OLLIE: On-Line Learning for Information Extraction

Valentin Tablan, Kalina Bontcheva, Diana Maynard, Hamish Cunningham
University of Sheffield
Regent Court, 211 Portobello St.
Sheffield S1 4DP, UK
{V.Tablan,diana,kalina,hamish}@dcs.shef.ac.uk

Abstract

This paper reports work aimed at developing an open, distributed learning environment, OLLIE, where researchers can experiment with different Machine Learning (ML) methods for Information Extraction. Once the required level of performance is reached, the ML algorithms can be used to speed up the manual annotation process. OLLIE uses a browser client while data storage and ML training is performed on servers. The different ML algorithms use a unified programming interface; the integration of new ones is straightforward.

1 Introduction

OLLIE is an on-line application for corpus annotation that harnesses the power of Machine Learning (ML) and Information Extraction (IE) in order to make the annotator's task easier and more efficient.

A normal OLLIE working session starts with the user uploading a set of documents, selecting which ML method to use from the several supplied by the system, choosing the parameters for the learning module and starting to annotate the texts. During the initial phase of the manual annotation process, the system learns in the background (i.e. on the server) from the user's actions and, when a certain degree of confidence is reached, it starts making suggestions by pre-annotating the documents. Initially, some of these suggestions may be erroneous but, as the user makes the necessary corrections, the system will learn from its mistakes and the performance will increase leading to a reduction in the amount of human input required.

The implementation is based on a client-server architecture where the client is any Java-enabled web browser and the server is responsible for storing data, training ML models and providing access services for the users.

The client side of OLLIE is implemented as a set of Java Server Pages (JSPs) and a small number of Java applets are used for tasks where the user interface capabilities provided by HTML are not enough.

The server side comprises a JSP/servlet server, a relational database server and an instance of the GATE architecture for language engineering which is used for driving all the language-related processing. The general architecture is presented in Figure 1.

The next section describes the client side of the OLLIE system while Section 3 details the implementation of the server with a subsection on the integration of Machine Learning. Section 4 talks about security; Section 6 about future improvements and Section 7 concludes the paper.

2 The OLLIE client

The OLLIE client consists of several web pages, each of them giving the user access to a particular service provided by the server.

One such page provides facilities for uploading documents in the system from a URL, a local file, or created from text pasted in a form. A variety of formats including XML, HTML, email and plain text are supported. When a document is created, the

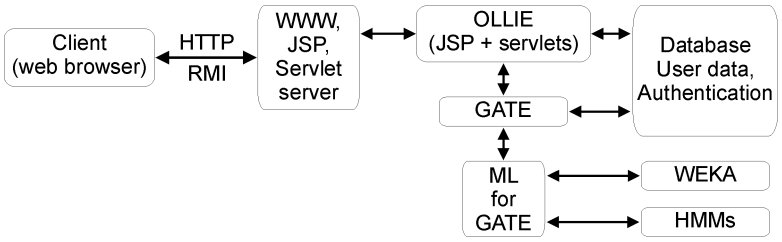

Figure 1: The general architecture of OLLIE

original markup –if present– is separated from textual content to prevent it from interfering with the subsequent language processing.

Another page lets the user choose which machine learning algorithm is to be used, the attributes that characterise the instances (e.g., orthography, part-of-speech), and parameters for the chosen learning method (e.g., thresholds, smoothing values). The classes to be learnt (e.g., Person, Organisation) are provided as part of the user profile, which can be edited on a dedicated page. All ML methods compatible with OLLIE have a uniform way of describing attributes and classes (see Section 3.1 for more details on the ML integration); this makes possible the use of a single user interface for all the ML algorithms available. The fine-tuning parameters are specific to each ML method and, although the ML methods can be run with their default parameters, as established by (Daelemans and Hoste, 2002), substantial variation in performance can be obtained by changing algorithm options.

