KrdWrd
Architecture for Unified Processing of Web Content

Johannes Steger
Neurbiopsychology Group
Institute of Cognitive Science
University of Osnabrück
jsteger@acm.org

Egon Stemle[*]
Computational Linguistics Group
Institute of Cognitive Science
University of Osnabrück
estemle@uos.de

Abstract

Algorithmic processing of Web content mostly works on textual contents, neglecting visual information. Annotation tools largely share this deficit as well.

We specify requirements for an architecture to overcome both problems and propose an implementation, the KrdWrd system. It uses the Gecko rendering engine for both annotation and feature extraction, providing unified data access in every processing step. Stable data storage and collaboration control scripts for group annotations of massive corpora are provided via a Web interface coupled with a HTTP proxy. A modular interface allows for linguistic and visual data feature extractor plugins.

The implementation is suitable for many tasks in the *Web as corpus* domain and beyond.

1 Introduction

Working with algorithms that rely on user-annotated Web content suffers from two major deficits:

For annotators, the presentation of Web sites in the context of annotation tools usually does not match their everyday Web experience. The lack or degeneration of non-textual context may negatively affect the annotators' performance and the learning requirements of special annotation tools may make it harder to find and motivate annotators in the first place.

Feature extraction performed on annotated Web pages, on the other hand, leaves much of the information encoded in the page unused, mainly those concerned with rendering.

In this paper, we present the design (2) and implementation (3) of the KrdWrd architecture that addresses these two issues. Section 4 contains a proof of concept in the context of CleanEval, i.e. the cleaning arbitrary web pages, and Section 5 concludes with an outlook on the possible applications and implementation improvements.

2 Design

2.1 Design Goals

We aim to provide an architecture for Web data processing based on the unified treatment of data representation and access on both the annotation and the processing side. This includes an application for users to annotate a corpus of Web pages by classifying continuous text elements and a back-end application that processes those user annotations and extracts features from Web pages for further automatic processing.

2.2 Requirements

Flexibility The system should be open enough to allow customization of every part but also, specifically provide stable interfaces for more common tasks to allow for modularization.

Stability We need a stable HTTP data source that is independent of the original Website, including any dependencies such as images, style-sheets or scripts.

Automaticity Back-end processing should run without requiring any kind of human interaction.

Replicability Computations carried out on Web page representations must be replicable across systems, including any user-side processing.

Quantity Corpus size should not influence the performance of the system and total process-

[*]Now at CIMeC, University of Trento, 38068 Rovereto.

ing time should scale linearly with the corpus.

Usability Acquisition of manually classified corpora requires a fair amount of contributions by users annotating pages. Achieving a high level of usability for the end-user therefore is paramount. As a guideline we should stay as close as possible to the everyday Web experience. We also need to provide tools for learning how to use the annotation tool and how to annotate Web pages.

2.3 Core Architecture

To address these requirements, we developed an abstract architecture, a simplified version of which is depicted in Figure 1. We outline the rationale for the basic design decisions below.

For rendering a Web page, an object tree is constructed from its HyperText Markup Language (HTML) source code. This tree can be traversed and its nodes inspected, modified, deleted and created through an API specified by the World Wide Web Consortium's (W3C) Document Object Model (DOM) Standard (Hors et al., 2004). Its most popular use case is client-side dynamic manipulation of Web pages, for visual effects and interactivity. This is most commonly done by accessing the DOM through a JavaScript interpreter. Essentially, a page's DOM tree gives access to all the information we set out to work on: structure, textual content and visual rendering data. Therefore, it serves as the sole interface between application and data.

While all browsers try to implement some part of the DOM standard (currently, Version 3 is only partially implemented in most popular browsers), they vary greatly in their level of compliance as well as their ability to cope with non-standard compliant content. This leads to structural and visual differences between different browsers rendering the same Web page.

Therefore, to guarantee *replicability*, we require the same DOM engine to be used through the envisioned system.

