Creating Custom Taggers by Integrating Web Page Annotation and Machine Learning

Srikrishna Raamadhurai* **Oskar Kohonen*** **Teemu Ruokolainen****
*Aalto University, Department of Information and Computer Science, Finland
**Aalto University, Department of Signal Processing and Acoustics, Finland
firstname.lastname@aalto.fi

Abstract

We present an on-going work on a software package that integrates discriminative machine learning with the open source WebAnnotator system of Tannier (2012). The WebAnnotator system allows users to annotate web pages within their browser with custom tag sets. Meanwhile, we integrate the WebAnnotator system with a machine learning package which enables automatic tagging of new web pages. We hope the software evolves into a useful information extraction tool for motivated hobbyists who have domain expertise on their task of interest but lack machine learning or programming knowledge. This paper presents the system architecture, including the WebAnnotator-based front-end and the machine learning component. The system is available under an open source license.

1 Introduction

A typical development cycle of a natural language processing (NLP) tool involves several different experts whose time is often limited as well as expensive. In particular, rule-based systems need experts to construct the rules, while data-driven systems require domain experts to produce annotated training data and machine learning experts to train the systems. Because of the required investment, tasks which lack commercial or academic interest are often left completely without applicable tools. Nevertheless, we believe that there exist many relatively simple tasks where necessary annotation for a machine learning system could be produced by motivated hobbyists who possess domain expertise but lack machine learning or programming knowledge. For example, consider identifying fields in classified ads such as product name, dimensions and price, or segmenting individual posts in a web forum. To this end, we present a software package that integrates discriminative machine learning with the open source WebAnnotator system (Tannier, 2012).

The combination of an annotation tool and machine learning is, of course, not a new idea, as it goes back at least to the Alembic system (Day et al., 1997), which was developed to accelerate the process of tailoring NLP tools to new domains, languages, and tasks, by attempting to reduce the work load of human annotators by employing pre-taggers learned from previously annotated data. Despite these ideas being around for a long time, they do not seem to have been integrated into a web-browser previously. Since a large amount of information is consumed using the web-browser, it is desirable to be able to train and apply automatic analysis tools directly within that context.

The paper is organized as follows. In Section 2, we review how the system is used, its general architecture and details related to how the machine learning is implemented. Section 4 provides discussion and conclusions on the work.

2 System

In this section we present in some detail the system that integrates discriminative machine learning with the open source WebAnnotator (Tannier, 2012). We review the usage of the system, its software architecture, the central aspects related to how machine learning is applied: the employed Conditional

Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations, pages 15–19, Dublin, Ireland, August 23-29 2014.

Random Fields-method, indexing and pre-processing web-pages, how training sets are constructed, and the applied feature extraction. Finally, we review how the trained system is applied to new web-pages.

The latest version of the software can be found at `https://github.com/okohonen/semantify`.

2.1 Overview of Usage

The system is installed as an add-on to the Firefox browser. Subsequently, it can be activated for any web page. The user can train several different models by indicating which model a particular annotation belongs to. The user can also define the tag set used for annotation. To annotate, the user highlights parts of the page with the mouse and selects the desired tag from the shown menu. Assigned annotations are denoted by colors. This process is presented in Figure 1 (a). When the user is done, she stores the annotated page as shown in Figure 1 (b). The system then stores the annotated page in its page index. Meanwhile, in the background, the system automatically produces a training set that contains all the pages annotated so far and learns a tagger. The user can ask the system to tag a new page as shown in Figure 1 (c), in which case the system pre-processes and tags the current page using the latest trained model. The system then adds the annotations matching the output of the machine learning model to the viewed web page. The automatically assigned tags are visually distinct from the manually annotated ones (lighter color scheme). The automatic taggings can then be corrected manually and added to the training set. An example of an automatically annotated page is shown in figure 1 (d).