Since OLLIE needs to support the users with the annotation process by learning in the background and suggesting annotations, it offers control over the accuracy threshold for these suggestions. This avoids annoying the users with wrong suggestions while assuring that suggestions the system is confident about are used to pre-annotate the data, reducing the workload of the user.

The document editor can then be used to annotate the text (see Figure 2). The right-hand side shows the classes of annotations (as specified in the user profile) and the user selects the text to be annotated (e.g., “McCarthy”) and clicks on the desired class (e.g., Person). The new annotation is added to the document and the server is updated immediately (so the new data becomes available to the ML algorithm too). The document editor also provides facilities for deleting wrong annotations, which are then propagated to the server, in a similar way.

The graphical interface facilities provided by a web browser could be used to design an interface for annotating documents but that would mean stretching them beyond their intended use and it is hard to believe that such an interface would rate very high on a usability scale. In order to provide a more ergonomic interface, OLLIE uses a Java applet that integrates seamlessly with the page displayed by the browser. Apart from better usability, this allows for greater range of options for the user.

The communication between the editor applet and the server is established using Java Remote Method Invocation (a protocol similar to the C++ Remote Procedure Call – RPC) which allows the instant notification when updates are needed for the document stored on the server. The continuous communication between the client and the server adds the benefit of data security in case of network failure. The data on the server always reflects the latest version of the document so no data loss can occur. The session data stored by the server expires automatically after an idle time of one hour. This releases the resources used on the server in case of persistent network failures.

The data structures used to store documents on the server are relatively large because of the numerous indices stored to allow efficient access to the annotations. The copy downloaded by the client when the annotation process is initiated is greatly reduced by filtering out all the unnecessary information. Most of the data transferred during the client-server communication is also compressed, which reduces the level of network traffic – always a problem in client server architectures that run over the Internet.