To reach a maximal level of *automaticity* and not to limit the *quantity* of the data, it is important that data analysis takes place in a parallel fashion and does not require any kind of graphical interface, so it can e.g. be executed on server farms. On the other hand we also need to be able to present pages within a browser to allow for user annotation. Consequently, the same DOM engine needs to power a browser as well as a headless back-end application, with *usability* being an important factor in the choice of a particular browser.

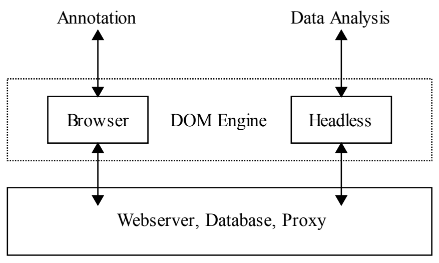


Figure 1: Basic KrdWrd Architecture: both users annotating corpus pages through their Web browser and back-end applications working on the data run the same DOM engine. The central server delivers and stores annotation data and coordinates user submissions.

The annotation process, especially the sequence of presentation of pages, is controlled by a central Web server – users cannot influence the pages they are served for annotation. Thereby any number of concurrently active users can be coordinated in their efforts and submissions distributed equally across corpus pages. All data, pristine and annotated, is stored in a database attached to the Web server. This setup allows the architecture to scale *automatically* with user numbers under any usage pattern and with reasonable submission *quantities*.

Stability of data sources is a major problem when dealing with Web data. As we work on Web pages and the elements contained in them, simple HTML dumping is not an option – all applications claiming to offer full rewriting of in-line elements fail in one way ore another, especially on more dynamic Web sites. Instead, we use a HTTP proxy to cache Web data used in our own storage. By setting the server to grab content only upon first request and providing an option to turn off download of new data, we can create a closed system that does not change once populated.

3 Implementation

We maintain the implementation in a source code repository at `http://krdwrd.org`. The documentation includes pointers to the required external software.

This section will first describe the DOM engine and its use by browser and back-end application (3.1), then the details of the implementation of central storage and control (3.2), and will end with listing possible feature extractors for the back-end (3.3).

3.1 DOM Engine

The choice of DOM engine is central to the implementation. We reviewed all major engines available today with respect to the requirements listed in 2:

The KDE Project's KHTML drives the Konquerer browser and some more exotic ones, but lacks a generic multi-platform build process.

This practical limitation is lifted by Apple's fork of KHTML, called WebKit. It is the underlying engine of Safari browsers on Mac OS X and Windows. There also exists a Qt and a GTK based open source implementation. Whereas they are quite immature at the moment and not very widely used, this will change in the future and WebKit will certainly become a valuable option at some point.

Whereas the open source variant of Google's browser, *Chromium*, promises superior execution speed by coupling WebKit with its own V8 JavaScript engine, it suffers from the same problem as WebKit itself namely, not being stable enough to serve as reliable platform – the Linux client for example is barely usable, a Mac client does not even exist, yet.

We also briefly evaluated Presto (Opera) and Trident (Microsoft), but discarded them due to their proprietary nature and lack of suitable APIs.

The Gecko engine (Mozilla Corporation), in conjunction with its JavaScript implementation Spidermonkey, marks a special case: It implements XUL (Goodger et al., 2001), the XML User Interface Language, as a way to create feature rich cross-platform applications. The most prominent of those is the Firefox browser, but also e.g. Thunderbird, Sunbird and Flock are built with XUL. An add-on system is provided that allows extending the functionality of XUL applications to third-party code, which gains full access to the DOM representation, including the XUL part itself. The proposed KrdWrd back-end can be implemented in the same manner as Firefox: provide custom JavaScript and XUL code on top of Mozilla's core XUL Runner. Code can easily be shared between a browser add-on and XUL applications and unsupervised operation is trivial to implement in a XUL program.

Given the synergy attainable in the XUL approach and Firefox' popularity amongst users, it was a simple decision to go with Mozilla Gecko for the core DOM implementation. We note that WebKit's rise and fast pace of development might change that picture in the future.