2.2 Overview of Architecture

The system architecture consists of two main components, namely, 1) an add-on to the Firefox-browser that allows annotation of web pages directly in the browser window, and 2) a machine learning component for annotating new pages. The browser add-on extends the WebAnnotator system (Tannier, 2012) by integrating it with the machine learning component and with functionality to show and edit the tags produced by the trained taggers. The machine learning component indexes the annotated web pages, pre-processes them to produce training sets of sentences, and trains models that can then be applied to new data. The Firefox add-on is implemented in Javascript and XUL while the machine learning component is implemented in Python. To bridge the language gap they communicate using XMLHTTPRequest. The machine learning component implements the well-known conditional random field (CRF) method, a discriminative modeling framework for sequence labeling (Lafferty et al., 2001).

2.3 Conditional Random Fields

Our system implements the linear-chain CRF model (Lafferty et al., 2001) which can inherently accommodate the arbitrary, overlapping features described below in Section 2.7. The CRF model is estimated based on the available training set of exemplar input-output sequence pairs. Test instances are decoded using the standard Viterbi search (Lafferty et al., 2001).

CRF parameter estimation is most commonly associated with the maximum likelihood approach employed by Lafferty et al.(2001). However, in our system, we rely on the averaged perceptron algorithm following Collins (2002a). (Note that the CRFs correspond to discriminatively trained hidden Markov models, and Collins (2002a) employs the latter terminology.) We apply the averaged perceptron learning approach for its simplicity and competitive performance (Zhang and Clark, 2011).

2.4 Web Page Index

The web page index in the machine learning component stores the annotated pages using the internal format of WebAnnotator, which is simply the original HTML-page augmented with -tags to encode and visualize the annotations. Apart from annotated pages, it is also possible to index unannotated pages if one has a particular set of pages that are to be tagged by the model.

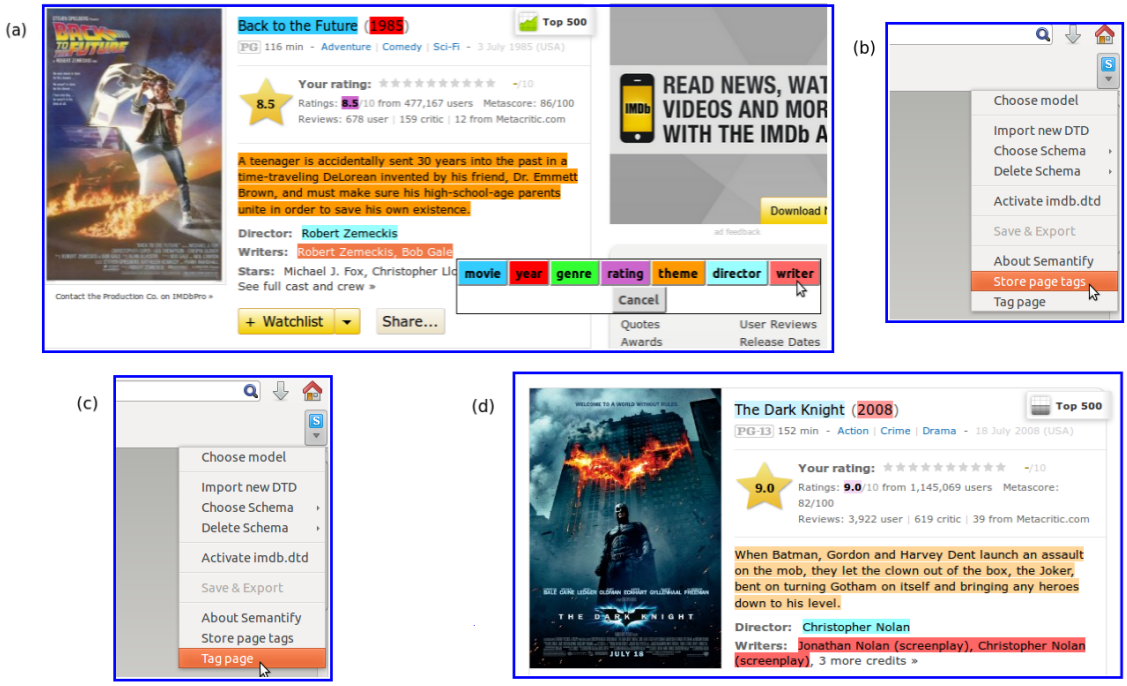

Figure 1: Sample screenshots of the tool depicting typical scenarios.