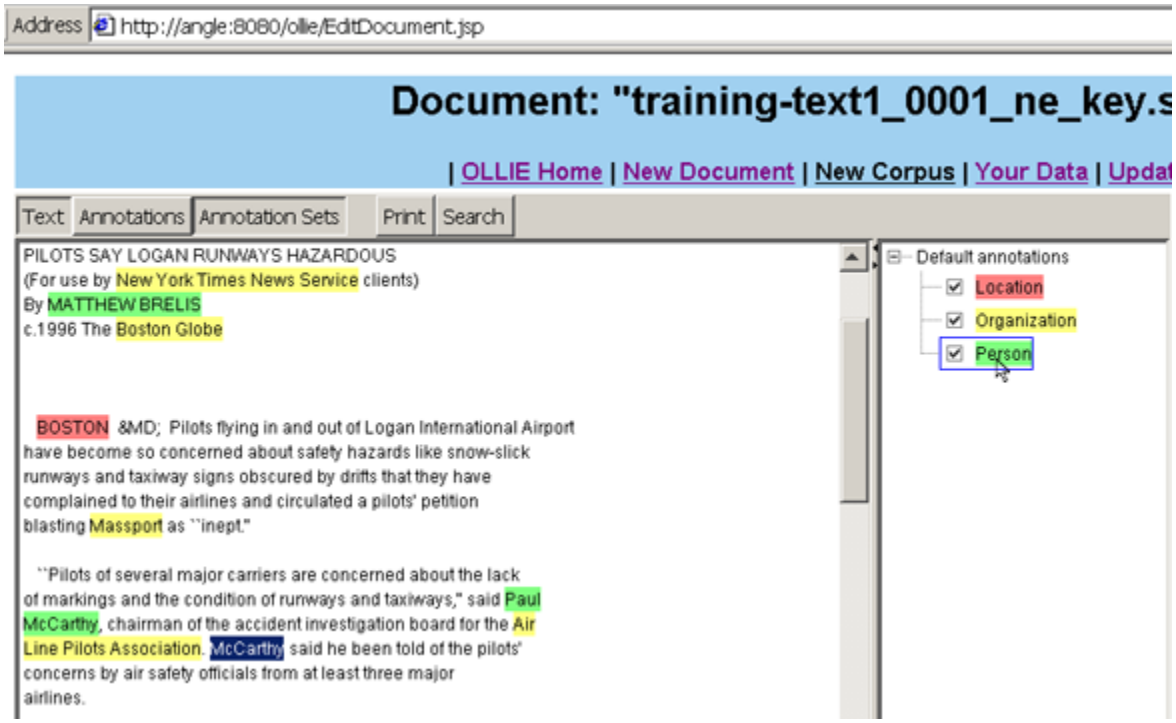

Figure 2: Annotating text in the OLLIE client

Another utility provided by the client is a page that lets the user specify the access rights to the document/corpus, which determine whether it can be shared for viewing or collaborative annotation (see Section 4 for details on security).

3 Implementation of the OLLIE server

While the client side of the OLLIE application is presented as set of web pages, the server part is based on the open source GATE architecture.

GATE is an infrastructure for developing and deploying software components that process human language (Cunningham et al., 2002). It is written in Java and exploits component-based software development, object orientation and mobile code. One quality of GATE is that it uses Unicode throughout (Tablan et al., 2002). Its Unicode capabilities have been tested on a variety of Slavic, Germanic, Romance, and Indic languages (Gambäck and Olsson, 2000; Baker et al., 2002). This allows OLLIE to handle documents in languages other than English. The back-end of OLLIE uses the GATE framework to provide language processing components, services for persistent storage of user data, security, and application management.

When a document is loaded in the OLLIE client and subsequently uploaded to the server, its format is analysed and converted into a GATE document which consists of textual content and one or more layers of annotation. The annotation format is a modified form of the TIPSTER format (Grishman, 1997), is largely isomorphic with the Atlas format (Bird and Liberman, 1999) and successfully supports I/O to/from XCES and TEI (Ide et al., 2000).[1] An annotation has a type, a pair of nodes pointing to positions inside the document content, and a set of attribute-values, encoding further linguistic information. Attributes are strings; values can be any Java object. An annotation layer is organised as a Directed Acyclic Graph on which the nodes are particular locations in the document content and the arcs are made out of annotations. All the markup contained in the original document is automatically extracted into a special annotation layer and can be used for processing or for exporting the document back to its original format.

[1]The American National Corpus is using GATE for a large TEI-based project.

Linguistic data (i.e., annotated documents and corpora) is stored in a database on the server (see Figure 1), in order to achieve optimal performance, concurrent data access, and persistence between working sessions.

One of the most important tasks for the OLLIE server is the execution and control of ML algorithms. In order to be able to use ML in OLLIE, a new processing resource was designed that adds ML support to GATE.

3.1 Machine Learning Support

Our implementation for ML uses classification algorithms for which annotations of a given type are instances while the attributes for them are collected from the context in which the instances occur in the documents.

Three types of attributes are defined: nominal, boolean and numeric. The nominal attributes can take a value from a specified set of possible values while the boolean and numeric ones have the usual definitions.

When collecting training data, all the annotations of the type specified as instances are listed, and for each of them, the set of attribute values is determined. All attribute values for an instance refer to characteristics of a particular instance annotation, which may be either the current instance or one situated at a specified relative position.

Boolean attributes refer to the presence (or absence) of a particular type of annotation overlapping at least partially with the required instance. *Nominal* and *numeric* attributes refer to features on a particular type of annotation that (partially) overlaps the instance in scope.

One of the boolean or nominal attributes is marked as the class attribute, and the values which that attribute can take are the labels for the classes to be learnt by the algorithm. Figure 3 depicts some types of attributes and the values they would take in a particular example. The boxes represent annotations, *Token* annotations are used as instances, the one in the centre being the *current* instance for which attribute values are being collected.