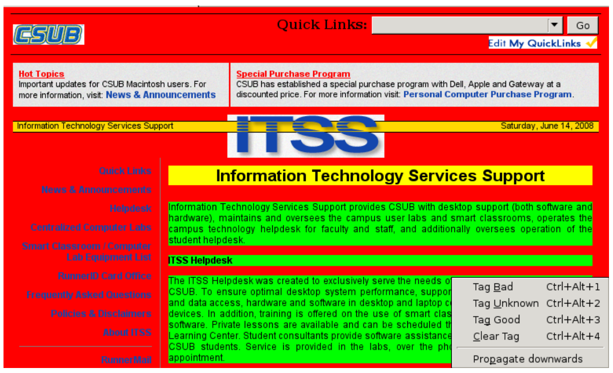
3.1.1 Firefox Add-on

Interactive visual annotation of corpus pages via Web browser is realized by the KrdWrd Firefox Add-on. The imposed annotation *base data* (Müller and Strube, 2003) are text elements in the DOM tree, which are non-overlapping word-, phrase-, and character-level strings, i.e. we do not superimpose a different structure. [1] The annotation then, is non-hierarchical, i.e. a single node can only be classified into one class at a time, and continuous, i.e. a class can only be assigned to one node at a time.

To facilitate adoption, it comes with a comprehensive user manual and an interactive tutorial (see below in 3.2.1). For easy setup, Firefox's proxy configuration is automatically pointed to a preconfigured host, respective credentials are auto-added to the password manager and the user is directed to a special landing page upon successful installation. The proxy feature also serves as a nice example of code shared between add-on and application. Furthermore, the installation binary is digitally signed, so the user does not have to go through various exception dialogs.

Once installed, the functionality of the Add-on is available via a broom icon in the status bar. Whereas it offers several functions centered around annotation and corpus selection, its core feature is simple: In highlight mode (the broom turns fuchsia) the mouse hovering over the page will highlight the text blocks below the cursor. The block can then be annotated using the context-menu or a keyboard short-cut, which will change its color to the one corresponding to the annotation class. Figure 2 shows a fully annotated page and the context-menu.

[1]However, while grabbing documents we surround text nodes of running text with additional <KW>-Elements: this delimits large amounts of text under a single node in the DOM tree, i.e. when the whole text could only be selected as a whole, these elements loosen this restriction but, on the other hand, do not affect the rendering of the Web page or other processing steps.



Quick Links: Go
Edit My QuickLinks

Hot Topics
Important updates for CSUB Macintosh users. For more information, visit: **News & Announcements**

Special Purchase Program
CSUB has established a special purchase program with Dell, Apple and Gateway at a discounted price. For more information visit: **Personal Computer Purchase Program**.

Information Technology Services Support Saturday, June 14, 2008

Quick Links
News & Announcements
Helpdesk
Centralized Computer Labs
Smart Classroom / Computer Lab Equipment List
RunnerID Card Office
Frequently Asked Questions
Policies & Disclaimers
About ITSS
RunnerMail

Information Technology Services Support

Information Technology Services Support provides CSUB with desktop support (both software and hardware), maintains and oversees the campus user labs and smart classrooms, operates the campus technology helpdesk for faculty and staff, and additionally oversees operation of the student helpdesk.

ITSS Helpdesk

The ITSS Helpdesk was created to exclusively serve the needs o
CSUB. To ensure optimal desktop system performance, suppor
and data access, hardware and software in desktop and laptop c
devices. In addition, training is offered on the use of smart clas
software. Private lessons are available and can be scheduled tl
Learning Center. Student consultants provide software assistance
CSUB students. Service is provided in the labs, over the pho
appointment.

Figure 2: Web pages can be annotated with the KrdWrd Firefox Add-on by hovering over the text by mouse and setting class labels by keyboard short-cut or pop-up menu.

3.1.2 XUL Application

The XUL application consists of a thin JavaScript layer on top of Mozilla's XUL Runner. It mainly uses the XUL browser control to load and render Web pages and hooks into its event handlers to catch completed page load events and the-like. Without greater C level patching, XUL still needs to create a window for all of its features to work. In server applications, we suggest using a virtual display such as Xvfb to fulfill this requirement.