2.5 Pre-processing

We parse the HTML using the Beautiful Soup-library[1], extracting the visible text parts. Subsequently, we tokenize the text by splitting at white space and at non-alphanumeric characters. The token sequence is grouped into sentences based on punctuation and HTML-tags. We consider that if an HTML-tag defines a new block then it also starts a new sentence (e.g. starts a block, while does not). For each token position we apply feature functions that are described in detail in Section 2.7.

The user's annotations are stored as `span`-tags that are identified by their `class` attributes. The `span`-tags are parsed so that the label information is extracted, but they are ignored when calculating the feature functions which should be identical regardless of how the page is annotated. If the user has not assigned any label we assume a default `Outside` class.

2.6 Building the Training Sets

The CRF parameter estimation requires training and development sets which must be drawn from annotated web pages in the page index. A special characteristic of the data set is that the label distribution is typically very skewed towards many `Outside` taggings and few non-`Outside` taggings. To ensure that the development set gets sufficiently many non-`Outside` taggings, we use use a modulus-based scheme that assigns 10% of the sentences to the development set while making sure that there are enough sentences containing taggings from each class. The training set and development set are formed by concatenating the preprocessed files for each individual page.

2.7 Feature Extraction

In addition to standard first-order state transition features, the CRF model includes feature functions which operate on the token sequence of the sentence and the HTML-tree of the page. We use *orthographic features* and *HTML-features* which consider the token string and HTML-tree, respectively. For orthographic features, we use the word in lower case, and generalized character-based following Collins (2002b). We extract features based on the following properties of the current node in the HTML-tree: parent nodes and the class-attribute. For the parent nodes we calculate both individual features for im-

[1] http://www.crummy.com/software/BeautifulSoup/

mediate parents as well as very specific features that concatenates all parents up to the `body`-tag. For the class-attribute we provide it both as it is and apply the same generalization functions as in the orthographic feature set. One could also extract features from other attributes than `class`. However, we suspect that they are less informative for the tagging task compared to the `class`-attribute which is often used to indicate structural properties of the page content. We also window the features to consider the previous and next positions in the input.

2.8 Annotating New Pages Automatically

When the user asks the system to tag a new page, the current page is sent to the machine learning component for preprocessing. The latest trained CRF is applied to the preprocessed page and each token is assigned a tag. We produce -tags similar to the internal format used by WebAnnotator, but with distinct attributes so the browser extension can distinguish manual and automatic annotation. To produce the modified HTML-page, we need to know the position in the HTML-string for each token in the preproessed file. In order to achieve this, we create an index of the HTML-string during preprocessing that maps every token to its original position in the string.

3 Discussion

For a software tool of the presented kind, the key performance measures are: accuracy on the task at hand, as a function of the number of annotations; and training time. As both measures are specific to task and implementation, addressing them both experimentally would require a large number of experiments which is not feasible in the allowed space. However, for illustration purposes, we will present a simple example application.

In general, for the proposed system to yield high accuracy, it needs to learn the desired categories from a few annotated examples. This requires input features that predict the desired target category well. The key benefit of the discriminative training employed is the ability to use a large set of rich and overlapping features. This allows the construction of feature sets that yield good performance on several different tasks, reducing the need for task-specific feature engineering which requires domain and programming expertise. Future work includes identifying additional features that yield good performance in a number of different tasks.