Since linguistic information, such as part-of-speech and gazetteer class, is often used as attributes for ML, OLLIE provides support for identifying a wide range of linguistic information - part-of-speech, sentence boundaries, gazetteer lists, and named entity class. This information, together with tokenisation information (kind, orthography, and token length) is obtained by using the language processing components available with GATE, as part of the ANNIE system (Cunningham et al., 2002). See Section 5 for more details on the types of linguistic features that can be used. The user chooses which of this information is to be used as attributes.

An ML implementation has two modes of functioning: training – when the model is being built, and application – when the built model is used to classify new instances. Our implementation consists of a GATE processing resource that handles both the training and application phases. It is responsible for detecting all the instances in a document and collecting the attribute values for them. The data thus obtained can then be forwarded to various external implementations of ML algorithms.

Depending on the type of the attribute that is marked as class, different actions will be performed when a classification occurs. For boolean attributes, a new annotation of the type specified in the attribute definition will be created. Nominal attributes trigger the addition of the feature specified in the attribute definition on an annotation of the required type situated at the position of the classified instance. If no such annotation is present, it will be created.

Once an ML model is built it can be stored as part of the user profile and reloaded for use at a later time.

The execution of the ML processing resource is controlled through configuration data that selects the type of annotation to be used as instances, defines all the attributes and selects which ML algorithm will be used and with what parameters.

One good source of implementations for many well-known ML algorithms is the WEKA library (Witten and Frank, 1999).[2] It also provides a wealth of tools for performance evaluation, testing, and attribute selection, which were used during the development process.

OLLIE uses the ML implementations provided by WEKA which is accessed through a simple wrapper that translates the requests from GATE into API calls "*understood*" by WEKA. The main types of requests dealt with by the wrapper are the setting of

[2]WEKA homepage: http://www.cs.waikato.ac.nz/ml/weka/

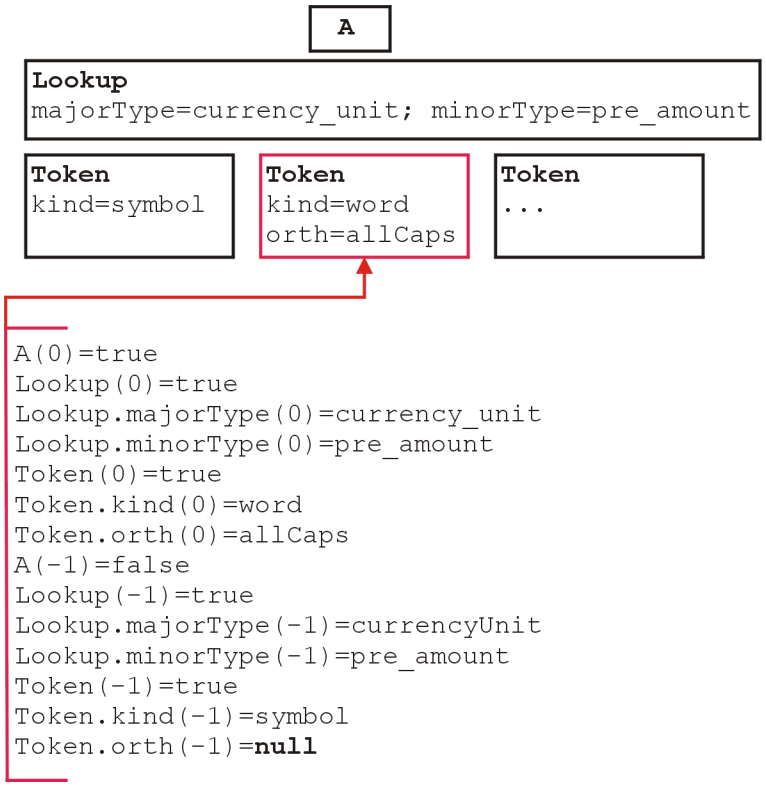

Figure 3: Example of attributes and their values.

configuration data, the addition of a new training instance and the classification of an application-time instance.