During operation the application parses the given command-line arguments, which triggers the loading of supplied URLs (local or remote) in dedicated browser widgets. When the "load complete" event fires, one of several extraction routines is run and results are written back to disk. The implemented extraction routines are:

grab for simple HTML dumps and screen-shots,

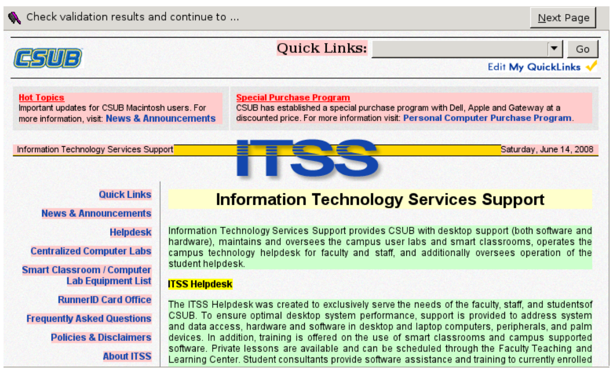
diff for computing a visual difference rendering of two annotation vectors for the same page,

merge for merging different annotations on the same Web page into one in a simple voting scheme, and

pipe for textual, structural and visual data for the feature pipelines.

3.2 Storage and Control

Central storage of Web pages and annotation data is provided by a database. Clients access it via CGI scripts executed by a Web server while the back-end uses python wrapper scripts for data exchange.



Quick Links: Go
Edit My QuickLinks

Hot Topics
Important updates for CSUB Macintosh users. For more information, visit: **News & Announcements**

Special Purchase Program
CSUB has established a special purchase program with Dell, Apple and Gateway at a discounted price. For more information visit: **Personal Computer Purchase Program**.

Information Technology Services Support Saturday, June 14, 2008

Quick Links
News & Announcements
Helpdesk
Centralized Computer Labs
Smart Classroom / Computer Lab Equipment List
RunnerID Card Office
Frequently Asked Questions
Policies & Disclaimers
About ITSS

Information Technology Services Support

Information Technology Services Support provides CSUB with desktop support (both software and hardware), maintains and oversees the campus user labs and smart classrooms, operates the campus technology helpdesk for faculty and staff, and additionally oversees operation of the student helpdesk.

ITSS Helpdesk

The ITSS Helpdesk was created to exclusively serve the needs of the faculty, staff, and studentsof CSUB. To ensure optimal desktop system performance, support is provided to address system and data access, hardware and software in desktop and laptop computers, peripherals, and palm devices. In addition, training is offered on the use of smart classrooms and campus supported software. Private lessons are available and can be scheduled through the Faculty Teaching and Learning Center. Student consultants provide software assistance and training to currently enrolled

Figure 3: During the tutorial, a Visual Diff between the user's submission and the sample data is presented right after submission. Here, the annotation from Figure 2 was wrong in tagging the sub-heading "ITSS Helpdesk": the correct annotation (*yellow*) is highlighted in the feedback in dark color – contrary to the heading "Information Technology Services Support" that was tagged correctly and hence, shows up in light color.

3.2.1 Web Server

Server-side logic is implemented by Python CGI scripts, thus any Web server capable of serving static files and executing CGI scripts is supported. Users can access the server directly by URL or via the Firefox Add-on menu. An overview page rendered by the server provides a submission overview as well as a detailed per-corpus submission list. In conjunction with the Add-on, server side scripts control serving of corpus pages by summing over submissions in the database and randomly selecting a page from those with the least total submission number. The Web server also delivers the actual HTML data to the client, whereas any embedded objects are served by the separate proxy server. Furthermore, it controls the tutorial: Users are presented with sample pages and asked to annotate them. Upon submission, a server side script compares the user's annotation with a reference annotation stored in the database and generates a page that highlights differences. The result is delivered back to the user's browser, as seen in Figure 3.

