diagrams of interconnected processing components, *documents* that represent the user's space intended for uploading resources and saving processing results, and *executions* that provide past and live workflow execution details and access points to user-interactive components should such be present in a workflow.

Component interoperability in Argo is ensured by UIMA which defines common structures and interfaces. A typical UIMA processing pipeline consists of a *collection reader*, a set of *analysis engines* and a *consumer*. The role of a collection reader is to fetch a resource (e.g., a text document) and deposit it in a *common annotation structure*, or *CAS*, as the subject of annotation. Analysis engines then process the subject of annotation stored in the CAS and populate the CAS with their respective annotations. The consumer's role is to transform some or all of the annotations and/or the subject of annotation from the CAS and serialise it into some storable format.

Readers, analysers and consumers are represented graphically in Argo as blocks with incoming only, incoming and outgoing, and outgoing only ports, respectively, visible in the middle of Figure 1.

[2] http://alias-i.com/lingpipe
[3] http://mallet.cs.umass.edu
[4] http://nlp.stanford.edu/software/index.shtml
[5] http://opennlp.apache.org



(a) Training (b) Tagging

Figure 2: Two generic workflows demonstrating the use of the Feature Generator component for (a) training and (b) tagging.

4 Machine learning components in Argo

In order to ensure flexibility in building workflows, we split the machine learning capability into three distinct processing components, namely feature generator, model trainer and tagger. The trainer and the tagger are intrinsic machine learning components, whereas the feature generator is a convenient and customisable processing component capable of building a feature space for a user-defined domain.

From UIMA's perspective, the feature generator and the tagger are both analysis engines whose purpose is to analyse the incoming CASes and enrich them with additional annotations; whereas the trainer is a consumer that transforms the information stored in CASes into a statistical model.

A typical use of the three components is shown in Figure 2. The three components are represented as the Feature Generator, CRF++ Trainer and CRF++ Tagger blocks. Figure 2a shows a process of building a statistical model supported by a document reader, common, well-established preprocessing components (in this case, to establish boundaries of sentences and tokens), and the previously mentioned editor for manually creating annotations[6]. The manual annotations serve to generate tags/labels which are used in the training process together with the features produced by Feature Generator. The trained model is then used in the workflow shown in Figure 2b to tag new resources. Although the tagging workflow automatically recognises the labels of interest (based on the model supplied in CRF++ Tagger), in practice, the labels need further correction, hence the use of Annotation Editor after the tagger.

4.1 Training and tagging

At present, our implementation of the training and tagging components is based on the conditional random fields (CRF) (Lafferty et al., 2001). Our choice is dictated by the fact that CRF models are currently one of the best models for tagging and efficient algorithms to compute marginal probabilities and n-best sequences are freely available.

We used the CRF++ implementation[7] and wrapped it into two UIMA-compatible components, CRF++ Trainer and CRF++ Tagger. The trainer deals with the optimisation of feature parameters, whereas word observations are produced by Feature Generator, as described in the following section.