4 Security

Because OLLIE is deployed as a web application — accessible by anyone with Internet access, security is an important issue. Users store documents on the server and the system also keeps some personal data about the users for practical reasons.[3] All users need to be provided with a mechanism to authenticate themselves to the system and they need to be able to select who else, apart from them, will be able to see or modify the data they store on the server.

Every user has a username and a password, used to retrieve their profiles and determine which documents they can access. The profiles also contain information specifying the types of annotations that they will be creating during the annotation process. For example, in the case of a basic named entity recognition task, the profile will specify Person, Organisation, and Location. These tags will then be provided in the document annotation pages.

The access rights for the documents are handled by GATE which implements a security model similar to that used in Unix file systems. Table 1 shows the combination of rights that are possible. They give a good granularity for access rights, ranging from private to world readable.

The set of known users is shared between GATE and OLLIE and, once a user is authenticated with the system, the login details are kept in session data which is stored on the OLLIE server. This allows for automatic logins to the underlying GATE platform and transparent management of GATE sessions.

5 ML Experiments and Evaluation

To the end of evaluating the suitability of the ML algorithms provided by WEKA for use in OLLIE we performed several experiments for named entity recognition on the MUC-7 corpus (SAIC, 1998). We concentrated on the recognition of ENAMEX enti-

[3] Storing email addresses for instance is useful for sending password reminders.

	User		User's Group		Other Users	
Mode	**Read**	**Write**	**Read**	**Write**	**Read**	**Write**
"World Read/Group Write"	+	+	+	+	+	-
"Group Read/Group Write"	+	+	+	+	-	-
"Group Read/Owner Write"	+	+	+	-	-	-
"Owner Read/Owner Write"	+	+	-	-	-	-

Table 1: Security model — the access rights

ties, i.e., Person, Organisation, and Location. The MUC-7 corpus contains 1880 Organisation (46%), 1324 Location (32%), and 887 Person (22%) annotations in 100 documents. The task has two elements: recognition of the entity boundaries and classification of the entities in the three classes. The results are summarised below.

We first tested the ability of the learners to identify correctly the boundaries of named entities. Using 10-fold cross-validation on the MUC 7 corpus described above, we experimented with different machine learning algorithms and parameters (using WEKA), and using different attributes for training.

5 different algorithms have been evaluated: Zero R and OneR – as baselines, Naive Bayes, IBK (an implementation of K Nearest Neighbour) and J48 (an implementation of a C4.5 decision tree).

As expected, the baseline algorithms performed very poorly (at around 1%). For IBK small windows gave low results, while large windows were very inefficient. The best results (*f-measure* of around 60%) were achieved using the J48 algorithm.

The types of linguistic data used for the attribute collection included part of speech information, orthography (upper case, lower case, initial upper case letter, mixture of upper and lower case), token kind (word, symbol, punctuation or number), sentence boundary, the presence of certain known names and keywords from the gazetteer lists provided by the ANNIE system. Tokens were used as instance annotations and, for each token, the window used for collecting the attributes was of size 5 (itself plus two other tokens in each direction).

Additional information, such as features on a wider window of tokens, tended to improve the recall marginally, but decreased the precision substantially, resulting in a lower F-measure, and therefore the trade off was not worthwhile.

We also tested the algorithms on a smaller news corpus (which contained around 68,000 instances as opposed to 300,000 for the MUC7 corpus). Again, the J48 algorithm scored highest, with the decision table and the K nearest neighbour algorithms both scoring approximately 1 percentage point lower than the J48.

The second set of experiments was to classify the named entities identified into the three ENAMEX categories: Organisations, Persons and Locations. Using 10-fold cross-validation on the MUC 7 corpus described above, we experimented with the WEKA machine learning algorithms and parameters, and using attributes for training similar to those used for boundary detection. The best results were achieved again with the J48 algorithm, and, for this easier task, they were situated at around 90%. The attributes were chosen on the basis of their information gain, calculated using WEKA's attribute selection facilities.