3.2.2 Database

The database mainly stores the raw HTML code of the corpus pages. User submissions are vectors of annotation classes, the same length as the number of text nodes in a page. In addition there is a user mapping table that links internal user ids to external authentication. Thereby user submissions are anonymized, yet trackable by id.

Given the simple structure of the database model, we choose to use zero-conf database back-end *sqlite*. This should scale up to some thousand corpus pages and users.

It is important to note that any database content must be pre-processed to be encoded in UTF-8 only. Unifying this bit of data representation at the very start is essential to avoid *encoding hell* later in the process. To this end, we rely on Mozilla's *Universal Charset Detector*[2], which is part of the Gecko engine, a mature *composite approach to language/encoding detection* (Li and Momoi, 2001) – the UTF-8 encoded output is fed into the database.

3.2.3 Proxy

Any object contained in the corpus pages needs to be stored and made available to viewers of the page without relying on the original Internet source.

Given an URL list, initial population of the proxy data can easily be achieved by running the XUL application in grabbing mode while letting the proxy fetch external data. Afterwards, it can be switched to block that access, essentially creating a closed system. We found WWWOffle to be a suitable proxy with support for those features while still being easy to setup and maintain.

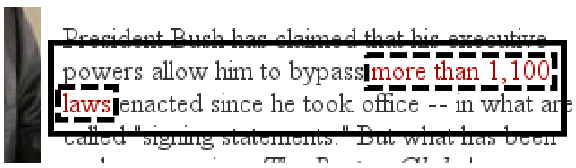
3.3 Feature Extractors

The XUL Application extracts information from corpus pages and dumps it into the file-system, to serve as input to specialized feature extractors. This implementation focuses on feature extraction on those nodes carrying textual content, providing one feature vector per such node. We therefore generate one feature vector per such node through a linguistic, visual and DOM-tree focused pipeline.

3.3.1 Text

For linguistic processing, the Application dumps raw text from the individual text nodes, with leading and trailing whitespace removed, converted to UTF-8 where applicable, i.e. the quirks of handling languages such as Chinese and Japanese, or even bi-directional languages like Hebrew are transparent to our processing and the subsequent

[2]`http://www.mozilla.org/projects/intl/detectorsrc.html`



President Bush has claimed that his executive
powers allow him to bypass more than 1,100
laws enacted since he took office -- in what are
called "signing statements." But what has been

Figure 4: Coordinates of a node's bounding box (straight) and text constituents (dotted) as provided to the visual processing pipeline.

applications need to be capable of handling these languages. External applications can read these data and write back the feature vector resulting from their computation in the same format.

For Computational Linguistic tools relying on phrase-level structured input, e.g. tokenizers, the Application can also dump raw text that more closely resembles the rendered output, i.e. paragraphs, spanning multiple nodes, are merged together and dumped in one line; each line – and hence, feature vector – is then duplicated as many times as nodes that are spanned.

3.3.2 Structural

During an Application run, a set of "DOM-Features" is directly generated and dumped as feature vector.

Choosing the right DOM properties and applying the right scaling is a non-trivial per-application decision. Our reference implementation includes features such as depth in the DOM-tree, number of neighboring nodes, ratio text characters to HTML code characters, and some generic document properties as number of links, images, embedded objects and anchors. We also provide a list of the types of node preceding the current node in the DOM-tree.

3.3.3 Visual

For visual analysis, the Application provides full-document screen-shots and coordinates of the bounding rectangles of all text nodes.[3] When text is not rendered in one straight line, multiple bounding boxes are provided as seen in Figure 4. This input can be processed by any application suitable for visual feature extraction.

For simple statistics dealing with the coordinates of the bounding boxes, we use a Python script to generate basic features such as total area

[3]This Extractor requires at least XUL Runner Version 1.9.2 (corresponding to Firefox Version > 3.5) which is still in beta at the time of this writing.

Table 1: BootCaT seed terms for *Canola* corpus

history	coffee	salt
spices	trade road	toll
metal	silk	patrician
pirate	goods	merchant

covered in pixel, number of text constituents, their variance in x-coordinates, average height and thelike.