For training time, it would be ideal if the system could be trained in real time, that is, once the user has submitted an annotated web page to the system, the training would be completed when the user wants to apply the model to a new web page. This would require training times on the order of a few seconds. Training time depends mostly on the employed classifier, training algorithm, and the number of sequences and tokens in the training set. We had assumed that training time would not be an issue, even for a naive implementation, because a typical user would only gather small data sets. However, it turned out that while the annotation may be small in the number of annotated web pages, typical web pages were larger in terms of token counts than we had anticipated and more advanced training techniques may be needed to reach real-time performance.

To illustrate the properties related to tagging web pages using the current implementation, we present a simple example application of the system to the task of extracting fields from the Internet Movie Database (IMDB).[2] We annotated 50 web pages from IMDB, each describing a different movie. The following fields were annotated: *director*, *genre*, *title*, *rating*, *theme*, *writer*, and *release year*. It should be noted that this task is from the easier end of the spectrum, as the fields can be extracted with a high accuracy using the markup structure alone. For experimental purposes, we performed cross-validation with training, development, and test sets constructed from 36, 4, and 10 web pages respectively. The system yielded the following token F-scores by category, director: 73%, genre: 99%, title: 86%, rating: 100%, theme: 100%, writer: 80%, and release year: 100%. This level of performance is promising, as several fields were extracted with perfect, or near perfect accuracy. The training times for the 36-page training sets varied between 1.5 and 3.5 minutes on a standard desktop computer (Intel i5-2500 with 16Gb RAM). The variance in training time is explained by the employed stopping criterion which

[2] http://www.imdb.com

terminates training based on the performance on the development set, resulting in varying numbers of passes over the training data. In the above example task, the training set sizes were on average: 22K sequences, 133K tokens, and 30K features.

The empirical training times are longer than what could be considered real-time performance. The goal of the current implementation has been accuracy rather than execution time, and for the latter, there is certainly room for improvement. However, for real-time performance, the improvement needed is large enough that a different training procedure may be necessary. A promising approach, that suits the setting well, is using online training, such that one would only train on the latest submitted web-page. Furthermore, it is usually the case, that the non-`Outside` annotation is concentrated on a fairly small subpart of the web page. This structure could be utilized to reduce computational cost. These approaches will be evaluated in future work.

4 Conclusions

We presented on-going work on a software package that integrates discriminative machine learning with the open source WebAnnotator system of Tannier (2012). The system allows users to annotate web pages directly within their web browser and train a machine learning tagger applicable for annotating new web pages. We hope the software evolves into a useful information extraction tool for motivated hobbyists who have domain expertise on their task of interest but lack machine learning or programming knowledge.

In future work, we plan to investigate the following aspects of the system. The utility of the system should be evaluated in real-life tasks and for the target user group. The machine learning component could then be improved further based on user experience. Perhaps most importantly, the extracted features can be improved using both generic and task-specific features. Also, while the system currently applies only supervised learning, it would be a natural setting to apply semi-supervised learning.

Acknowledgements

The work was funded by an Exploratory Research Project grant from Aalto Science Institute, the Academy of Finland research project on Multimodal Language Technology, Langnet (Finnish doctoral programme in language studies), and the Academy of Finland under the Finnish Centre of Excellence Program 2012–2017 (grant no. 251170).

References

Michael Collins. 2002a. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, volume 10, pages 1–8.

Michael Collins. 2002b. Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 489–496. Association for Computational Linguistics.

David Day, John Aberdeen, Lynette Hirschman, Robyn Kozierok, Patricia Robinson, and Marc Vilain. 1997. Mixed-initiative development of language processing systems. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, ANLC '97, pages 348–355, Stroudsburg, PA, USA. Association for Computational Linguistics.

John Lafferty, Andrew McCallum, and Fernando C.N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289.

Xavier Tannier. 2012. WebAnnotator, an Annotation Tool for Web Pages. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012)*, Istanbul, Turkey, May.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.