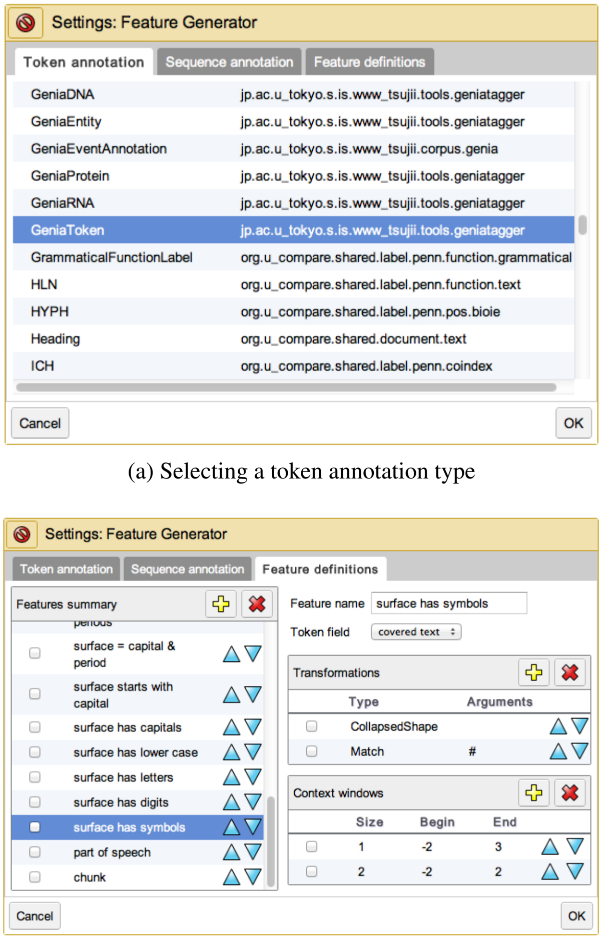
4.2 From annotations to features

The Feature Generator component is an intermediary between annotations stored in CASes and the training component. This component is customisable via the component's settings panel, parts of which are shown in Figure 3. The panel allows the user to 1) identify the stream of tokens[8] (Figure 3a), 2) identify the stream of token sequences (usually

[6] The preprocessing and manual annotation components could be replaced with CAS Reader, a component capable of supplying the workflow with a previously annotated set of documents.

[7] http://code.google.com/p/crfpp/

[8] The definition of *token* depends on the selected UIMA annotation type. It may range from a simple span of text to a complex lexical or semantic structure.



GeniaDNA	jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger
GeniaEntity	jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger
GeniaEventAnnotation	jp.ac.u_tokyo.s.is.www_tsujii.corpus.genia
GeniaProtein	jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger
GeniaRNA	jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger
GeniaToken	jp.ac.u_tokyo.s.is.www_tsujii.tools.geniatagger
GrammaticalFunctionLabel	org.u_compare.shared.label.penn.function.grammatical
HLN	org.u_compare.shared.label.penn.function.text
HYPH	org.u_compare.shared.label.penn.pos.bioie
Heading	org.u_compare.shared.document.text
ICH	org.u_compare.shared.label.penn.coindex

(a) Selecting a token annotation type

Features summary

	periods
□	surface = capital & period
□	surface starts with capital
□	surface has capitals
□	surface has lower case
□	surface has letters
□	surface has digits
□	surface has symbols
□	part of speech
□	chunk

Feature name: surface has symbols

Token field: covered text

Transformations

	Type	Arguments
□	CollapsedShape	
□	Match	#

Context windows

	Size	Begin	End
□	1	-2	3
□	2	-2	2

(b) Defining features

Figure 3: Feature Generator settings panel allows the user to (a) select labels for machine learning and (b) define features.

sentences), and 3) define features or token observations (Figure 3b).

Each feature definition consists of a name, a token field, an optional list of token field transformations, and an optional set of context windows. The name is only for the user's convenience of identifying individual feature definitions. The token field is the primary subject of transformations (if any) and it is one of the data fields of the selected token annotation type. For instance, the token annotation type may define data fields such as part of speech, chunk, or lemma. By default, the system selects "covered text", i.e., the span of text covered by an annotation, since this data field is available for any annotation.

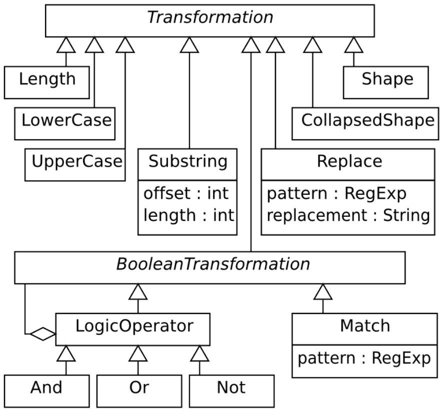


Figure 4: UML diagram of transformation types

If no transformation is declared, the string representation of the token field's value ultimately becomes the value of the generated feature. If the user declares one or more transformations then these are applied on the token field's value in sequence, i.e., an outcome of the preceding transformation becomes an input of the following one. Figure 4 shows the various transformations currently available in the system.

Context windows allow for enriching the current token's feature set by introducing observations from surrounding tokens as n-grams. For example, the selected feature definition in Figure 3b, "surface has symbols", declares the covered text as the feature's basis and defines two transformations and two context windows. The two transformations will first transform the covered text to a collapsed shape (e.g., "NF-kappa" will become "A#a") and then produce "Y" or "N" depending on whether the collapsed shape matches the simple regular expression "#" (e.g., "A#a" will become "Y"). The two context windows define six unigrams and four bigrams, which will ultimately result in this single feature definition's producing ten observations for training.

5 Evaluation

We show the performance of taggers trained with two distinct sets of features, basic and extended. The basic set of features uses token fields such as the covered text and the part of speech *without* any transformations or context n-grams. The extended set makes the full use of Feature Generator's settings and enriches the basic set with various transformations and context n-grams. The transformations in-

Dataset	Setup	P	R	F
CoNLL	Best	94.29	94.01	94.13
	L2 IOBES	92.20	93.43	92.81
	L2 IOB	92.14	93.27	92.70
	L1 IOBES	91.95	93.17	92.55
	L1 IOB	91.83	93.11	92.46
	Baseline	72.58	82.14	77.07
BioNLP/ NLPBA	Best	76.00	69.40	72.6
	L1 IOBES	66.22	65.06	65.63
	L2 IOB	66.06	64.87	65.46
	L1 IOB	66.05	64.61	65.32
	L2 IOBES	65.77	64.79	65.28
	Baseline	52.60	43.60	47.70

Table 1: Performance of various setups (L1 vs L2, and IOB vs IOBES) on the chunking and NER tasks. The setups are ordered by F-score.

Dataset	Setup	P	R	F
CoNLL	Basic	73.80	84.50	78.78
	Extended	92.20	93.43	92.81
BioNLP/ NLPBA	Basic	37.06	48.13	41.88
	Extended	66.22	65.06	65.63

Table 2: Comparison of setups with basic and extended features for the chunking and NER tasks.

clude surface shape, length, prefixes, suffixes, and the presence of various combinations of letters, digits and symbols. The context n-grams include unigrams for all feature definitions and bigrams for selected ones. Figure 3b shows a sample of the actual extended set.

We use two datasets, one prepared for the CoNLL 2000 shared task (Tjong et al., 2000) and another prepared for the BioNLP/NLPBA 2004 shared task (Kim et al., 2004). They represent two different tagging tasks, chunking and named entity recognition, respectively. The CoNLL 2000 chunking dataset involves 10 labels and comes pre-tokenised with 211,727 tokens in the training set and 47,377 tokens in the test set. The dataset also provides part-of-speech tags for each token. The BioNLP/NLPBA 2004 named entity recognition dataset involves five biology-related labels and consists of 472,006 and 96,780 tokens in the training and testing sets, respectively. Contrary to the former dataset, there is no other information supporting the tokens in the BioNLP/NLPBA dataset. To compensate for it we automatically generated part of speech and chunk labels for each token.

The chosen datasets/tasks are by no means an exhaustive set of representative comparative-setup datasets available. Our goal is *not* to claim the superiority of our approach over the solutions reported in the respective shared tasks. Instead, we aim to show that our generic setup is comparable to those task-tuned solutions.

We further explore the options of both Feature Generator and CRF++ Trainer by manipulating labelling formats (IOB vs IOBES (Kudo and Matsumoto, 2001)) for the former and parameter estimation algorithms (L_2- vs L_1-norm regularisation) for the latter. Ultimately, there are 32 setups as the result of the combinations of the two feature sets, the two datasets, the two labelling formats and the two estimation algorithms.

5.1 Results

Table 1 shows the precision, recall and f-scores of our extended-feature setups against each other as well as with reference to the best and baseline solutions as reported in the respective shared tasks. The gap to the best performing solution for the chunking task is about 1.3% points in F-score, ahead of the baseline by 15.7% points. Respectively for the NER task, our best setup stands behind the best reported solution by about 7% points, ahead of the baseline by about 18% points. In both instances our solution would be placed in the middle of the reported rankings, which is a promising result, especially that our setups are based solely on the tokens' surface form, part of speech, and (in the case of the NER task) chunk. In contrast, the best solutions for the NER task involve the use of dictionaries and advanced analyses such as acronym resolution.