4 Case Study

The current implementation comprises an extensive system for pre-processing and automated cleaning of Web pages, i.e. a typical Web-as-corpus task, where users are provided with accurate Web page presentations and annotation utilities in a typical browsing environment, while supervised machine learning algorithms also operate on representations of the visual rendering of Web pages.

The sequence of steps includes corpus creation and acquisition of hand-annotated training data on that corpus (4.1), feature extraction (4.2), training of a classifier and producing annotated test results (4.3).

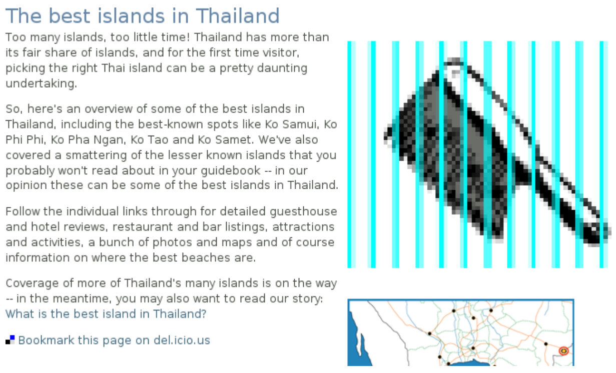
The underlying data, tools, and programs are bundled with the KrdWrd distribution as usage example.

4.1 Data Acquisition

Gathering a set of sample pages is the first step before utilizing people to tag new data. Therefore, we acquired a new corpus named *Canola* by using the BootCaT (Baroni and Bernardini, 2004) tool to produce a URL list from the seed terms in Table 1 using the Yahoo search engine.

To populate the proxy, we ran the Application on every URL once and also extracted the textual content of the pages. We then filtered for text lengths between 500 and 6,000 characters [4] and ran the Application once again, this time dumping the raw HTML code of the pages in UTF-8 format. During this second pass, the proxy is switched to block access to external sources. This ensures that no dynamic external content makes it into the corpus data, while letting innocent content pass. See Figure 5 for an example.

[4]…for Chinese these numbers had to be cut down to 50 and 600, however.



The best islands in Thailand

Too many islands, too little time! Thailand has more than its fair share of islands, and for the first time visitor, picking the right Thai island can be a pretty daunting undertaking.

So, here's an overview of some of the best islands in Thailand, including the best-known spots like Ko Samui, Ko Phi Phi, Ko Pha Ngan, Ko Tao and Ko Samet. We've also covered a smattering of the lesser known islands that you probably won't read about in your guidebook -- in our opinion these can be some of the best islands in Thailand.

Follow the individual links through for detailed guesthouse and hotel reviews, restaurant and bar listings, attractions and activities, a bunch of photos and maps and of course information on where the best beaches are.

Coverage of more of Thailand's many islands is on the way -- in the meantime, you may also want to read our story: What is the best island in Thailand?

Bookmark this page on del.icio.us

Figure 5: IFrames with dynamic URLs which usually come from advertisements are blocked as a nice side-effect of the Proxy setup.

The resulting HTML is post-processed to ensure that references and encodings are consistent: The head tag is expanded by a `<base href="`*`original url`*`" />` line, so a browser later viewing the dumped HTML will request embedded objects by their original URLs, which can then be served by the proxy. After removing any non-UTF-8 encoding hints, the data is fed into the database's page table, with a unique page id and the corpus id.

The pre-processed data is now ready to be processed by annotators. For gathering training data, students were asked to go through the ten Web page annotation tutorial once – to get acquainted with the annotation tool, i.e. the Add-on, and different aspects of how to apply the guidelines [5] to real-world Web pages – and then annotate pages from the *Canola* corpus as part of an homework assignment. The annotation process consisted of tagging text on Web pages with *three* tags 'good', 'bad', and 'uncertain'.