The named entity recognition experiments were performed mainly to evaluate the WEKA ML algorithms on datasets of different sizes, ranging from small to fairly large ones (300,000 instances). The different ML algorithms had different memory requirements and execution speed, tested on a PIII 1.5GHz PC running Windows 2000 with 1GB RAM. From all algorithms tested, the decision table and decision tree were the slowest (325 and 122 seconds respectively on 68,000 instances) and required most memory - up to 800MB on the big datasets. Naive Bayes was very fast (only 0.25 seconds) with 1R following closely (0.28 seconds).

6 Further Work

OLLIE is very much work-in-progress and there are several possible improvements we are considering.

When dealing with a particular corpus, it is rea-

sonable to assume that the documents may be quite similar in terms of subject, genre or style. Because of that, it is possible that the quality of the user experience can be improved by simply using a list of positive and negative examples. This would allow the system not to make the same mistakes by always missing a particular example or always annotating a false positive – which can be very annoying for the user.

The big difference in execution time for different ML algorithms shows that there are practical advantages that can be gained from having more than one ML algorithm integrated in OLLIE, when it comes to supporting the user with the annotation task. Since the two experiments showed that Naive Bayes performs only slightly worse than the best, but slower algorithms, it may be feasible to train both a fast Naive Bayes classifier and a slower, but more precise one. In this way, while the slower ML algorithm is being re-trained on the latest data, OLLIE can choose between using the older model of this algorithm or the newly re-trained faster baseline, depending on which ones gives better results, and suggest annotations for the current document. As with other such environments, this performance is measured with respect to the latest annotated document.

We hope to be able to integrate more learning methods, e.g., TiMBL (Daelemans et al., 1998) and we will also provide support for other people willing to integrate theirs and make them available from our OLLIE server or run their own server.

We plan to experiment with other NLP tasks, e.g, relation extraction, coreference resolution and text planning for language generation.

Finally, we are working on a Web service implementation of OLLIE for other distributed, Grid and e-science applications.

7 Conclusion

OLLIE is an advanced collaborative annotation environment, which allows users to share and annotate distributed corpora, supported by adaptive information extraction that trains in the background and provides suggestions.

The option of sharing access to documents with other users gives several users the possibility to engage in collaborative annotation of documents. For example, one user can annotate a text with organisations, then another annotate it with locations. Because the documents reside on the shared server one user can see errors or questionable markup introduced by another user and initiate a discussion. Such collaborative annotation is useful in the wider context of creating and sharing language resources (Ma et al., 2002).

A number of Machine Learning approaches for Information Extraction have been developed recently, e.g., (Collins, 2002; Bikel et al., 1999), including some that use active learning, e.g., (Thompson et al., 1999) or offer automated support, e.g, (Ciravegna et al., 2002), in order to lower the overhead of annotating training data. While there exist corpora used for comparative evaluation, (e.g., MUC or the CMU seminar corpus), there is no easy way to test those ML algorithms on other data, evaluate their portability to new domains, or experiment with different parameters of the models. While some of the algorithms are available for experimentation, they are implemented in different languages, require different data formats, and run on different platforms. All of this makes it hard to ensure experimental repeatability and eliminate site-specific skew effects. Also, since not all systems are freely available, we propose an open, distributed environment where researchers can experiment with different learning methods on their own data.

Another advantage of OLLIE is that it defines a simple API (Application Programming Interface) which is used by the different ML algorithms to access the training data (see Section 3.1). Therefore, the integration of a new machine learning algorithm in OLLIE amounts to providing a wrapper that implements this API (a straightforward process). We have already provided a wrapper for the ML algorithms provided by the WEKA toolkit which can be used as an example.

Although OLLIE shares features with other adaptive IE environments (e.g., (Ciravegna et al., 2002)) and collaborative annotation tools (e.g., (Ma et al., 2002)), it combines them in a unique fashion. In addition, OLLIE is the only adaptive IE system that allows users to choose which ML approach they want to use and to comparatively evaluate different approaches.