The tested combinations of the labelling formats and parameter estimation algorithms showed to be inconclusive, with a difference between the best and worst setups of only 0.35% points for both tasks.

The advantage of using the extended set of features over the basic set is clearly illustrated in Table 2. The performance of the basic set on the chunking dataset is only at the level of the baseline, whereas for the NER task it falls nearly 6% points behind the

Dataset	Setup	L2	L1
CoNLL	Extended IOB	555	187
	Basic IOB	134	70
	Extended IOBES	528	209
	Basic IOBES	139	72
BioNLP/ NLPBA	Extended IOB	865	179
	Basic IOB	226	72
	Extended IOBES	860	201
	Basic IOBES	217	79

Table 3: Number of iterations needed for the optimisation algorithm to converge.

baseline (which comes as no surprise given that the baseline system is a string match of entities found in the training set).

Table 3 shows the number of iterations[9] needed for the optimisation algorithm of the trainer to converge. The advantage of the L1 regularisation is apparent with nearly two to five times less iterations needed when compared to the L2 regularisation. Given the close F-scores achieved by the two family of setups, the L1 regularisation becomes a clear winner in our experimentation setup.

6 Conclusions

Argo's strength is manifested by its online availability, an intuitive graphical user interface available from a web browser, convenience in building even most complex text processing workflows, and the availability of trainable machine learning components. The Feature Generator component, customisable entirely through a GUI, provides the flexibility needed to extend the basic set of features without resorting to programming. The experiment results showed that an extended, yet generic, set of features can be taken to competitive levels in terms of effectiveness.

7 Acknowledgements

This work was partially supported by Biotechnology and Biological Sciences Research Council (BBSRC BB/G53025X/1 From Text to Pathways) and Korea Institute of Science and Technology Information (KISTI Text Mining and Pathways).

References

P. Corbett and P. Murray-Rust. 2006. High-throughput identification of chemistry in life science texts. *Comp Life*, pages 107–118. LNBI 4216.

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proc. of the 40th Anniversary Meeting of the Association for Computational Linguistics*.

D. Ferrucci and A. Lally. 2004. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*, 10(3-4):327–348.

Y. Kano, R. Dorado, L. McCrochon, S. Ananiadou, and J. Tsujii. 2010. U-Compare: An integrated language resource evaluation platform including a comprehensive UIMA resource library. In *Proc. of the Seventh International Conference on Language Resources and Evaluation (LREC 2010)*, pages 428–434.

J.-D. Kim, T. Ohta, Y. Tsuruoka, Y. Tateisi, and N. Collier. 2004. Introduction to the bio-entity recognition task at jnlpba. In *Proc. of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, JNLPBA '04, pages 70–75, Geneva, Switzerland. Association for Computational Linguistics.

B. Kolluru, S. Nakjang, R. P. Hirt, A. Wipat, and S. Ananiadou. 2011. Automatic extraction of microorganisms and their habitats from free text using text mining workflows. *Journal of Integrative Bioinformatics*, 8(2):184.

T. Kudo and Y. Matsumoto. 2001. Chunking with support vector machines. In *Proc. of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, NAACL '01, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

J. Lafferty, A. Mccallum, and F. Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA.

K. S. Tjong, F. Erik, and S. Buchholz. 2000. Introduction to the CoNLL-2000 shared task: chunking. In *Proc. of the 2nd workshop on Learning language in logic and the 4th Conference on Computational natural language learning*, pages 127–132, Morristown, NJ, USA. Association for Computational Linguistics.

[9] We do not report detailed CPU times due to experimenting on resource-shared machines. Such a setup makes direct side-by-side comparisons largely skewed. As a reference we note that the workflows completed in 15 minutes to about 11 hours depending on a feature space size and machine load.