Over the course of two weeks, about 60 students provided a total average of 7.75 annotations per page. As the time data in Figure 6 suggests, users learn quickly; Average per-page annotation times drop well below three minutes after some training. The tutorial with its ten pages took on average 22 minutes to complete; note however, these pages were shortened and stripped down to illustrate particular aspects of Web pages.

Integration of the Add-on in users' environments was flawless and we did not receive any reports of usability or general handling problems.

[5]A refined version of the official 'CLEANEVAL: Guidelines for annotators' `http://cleaneval.sigwac.org.uk/annotation_guidelines.html` available at `https://krdwrd.org/manual/html`.

Manual inspection of submissions also did not show any anomalies, but to the contrary, indicated that submitters took great care to provide adequate annotations (c.f. Figure 7).

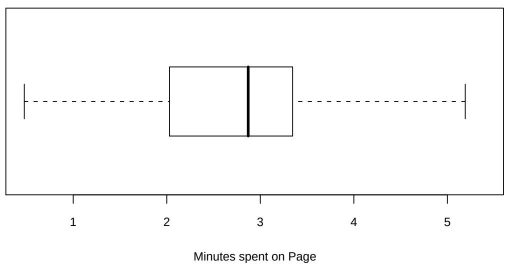


Figure 6: Time spent for annotation of a single Web page across all annotators of the *Canola* corpus.

The data obtained from user annotations was next merged into a single corpus using the Application's *merge* function (c.f. 3.1.2), resulting in a total of 216 corpus pages, each backed by up to 8 user submissions. Different treatment of JavaScript on the client side resulted in partial misalignment on some pages: dynamic client code had inserted or re-ordered nodes in some instance while not in others. We extended the merge procedure to accept some fuzziness in node matching, but still lost data from about 5% of submissions that could not be re-aligned. Until this problem is solved, we turn off JavaScript for Web content via the Firefox Add-On. Note that attaching unique IDs to text nodes is only a partial solution to this problem: A common JavaScript idiom is to clone an existing element and to populate it with new content, ultimately leading to different nodes with the same "unique" ID.

4.2 Extraction Pipeline

Feature Extraction commences by running the KrdWrd application extraction pipeline over the merged data obtained during annotation. For the *Canola* corpus' 216 pages, it took 2.5 seconds on average per page to generate text (2.5 million characters total), DOM information (46575 nodes total), screen-shots (avg. size 997x4652 pixels) and a file with the annotation target class for each text node.

We only used the stock KrdWrd features on the DOM tree and visual pipeline. For computing textual features, we borrowed Victor's (Spousta et al., 2008) text feature extractor.

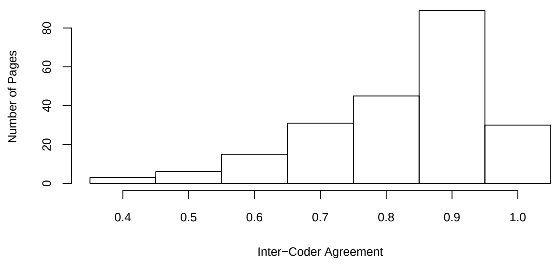


Figure 7: Fleiss's multi-π agreement (Artstein and Poesio, 2008) between submissions for pages over the *Canola* corpus.

4.3 Experiment

We used the data gathered by the feature extraction for training a Support Vector Machine (Chang and Lin, 2001). We used an RBF kernel with optimal parameters determined by a simple grid search to create ad-hoc models on a per-pipeline basis. The total number of feature vectors corresponded to the number of text nodes in the corpus and was 46575. Vector lengths for the different pipelines and test results from 10-fold cross validation are shown in Table 2.

Although the results for the single pipelines look quite promising – especially the surprisingly good performance of the visual pipeline given its limited input – combinations of feature sets in a single SVM model perform only marginally better. We therefore suggest running separate classifiers on the feature sets and only merging their results later, possibly in a weighted voting scheme. DOM features would certainly benefit most from e.g. a classifier that can work on structured data.

4.4 Inspecting Classifier Results

The classification results can be back-projected into the DOM-trees using the Application's *diff* function. As in the tutorial for annotators, it produces a visual diff, showing where the classifier failed. Note that these results are just Web pages, so they can be viewed anywhere without the help of the Add-on. This quickly turned out to be a valuable tool for evaluation of classification results.

Table 2: 10-fold cross validated classification test results for different combinations of the textual (cl), DOM-property based (dom) and visual (viz) pipelines on the *Canola* data set obtained using stock SVM regression with a RBF kernel.

Modules	Feat.	Acc.	Prec.	Recall
cl	21	86%	61%	76%
dom *	13	65%	64%	56%
viz *	8	86%	64%	82%
cl dom *	34	67%	74%	57%
dom viz *	21	67%	72%	59%
cl viz	29	86%	63%	78%
cl dom viz	42	68%	76%	58%

* data obtained by training on reduced number of input vectors.

5 Conclusion

Employing KrdWrd in the *Canola* case study showed that we achieved what we set out for and gave some valuable experience for possible improvements:

The KrdWrd Firefox Add-On is the first tool for Web page annotation that integrates flawlessly into a users daily browsing experience. It is unobtrusive and has a simple and intuitive user interface. Users quickly learn how to annotate and produce quite uniform results, given sufficient annotation guidelines.

The KrdWrd application and supporting infrastructure are a reliable platform under a real-world usage scenario. By decoding any input data to UTF-8 at the moment it enters the system and ensuring that we explicitly deliver UTF-8 exclusively throughout the system, we circumvented all usual encoding problems.

The overall handling of JavaScript is not satisfactory. To address the diversions between submits occurring after dynamic client-side JavaScript execution on different clients, the Add-on could hook into the node creation and clone processes. They could be suppressed entirely or newly created nodes could grow a special id tag to help identifying them later.

For result analysis, we would like to expand the visual diff generated from classification results. Showing results from separate runs on different subsets of the data or different parameters on one page would facilitate manual data inspection. Presenting selected feature values per node might also help in developing new feature extractors, especially in the DOM context.

Furthermore, we would like to integrate the JAMF framework (Steger et al., 2008), a component-based client/server system for building and simulating visual attention models, into the tool chain. This would allow for features based on the analysis of the rendered pages akin to how humans perceive these pages while browsing.

Summarizing, we designed and implemented an architecture for holistic treatment of Web pages in classification tasks. We demonstrated that the KrdWrd system can be used to automatically build an annotated corpus from user submissions. We also showed the broad set of features for text, structure and imagery it can help to extract, and how their contribution to classification can be assessed graphically.

References

Ron Artstein and Massimo Poesio. 2008. Inter-coder agreement for computational linguistics. *Computational Linguistics*, 34(4):555–596.

Marco Baroni and Silvia Bernardini. 2004. Bootcat: Bootstrapping corpora and terms from the web. Proceedings of LREC 2004.

Chih-Chung Chang and Chih-Jen Lin, 2001. *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Ben Goodger, Ian Hickson, David Hyatt, and Chris Waterson. 2001. Xml user interface language (xul) 1.0. Recommendation, Mozilla.org.

Arnaud Le Hors, Philippe Le Hgaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. 2004. Document object model (dom) level 3 core specification. Recommendation, W3C.

Shanjian Li and Katsuhiko Momoi. 2001. A composite approach to language/encoding detection. In *19th International Unicode Conference*.

Christoph Müller and Michael Strube. 2003. Multi-level annotation in mmax. In *Proc. of the 4th SIGDIAL*.

Miroslav Spousta, Michal Marek, and Pavel Pecina. 2008. Victor: the web-page cleaning tool. In *Proceedings of the 4th Web as Corpus Workshop (WAC4) – Can we beat Google?*

Johannes Steger, Niklas Wilming, Felix Wolfsteller, Nicolas Höning, and Peter König. 2008. The jamf attention modelling framework. In Lucas Paletta and John K. Tsotsos, editors, *WAPCV*, volume 5395 of *Lecture Notes in Computer Science*, pages 153–165. Springer.