References

[Baker et al.2002] P. Baker, A. Hardie, T. McEnery, H. Cunningham, and R. Gaizauskas. 2002. EMILLE, A 67-Million Word Corpus of Indic Languages: Data Collection, Mark-up and Harmonisation. In *Proceedings of 3rd Language Resources and Evaluation Conference (LREC'2002)*, pages 819–825.

[Bikel et al.1999] D. Bikel, R. Schwartz, and R.M. Weischedel. 1999. An Algorithm that Learns What's in a Name. *Machine Learning, Special Issue on Natural Language Learning*, 34(1-3), Feb.

[Bird and Liberman1999] S. Bird and M. Liberman. 1999. A Formal Framework for Linguistic Annotation. Technical Report MS-CIS-99-01, Department of Computer and Information Science, University of Pennsylvania. `http://xxx.lanl.gov/-abs/cs.CL/9903003`.

[Ciravegna et al.2002] F. Ciravegna, A. Dingli, D. Petrelli, and Y. Wilks. 2002. User-System Cooperation in Document Annotation Based on Information Extraction. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, pages 122–137, Siguenza, Spain.

[Collins2002] M. Collins. 2002. Ranking algorithms for named entity extraction: Boosting and the voted perceptron. In *Proceedings of the 40th Annual Annual Meeting of the Association for Computational Linguistics (ACL'02)*, Philadelphia,PA.

[Cunningham et al.2002] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*.

[Daelemans and Hoste2002] Walter Daelemans and Véronique Hoste. 2002. Evaluation of Machine Learning Methods for Natural Language Processing Tasks. In *LREC 2002 Third International Conference on Language Resources and Evaluation*, pages 755–760, CNTS Language Technology Group, University of Antwerp, UIA, Universiteitsplein 1 (bldng A), B-2610 Antwerpen, Belgium.

[Daelemans et al.1998] W. Daelemans, J. Zavrel, K. van der Sloot, and A. van den Bosch. 1998. Timbl: Tilburg memory based learner version 1.0. Technical Report Technical Report 98-03, ILK.

[Gambäck and Olsson2000] B. Gambäck and F. Olsson. 2000. Experiences of Language Engineering Algorithm Reuse. In *Second International Conference on Language Resources and Evaluation (LREC)*, pages 155–160, Athens, Greece.

[Grishman1997] R. Grishman. 1997. TIPSTER Architecture Design Document Version 2.3. Technical report, DARPA. `http://www.itl.nist.gov/-div894/894.02/related_projects/tipster/`.

[Ide et al.2000] N. Ide, P. Bonhomme, and L. Romary. 2000. XCES: An XML-based Standard for Linguistic Corpora. In *Proceedings of the Second International Language Resources and Evaluation Conference (LREC)*, pages 825–830, Athens, Greece.

[Ma et al.2002] X. Ma, H. Lee, S. Bird, and K. Maeda. 2002. Models and tools for collaborative annotation. In *Proceedings of 3rd Language Resources and Evaluation Conference (LREC'2002)*, Gran Canaria, Spain.

[SAIC1998] SAIC. 1998. Proceedings of the Seventh Message Understanding Conference (MUC-7). `http://www.itl.nist.gov/iaui/894.02/-related_projects/muc/index.html`.

[Tablan et al.2002] V. Tablan, C. Ursu, K. Bontcheva, H. Cunningham, D. Maynard, O. Hamza, Tony McEnery, Paul Baker, and Mark Leisher. 2002. A unicode-based environment for creation and use of language resources. In *Proceedings of 3rd Language Resources and Evaluation Conference*.

[Thompson et al.1999] C. A. Thompson, M. E. Califf, and R. J. Mooney. 1999. Active learning for natural language parsing and information extraction. In *Proceedings of the International Conference on Machine Learning*, pages 406–414.

[Witten and Frank1999] I. H. Witten and E. Frank. 1999. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann.