Institute of Cognitive Science

Master's Thesis

Hybrid Sweeping: Streamlined Perceptual Structured-Text Refinement

Egon W. Stemle
egon.stemle@uos.de

March 20, 2009

Supervisors:

Stefan Evert
Computational Linguistics Group

Peter König
Neurobiopsychology Group

Institute of Cognitive Science
University of Osnabrück
Germany

Abstract

This thesis discusses the KrdWrd Project.

The Project goals are to *provide tools and infrastructure for acquisition, visual annotation, merging and storage of Web pages as parts of bigger corpora*, and to *develop a classification engine that learns to automatically annotate pages, operate on the visual rendering of pages, and provide visual tools for inspection of results*.

Attributions

The KrdWrd Project is a joint development with Johannes Steger.

I am the single author of chapters 1, 3, and 5.

Johannes Steger is the main author of a joint paper we will submit to the 5th Web as Corpus workshop hosted by the Association for Computational Linguistics Special Interest Group in San Sebastian, Spain, on 7 September, 2009; chapter 2 (fully) and 4 (mostly) are included from this paper.

I designed, implemented, and carried out the data analyses on the gathered user data. I set-up and maintained the general infrastructure of the project. I wrote the homework assignment and presented it in class. I co-authored the manual for the KrdWrd Add-on (the main authors are Maria Cieschinger and Kilian Klimek). I also co-authored the refinement of the annotation guidelines (the main authors are Maria Cieschinger and Kilian Klimek).

Contents

1 Introduction

This thesis presents the KrdWrd Project [@KRDW].

The KrdWrd Project deals with the design of an abstract architecture for A) the unified treatment of Web data for automatic processing, *without* neglecting visual information, on annotation and processing side and B) the appropriate annotation tool to gather data for supervised processing of such data.

The Project comprises an implementation appropriate for pre-processing and cleaning of Web pages, where users are provided with accurate Web page presentations and annotation utilities in a typical browsing environment, while machine learning algorithms also operate on representations of the visual rendering of Web pages. The system also preserves the original Web documents and all the additional information contained therein to make different approaches comparable on identical data.

1.1 Motivation

The Field of Statistical Natural Language Processing (NLP) has developed an amazing set of utilities, tools, and applications – but they all depend on and benefit from substantial amounts of electronically readable, pre-processed text, referred to as corpora, for their statistical language models. These corpora should be *representative* of the type of text they cover, that is domain, genre, and style of the language should be controllable, they should be easily accessible and freely available, only little work should suffice to obtain the desired and needed quantities of running text, and for the purpose of monitoring language development the collection process should be synchronous or dynamic. [KG03, KB06, FNKdS07]

Unfortunately, it has proven very difficult to obtain large quantities of 'traditional' text that is not overly restricted by authorship or publishing companies and their terms of use, or other forms of intellectual property rights[1], is versatile – and controllable – enough in type, and hence, suitable for various scientific use-cases. [Kil07, @SZAG, BNC]

The growth of the World Wide Web as an information resource has been providing an alternative to large corpora of news feeds, newspaper texts, books, and other electronic versions of classic printed matters: The idea arose to gather data from the Web for it is an unprecedented and virtually inexhaustible source of authentic natural language data and offers the NLP community an opportunity to train statistical models on much larger amounts of data than was previously possible.[GN00, DW05, Eve08]

However, we observe that after crawling content from the Web the subsequent steps, namely, language identification, tokenising, lemmatising, part-of-speech tagging, indexing, etc. suffer from *'large and messy' training corpora [...] and interesting [...] regularities may easily be lost among the countless duplicates, index and directory pages, Web spam, open or disguised advertising, and boilerplate [BDD+07].*

[1] For Web corpora it is believed that they are a special case of the caches and indexes used by search engines, that is to say that copyright infringement complaints can always be relayed to Google and others.

The consequence is that thorough pre-processing and cleaning of Web corpora is crucial in order to obtain reliable frequency data. The dimension of this task calls for an automated solution, the broadness of the problem for supervised machine learning approaches.

1.2 Relation to Recent Work

CleanEval is a shared task and competitive evaluation on the topic of cleaning arbitrary Web pages. In 2007 the first exercise took place and brought together language technology research and development in the field: even though the organisers had not imagined that Machine Learning (ML) methods were suitable for this task, at the event several systems did use them. The Participants also used heuristic rules – but many different ML methods were seen. The language models were mainly language independent, e.g. average length of a sentence or number or characters in a sentence, ratio of punctuation marks and other character classes, etc.

The methods may be equally well suited for the cleaning task – and another CleanEval competition will be held where the enhanced successors of these systems will compete – but the context of the task needs modifications. We will formulate the prevailing criticism here but note that the organisers have already acknowledged much of the criticism [BCKS08]:

The Annotation Guidelines arguably left some space for cases in which annotators felt the guidelines to be insufficient.

The Annotation Set-up consisted of two windows open, one showing the page as rendered by a browser, and the other showing a pre-cleaned version of the page, in a plain-text editor; however, at least two participating systems found that using e.g. a Web browser for annotation significantly improves the annotation speed compared to this method. [BDD+07, SMP08]

The Output Document Format consisted of one plain-text document per cleaned Web page, had all boilerplate removed, and simple mark-up added; however, this simple format implied that the link between the original Web page and the retained material was lost and that no structural information was available, that no explicit statement about what was considered boilerplate was left in the data, and it turned out that inserting the mark-up was more problematic than removing boilerplate. [BCKS08]

The Size of the Development Set was set small – for the mentioned reasons; too small for ML methods to be applied. Furthermore, having the same page only annotated by two people is certainly good for 'the quantity of data' but not for the quality.

The Scoring Metrics was the Levenshtein edit distance applied to words instead of tokens and without the substitution operator, i.e. the distance between two strings was given by the minimum number of insertions and deletions of single words needed to transform one into the other. This was problematic for two reasons: firstly, the algorithm was slow to run and secondly, the measure does not compensate for the fact that a single wrong choice may lead to many – in consequence – misaligned words.

Remark: we are well aware of page segmentation algorithms that carry the term *visual* in their name, most noticeably [RHJRWY04] but consider them to be *visual* in the sense of our structural DOM pipeline, i.e. they do *not* treat visual appearance in our sense where visual feature maps can be computed and thus, successful attention models.

1.3 Structure of the Work

This thesis follows the KrdWrd Project throughout its development: We start in 2 by describing the design goals and principles underlying the abstract KrdWrd architecture and its implementation. We continue in 3 and 4 where we present a specific instantiation of the architecture, which comprises an extensive system for automated Web cleaning tasks, and show how it was put into use. We conclude in 5 with a summary and an outlook on further development.

The appendix contains additional material that was authored or co-authored by me, and links to other relevant resources available online.

2 The KrdWrd Architecture – A DOM-Tree Based, Modular Content-Extraction Framework for Web Pages

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

1. 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.

2. 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.

We will now present the design (2.1) and implementation (2.2) of the KrdWrd Architecture that addresses these two issues.

2.1 Design

2.1.1 Design Goals

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

2.1.2 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 pages' representation must be replicable across systems, including any user-side processing.

Quantity Corpus size should not influence the performance of the system and total processing time should grow linearly with the corpus.

Usability Acquisition of manually classified corpora requires a fair amount of contributions by users annotating the pages. Achieving a high level of usability for the end-user therefore is of paramount importance. 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.

2.1.3 Core Architecture

To address these requirements, we developed an abstract architecture, a simplified version of which is depicted in figure 2.1. We will 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 [HHW+04]. It's 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 allows access to all the information we set out to work on: structure, textual content and visual rendering data. We therefore make it 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 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.

The annotation process, especially the order 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 a option to turn off download of new data, we can create a closed system that does not change anymore once populated.

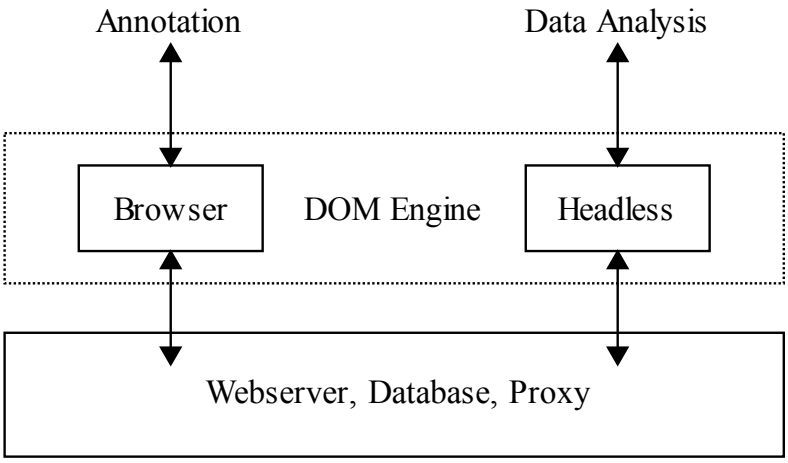


Figure 2.1: Basic KrdWrd Architecture: both users annotating corpus pages through their Web browser and back-end applications working on the data run the same DOM engine. The central server delivers and stores annotation data and coordinates user submissions.

2.2 Implementation

2.2.1 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 2.1:

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 WepKit 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 it's own V8 JavaScript engine, it suffers from the same problem as WebKit itself of not being stable enough yet to serve as reliable platform - The Linux client for example is barely usable, a Mac client does not exist yet.

We also briefly checked on 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 [GHHW01], 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

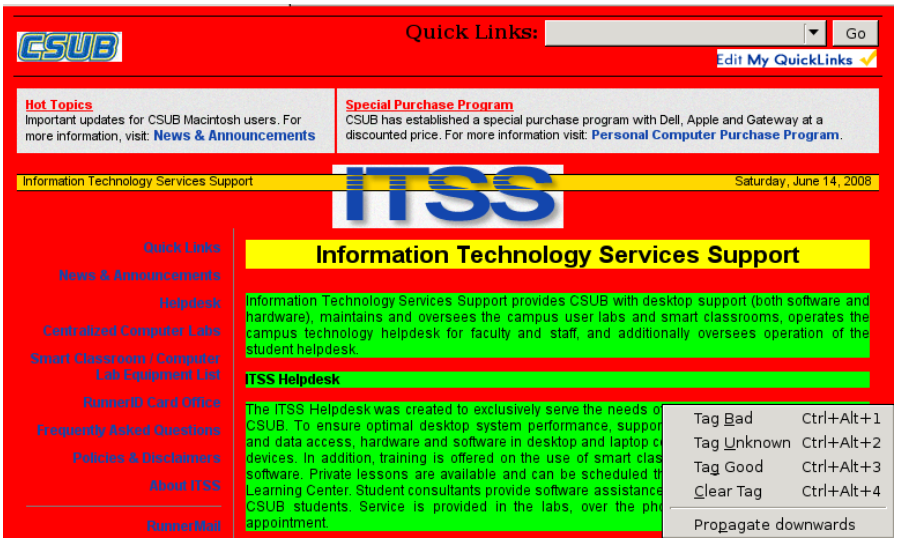


Figure 2.2: Web pages can be annotated with the KrdWrd Add-on by hovering over the text by mouse and setting class labels by keyboard short-cut or pop-up menu

third-party code that 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.

2.2.1.1 Firefox Add-on

Interactive visual annotation of corpus pages via Web browser is realized by the KrdWrd Add-on. To facilitate adoption, it comes with a comprehensive user manual and an interactive tutorial (see below in 2.2.2.1). For easy setup, Firefox' 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 lots of 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 2.2 shows a fully annotated page and the context-menu.

2.2.1.2 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 fulfil this requirement.

In operation the application parses the command-line given, 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 written back to disk. 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,

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

2.2.2 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.

2.2.2.1 Web Server

Server-side logic is implemented by Python CGI scripts, so 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 2.3.

2.2.2.2 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.

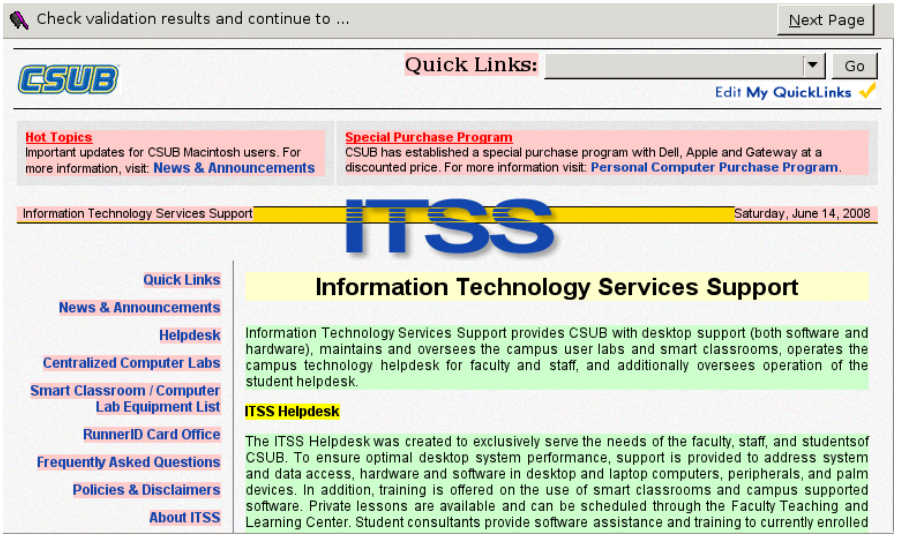


Figure 2.3: During the tutorial, a Visual Diff between the user's submission and the sample data is presented right after submission. Here, the annotation from 2.2 was wrong in tagging the sub-heading "ITSS Helpdesk": the correct annotation (*yellow*) is highlighted in the feedback.

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.

2.2.2.3 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.

2.2.3 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.

2.2.3.1 Text

For linguistic processing, the Application dumps raw text from the individual nodes, with leading and trailing whitespace removed, converted to UTF-8 where applicable. External applica-

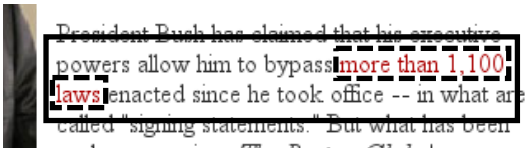


Figure 2.4: Coordinates of a node's bounding box (straight) and text constituents (dotted) as provided to the visual processing pipeline.

tions can read these data and write back the feature vector resulting from their computation in the same format.

2.2.3.2 Structural

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

Choosing the right DOM properties and apply 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.

2.2.3.3 Visual

For visual analysis, the Application provides full-document screen-shots and coordinates of the bounding rectangles of all text nodes.[1] When text is not rendered in one straight line, multiple bounding boxes are provided as seen in figure 2.4. 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 simple Python script to generate basic features such as total area covered in pixel, number of text constituents, their variance in x-coordinates, average height and the-like.

Furthermore, we provide a tool chain to use the JAMF framework [SWW+08], a component-based client/server system for building and simulating visual attention models. The "CoordRect" JAMF component generates masks the same size as the Web page screen-shot based on the node coordinates. It therefore allows region-wise analysis of the page rendering with the default component set provided by JAMF, which is focused on visual feature extraction. Results are read by the JAMF Python client and converted into feature vectors on a per-node basis.

Clearly, the components and filter of JAMF model employed or using an entirely different framework for actually visual feature extraction are per-application decisions to be made.

[1] 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

3 The KrdWrd Annotation Framework – Gathering Training Data for Sweeping Web Pages

The KrdWrd System is an implementation of the architecture we presented in 2: It comprises an extensive system for automated Web cleaning tasks.

For training the KrdWrd ML Engine, a substantial amount of hand-annotated data, viz. Web pages, are needed. Following, we present the parts of the system that cover the acquisition of training data, i.e. the steps before training data can be fed into a ML Engine.

Hence, after an overview of the sequence of steps needed to gather new training data in 3.1, an in-depth description of the processing steps *before* Web pages can be presented to annotators in 3.2, presentation of the actual tool annotators use in 3.3, and the compilation of their submitted results in 3.4, we will be ready to feed the KrdWrd Gold Standard to a ML Engine. An exemplification, the KrdWrd ML Engine, is covered in 4.

3.1 System Overview

Two fundamental ideas behind this part of the system are: firstly, Web pages have a textual representation, namely the text they contain, a structural representation, namely their DOM tree, and a visual representation, namely their rendered view – all representations should be considered when automatically cleaning Web pages, and consequently, all should be annotated during acquisition of training data for ML tasks. Secondly, data acquisition for training of supervised ML algorithms should preserve pristine, unmodified versions of Web pages – this will help to reproduce results *and* to compare those of different architectures.

3.1.1 Functional Walk-Through

Gathering a set of sample pages is at the beginning of tagging new data. The process needs to be coordinated by the administrators of the system, i.e. server level access is needed to make new corpora available for later tagging by users. The Process starts with a list of seed terms which are used to construct an ad-hoc corpus of Web pages where the result is a list of Uniform Resource Locators (URL[1]).

The URL list is then *harvested*, i.e. the according Web pages are downloaded and saved for further processing. This process is coordinated by the administrators of the system and is started as automated batch-job on the server where its input is the URL List and the result is the set of downloaded Web pages and their content.

These Web Pages are then available online to users for tagging, i.e. there are no constraints on who is able to access these pages; however, keeping track of *who tagged what* requires to differentiate between users, and hence, registration with the system, viz. logging in. The Web

[1] see [@URL] for details – but also [@ADDR].

pages are accessible via the KrdWrd Add-on in combination[2] with the Web Services hosted on [@KRDW, Web Site].

Users can tag new, alter or redisplay formerly tagged Web pages with the help of the KrdWrd Add-on. The KrdWrd Add-on builds upon and extends the functionality of the Firefox [@FF] browser and facilitates the visual tagging of Web pages, i.e. users are provided with an accurate Web page presentation and annotation utility in a typical browsing environment. Readily (or partly) tagged pages are directly sent back to the server for storage in the KrdWrd Corpus data pool and for further processing.

Updated or newly submitted tagging results are regularly merged, i.e. submitted results from different users for the same content are processed and compiled into a majority-driven uniform view. This automated process uses a *winner takes all strategy* and runs regularly on the server – without further ado. The *merged* content is stored in the KrdWrd data pool and hence, available for browsing, viewing, and analysis by the KrdWrd Add-on[2] and furthermore, it can be used as training data for Machine Learning algorithms.

3.1.2 Implementation Survey

The KrdWrd Infrastructure consists of several components that bring along the overall functionality of the system. They are run either on the KrdWrd Server or are part of the KrdWrd Add-on and hence, build upon and extend the functionality of the Firefox browser. The Server components are hosted on a Debian GNU/Linux [@DEB] powered machine. However, the requirements[3] are rather limited and many other standard linux - or linux-like - systems should easily suffice, and even other platforms should be able to host the system. Nevertheless, the KrdWrd Add-on strictly runs only as an extension of the Firefox browser, version 3[4].

Access to the system is given as HTTP Service hosted on `krdwrd.org`, an SSL-certified virtual host running on an Apache Web Server [@HTTP] accompanied by mailing services, a dedicated trac as Wiki and issue tracking system for software development (extended with a mailing extension), and subversion [@SVN] as version control system. The interfacing between the KrdWrd Add-on and the Web Server is done via CGI [@CGI] scripts, which itself are mostly written in the Python programming language [@PYTH].

3.2 Pre-Processing: Harvesting Web Pages

Generally, pre-processing is the first step to streamline external data for further processing in a customised data-processing pipeline, and in the KrdWrd System it constitutes harvesting data, i.e. grab Web pages off the web, convert them into UTF-8 encoding [@UNIC], make links on these pages relative [@W3ba], and compile them into a corpus that can be tagged by users.

3.2.1 URL List Generation

For downloading pages off the Web the KrdWrd System needs to be told *which* pages to grab. But because we are interested in a wide spectrum of layouts we need to scatter the URLs to be fetched over different web sites, i.e. we are not interested in having a small number of site

[2] Indeed, the data is accessible with *any* browser – but the KrdWrd Add-on enhances the experience.
[3] These include sed, awk, python, bash, subversion, XULRunner, wwwoffle, apache, R.
[4] But it could be converted into a self-contained XULRunner application.

URLs and then recursively grab these sites but we want a large number of URLs from different sites.

To this end we utilise the BootCaT toolkit[BB04] to construct an ad-hoc URL list: a set of seed terms is used for automated queries against a Web search engine, the top results for querying random combinations of the terms are downloaded and analysed, i.e. unigram term counts from all retrieved Web pages are compared with the corresponding counts from a reference corpus. In the last step[5] multi-word terms are extracted and used a seed terms for the query process. However, we used the top results from these last multi-word queries as URL List.

en détail: We used the BootCaT installation of the Institute of Cognitive Science's Computational Linguistics group at the University of Osnabrück[@CL] – at the time of writing this was the initial version with the updates from February 2007.

The topic of the seed terms for the BootCaT procedure was "Nuremberg in the Middle Ages", the terms were: *history, coffee, salt, spices, trade road, toll, metal, silk, patrician, pirate, goods, merchant*, the Internet search engine BootCaT used was Yahoo![@YAHO], the reference corpus was the British National Corpus (BNC)[6], and the procedure resulted in 658 URLs from unique domains; note that we departed from the original BootCaT recipe and only allowed one URL per domain. This URL list was passed on to the KrdWrd Harvester – but, of course, *any* URL list can be feed to the Harvester.

The seed terms, the command sequence, and the URL list can be found at `https://krdwrd.org/trac/browser/tags/harvest/canola`.

3.2.2 The KrdWrd App: Harvesting Mode

The automated downloading of Web content is done by the KrdWrd App in harvesting mode, namely by feeding the App an URL list as input and have it then fetch and store downloaded content for further processing. Moreover this process resolves three significant concerns:

Enforce UTF-8 Character Encoding for grabbed documents – character encoding has been the cause for much hassle in data processing, and to eliminate it – or at least reduce it to a minimum – we transform *every* document into UTF-8 encoding [@UNIC] and make sure that successive processing steps are UTF-8 aware.

Change the <BASE> Element for grabbed documents (or insert one) [@W3ba, @ADDR] – for smooth integration into the KrdWrd system we change this attribute such that relative URIs are resolved relative to our system.

Surround Text with <KW> Elements in grabbed documents – these additional elements splits up text: when large amounts of text fall under a single node in the DOM tree, i.e. when the whole text can 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.

[5] Though, this loop can be repeated multiple times with unigram term counts until the corpus of retrieved Web pages reaches a certain size or matches other characteristics.

[6] The data was obtained under the terms of the BNC End User Licence. For information and licensing conditions relating to the BNC, please see the web site at `http://www.natcorp.ox.ac.uk`

Finally, the System extracts the textual content of each page and only considers documents of certain text length as appropriate for further processing and discards all others. The rational is that *very* short and *very* long web pages rarely contain useful samples of interesting running text.

en détail: We used the previously generated URL list and fed it to the KrdWrd App in harvesting mode, which then retrieved Web pages via the KrdWrd Proxy (see 3.2.3) just as if someone operating a Firefox browser had viewed them. The textual length restriction was set to only allow for a *decent* amount of text, which we thought holds for documents consisting of 500 to 6,000 words[7]. Finally, we manually inspected the remaining grabbed pages for problems arising from limitations – and had to discard two files. Overall, the process resulted in 228 pages that were considered for further processing.

The currently used 'harvester' can be found at `https://krdwrd.org/trac/browser/trunk/src/app/harvest.sh`.

3.2.3 The KrdWrd Proxy

The KrdWrd Harvester and the KrdWrd Add-on make all Internet connections through the KrdWrd Proxy. This storage fills up with the harvested Web pages but also with all directly-linked material, which is included via absolute or relative links, or e.g. *generated* by scripts. Often, this 'additional' material is considered superfluous and therefore discarded; moreover, the non-textual content of Web pages is often stripped off – or the textual or structural content altered. See e.g. [@POTA, @CEan], or more generally [KB06, FNKdS07, EKS08].

Unfortunately, this renders it very difficult – or even impossible – to compare work in cases where one utilises data that is not available any more or only in an altered form. This is to say indeed, in the end we *also* want text but with different requirements of competing systems the base material must be pristine, i.e. the most 'natural' and the least modified version of data should be conserved. To this end, we utilise the World Wide Web Offline Explorer (wwwoffle)[@WOFF] as a proxy, which can be operated in two modes: *online* and *offline*.

wwwoffle Online Mode allows for caching of pages that are downloaded for later review, use with one or more external proxies, control over which pages cannot be accessed and which pages are not to be stored in the cache.

wwwoffle Offline Mode allows for use of normal browser to follow links, control which pages can be requested, and for non-cached access to Intranet servers.

wwwoffle generally allows for a searchable cache index with the addition of included programs, and viewable indexes sorted by name, date, server domain name, type of file. The configuration is done in a single configuration file, which can be accessed via an interactive web page to allow editing; user customisable error and information pages are also easily configurable.

During pre-processing the KrdWrd Online Proxy is used; it runs as a daemon and responds

[7] We used the linux `wc`[@WC] command, i.e. a word is a string of characters delimited by white space characters.

only to internal requests but material that is downloaded in online mode will be available for requests in offline mode.

The KrdWrd Offline Proxy runs as a daemon and responds to network requests from the Internet, it is publicly available[8], and can be accessed via proxy.krdwrd.org:8080. This proxy does not fetch new pages into the KrdWrd Cache, i.e. all Web page request coming from the client computer, e.g. from a user surfing the net with an installed and enabled KrdWrd Add-on, will be filtered and only requests for content that had previously been downloaded in online mode will be allowed. The offline mode is automatically configured by the KrdWrd Add-on .

The Proxy data pool holds unmodified, re-loadable (near[9]) copies of all the Web pages from within the KrdWrd Corpus.

en détail: We set-up and configured two instances of wwwoffle on the KrdWrd Host; one publicly available, operating in offline mode and constituting the KrdWrd Offline Proxy, and one for use by the KrdWrd Harvester, operating in online mode and constituting the KrdWrd Online Proxy. The two instances are operational at the same time and they share the same data pool; this is easily possible and does not result in data inconsistencies because the offline proxy only reads data from the pool – it never writes data to the pool. Additionally, we configured the online proxy to *never* re-grab material, i.e. the first encounter of new content will be the one the systems keeps.

The currently used configuration can be found at https://krdwrd.org/trac/browser/trunk/src/utils/wwwoffle.

3.3 Manual Annotation: Classification of Web-Page Content by Human Annotators

The pre-processed data is now ready to be processed by annotators, and we will present the setting in which the annotated data, the foundation for the gold standard, was acquired.

The KrdWrd System incorporates the KrdWrd Add-on, an extension for the Firefox browser, which facilitates the visual tagging of Web pages. However, users also need to be told *what* to tag *how* – therefore, a refined version of the official 'CLEANEVAL: Guidelines for annotators' [@CEan] is provided, and – additionally – users are encouraged to work through a small tutorial to get acquainted with different aspects of how to apply the guidelines to real-world Web pages. The snag of finding people to actually put the system into use was kindly solved by the lecturers of the *Introduction to Computational Linguistics* class of 2008 from the Cognitive Science Program at the University of Osnabrück by means of an homework assignment for students.

3.3.1 The KrdWrd Add-on: An Annotation Platform

The KrdWrd Add-on receives data from the server, modifies the rendering of Web pages by highlighting selected text, supports the tagging of different parts of a page differently, and finally, sends an annotated page back to the server for storage and subsequent processing.

[8] with the exception that there exists a *dummy* login...

[9] Dynamically generated links are challenging and may lead to missing content.

It extends the functionality of the Firefox browser with a status-bar menu where – beside some administrative tasks – the user may choose to put the current browser tab into *tracking mode*. In this mode pre-defined colour coded tags are integrated into the familiar view of a Web page A) to highlight the part of the page where the mouse is hovering over and thereby, is subject to tagging, and B) to highlight the already tagged parts of the page.

The annotation process is straightforward (cf. figure 3.1 for a partly annotated page):

1. Users move the mouse over the Web page and the block of text *under* the mouse pointer is highlighted (Sometimes this block will be rather small, sometimes it may cover large portions of text),

2. Users assign tags to the highlighted blocks by either using assigned keyboard shortcuts or via entries in the context menu (Afterwards, these blocks stay coloured in the respective colours of the assigned tags),

3. Users submit the page, i.e. the Web page *and* the incorporated tags are transfered to the server – this is done by pressing a shortcut or via an entry in the status-bar menu (The tagged page, or a partly tagged page, for that matter, can be re-submitted to the server), and

4. The KrdWrd System serves a new, untagged page for tagging[10].

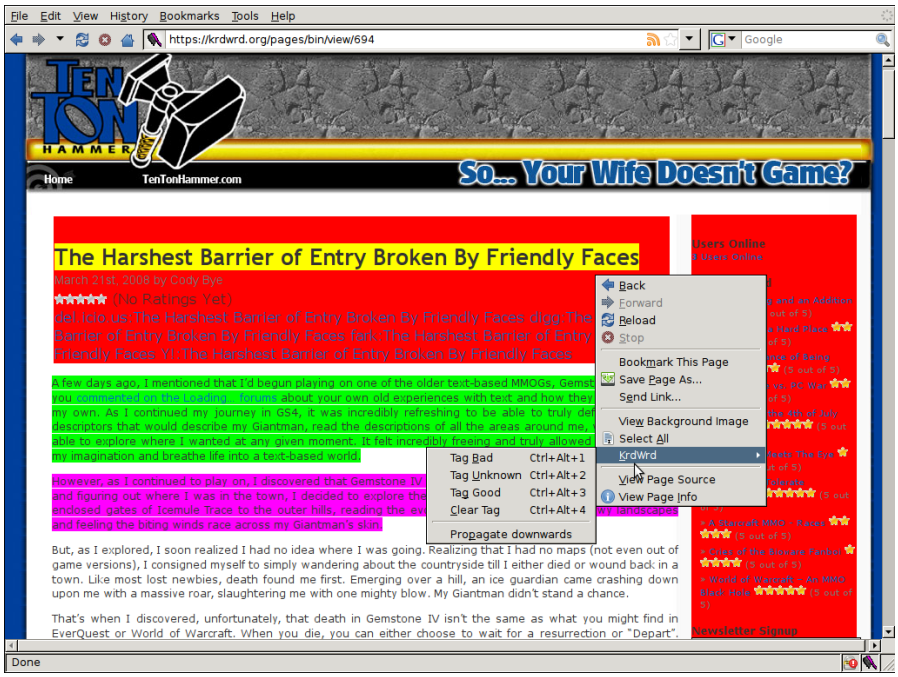


Figure 3.1: We used the lovely colour fuchsia to highlight the part of the page where the mouse is hovering over, and the colours red, yellow, and green[11] for the already tagged parts, where red corresponded to *Bad*, yellow to *Unknown*, and green to *Good* (cf. 3.3.2 for details).

[10]This new page is randomly selected among the set of pages with the lowest count of aggregated submissions per user, i.e. at large, the submissions will be evenly distributed over the corpus – but cf. 3.3

Furthermore, the KrdWrd Add-on is accompanied by a manual [Krd] (cf. appendix A), which explains how to install the Add-on, get started with tagging pages, how to actually tag them, i.e. it includes the annotation guidelines, and also gives some tips & tricks on common tasks and problems.

The Add-on is available from https://krdwrd.org/trac/wiki/AddOn.

3.3.2 The KrdWrd Annotation Guidelines

The KrdWrd Annotation Guidelines specify which tag should be assigned to particular kinds of text. We used the CleanEval (CE) annotation guidelines as a start (cf. [@CEan]), and made a few substantial changes however, because we realised that there were several cases in which their guidelines were insufficient.

The most important change we made was the addition of a third tag 'uncertain' whereas originally, only the two tags 'good' and 'bad' were available. It had soon become apparent that on some Web pages there were passages that we did not want to be part of a corpus (i.e. that we did not want to tag 'good') but that we did not want to throw out altogether either (i.e. tag them as 'bad'). We also decided to tag all captions as 'uncertain'.

Another rationale behind this introduction of this third tag was that we might want to process this data at a later stage. Also note that in [SMP08] other CE participants also used a three-element tag set.

We adopted the following guidelines from the CE contest, and all of these items were supposed to be tagged 'bad':

- Navigation information
- Copyright notices and other legal information
- Standard header, footer, and template material that are repeated across (a subset of) the pages of the same site

We modified the requirement to clean Web pages of internal and external link lists and of advertisement slightly: The KrdWrd Guidelines state that *all* (hyper-)links that are *not* part of the text are supposed to be tagged as 'bad'. This, of course, includes link lists of various kinds, but preserves links that are grammatically embedded in 'good' text. We also restricted ourselves as to discard advertisement from *external* sites only. Some of the pages were pages about certain products, i.e. advertisement, but we did not want to exclude these texts (if they fulfilled our requirements for 'good' text, as defined below).

The two sorts of text, we did not exclude specifically (as the CE guidelines did), were Web-spam, such as automated postings by spammer or blogger, and cited passages. Instead, we required 'good' text to consist of complete and grammatical English sentences that did not contain 'non-words' such as file names. That way, we filter out automatically generated text *only if* it is not grammatical or does not make up complete sentences, and keep text that can be useful for information extraction with statistical models.

Our refined annotation guidelines still leave some small room for uncertainties (but probably *all* such guidelines suffer from this problem). We are optimistic, however, that they are a

[11] fuchsia – there is a short story behind this color: https://krdwrd.org/trac/wiki/KrdWrd – red, yellow, and green, respectively

clear improvement over the original CE guidelines and that our Web corpus will only contain complete and grammatical English sentences that contain 'normal' words only.

The annotation guidelines are available from https://krdwrd.org/manual/html/.

3.3.3 The KrdWrd Tutorial: Training for the Annotators

For initial practice, we developed an interactive tutorial that can be completed online (as feature of an installed Add-on).

The interactive tutorial can be accessed from the status bar by clicking 'Start Tutorial', and is designed to practice the annotation process itself and to learn how to use the three different tags correctly. Eleven sample pages are displayed one after another, ranging from easy to difficult (these are the same samples as in the 'How to Tag Pages' section of the manual).

The user is asked to tag the displayed pages according to the guidelines presented in the manual. We inserted a validation step between the clicking of 'Submit' and the presentation of the next page, giving the user feedback on whether or not she used the tags correctly. Passages that are tagged in accordance with our annotations are displayed in a light-coloured version of the original tag, i.e. text correctly tagged as 'bad' will be light-red, 'good' text will be light-green, and text that was tagged correctly as 'uncertain' will be light-yellow. The passages with *differing* annotations are displayed in the colour in which they should have been tagged, using the normal colours, i.e. saturated red, green, and yellow (cf. 2.3). After clicking 'Next Page' on the right top of the screen, the next page will be shown.

If a user should decide to quit the interactive tutorial before having tagged all eleven sample pages, the next time she opens the tutorial, it will begin with the first of the pages that have not been tagged, yet. And should a user want to start the tutorial from the beginning, she can delete previous annotations via 'My Stats' in the status bar. Then, the next time the tutorial is opened it will start from the very beginning. By pressing 'Start Tutorial' in the status bar during the practice and *before* the submission of the current page, that same page will be displayed again, un-annotated. When using 'Start Tutorial' *after* a page's submission and before clicking 'Next Page' in the notification box at the top, the next page of the tutorial will be shown.

As stated above, it is our goal that the interactive tutorial will help users getting used to the annotation process, and we are also optimistic that it helps understanding and correctly applying the tagging guidelines as presented in the manual.

3.3.4 The KrdWrd Assignment: A Competitive Shared Annotation Task

Finally, our efforts were incorporated into an assignment for the class 'Introduction to Computational Linguistics' where – from a maximum number of 100 students – 68 completed the assignment, i.e. their effort was worth at least 50% of the assignment's total regular credits. The assignment was handed out 7, July, was due 18, July 2008, and consisted of two exercises:

1. The first task was to complete the interactive online tutorial, i.e. the students had to go through the eleven sample pages, annotate them, and – ideally – think about the feedback. This task was worth 20% of the credits.

2. The second task was to tag pages from our assembled corpus; 15 tagged pages were worth 80% of the credits and 10 additional pages were worth an extra that was counted

towards the credits of all other homework assignments, i.e. students could make up for 'lost' credits[12].

The assignment is enclosed in the appendix (cf. A).

3.4 The Gold Standard: Compilation and Analysis of manually annotated Data

The data for the gold standard was collected via the KrdWrd Add-on (cf. 3.3.1) as an homework assignment (cf. 3.3.4) for a Computational Linguistics class, which is a second year undergraduate Cognitive Science class at the University of Osnabrück. The Annotators were introduced to the KrdWrd Annotation Guidelines (cf. 3.3.2) by means of the KrdWrd Tutorial (cf. 3.3.3), and were supposed to work independently (e.g. from their home PC) though, could have sat near each other. However, we did take precautions against naïve copying by enforcing authentication for the users with their student accounts, hiding other users' results, and serving random pages for tagging – thus, even if students were exchanging information it could rather have been about the assignment and the tagging in general than a specific Web site in particular.

3.4.1 Initial Observations

From 100 student subscribed to the class 69 installed the Add-on and submitted at least one page (not necessarily a tagged one, though...). This was also about the ratio of students who took the final exam for this course hence, we can say that almost every students seriously interested in finishing this class also took the homework assignment.

The majority of submissions came within the last 4 days of the period of time they were granted to finish the assignment – with a major peak at the last day; which, according to all we know, is quite common. This has probably also led to only very few people making use of the re-submit feature, i.e. continuing or modifying an already submitted page.

The possibility to interact with the KrdWrd Team, e.g. to solve installation problems, or to exchange information via an e-Mail list we had set up for this purpose was rarely used (cf. `https://krdwrd.org/trac/mail/threads`). The few reported problems however, led to some beneficial improvements of the documentation.

Our initial Data Set (before further clean-up): 228 Web pages, consisting of almost 440,000 words and over 2.6 million characters, were independently processed by 69 users who submitted 1767 results (re-submits for a page counted only once), which is an average of 7.75 submissions per page.

3.4.2 The KrdWrd App: Annotations Merging Mode

The KrdWrd App in merging mode compares the initially grabed *master* with the user submitted results and, for every text node in the DOM tree, computes a majority vote and assigns this as the gold standard tag to the node of a newly created document.

[12] As a matter of fact, 43 students received the total of 100% regular credits + 100% extra credits.

The process is carried out offline on the server: the input is one URL of a master document and the URLs of the respective user submitted results. After reading all documents the DOM trees of all documents are traversed top-down and tags along the traversal are propagated further down as long as, no more specific tag is encountered, i.e. a tag can not overwrite another one further down the path but is *pushed down* as far as possible (cf. figure 3.2 for an illustration). At the end of each path in the tree the assigned tags are counted. After having traversed all documents a sanity check is carried out[13] [14] namely, are there documents which still have unseen nodes, or are there documents which had less nodes than the master document? In either case, these submissions are discarded from further processing.

The remaining submissions are taken into account for the majority vote on each node of the master document. Another document is generated, which includes the resulting tags.

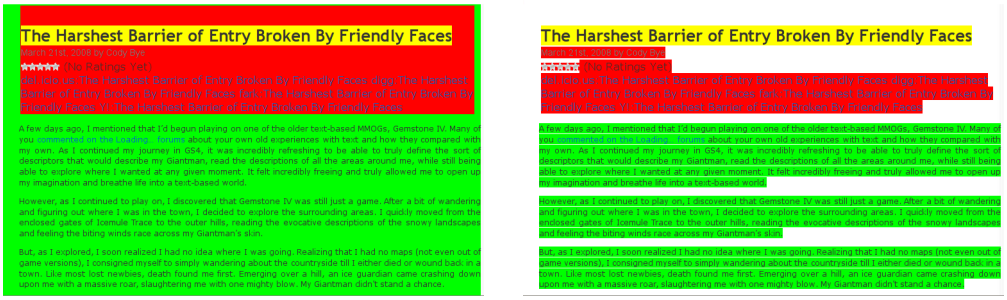


Figure 3.2: On the left is the un-propagated page with major parts having been tagged green and red. On the right is the propagated version where the green has been pushed down into the text nodes; the same holds for red but note that the heading in yellow has not been overwritten.

3.4.3 Merge Analysis

Before we started to analyse the results of the merging process we excluded the results of one user who had only submitted one page. Then, the merging process revealed the following problematic cases (usually by rejecting user results on grounds of the sanity check): 2 pages with no results left to merge, 3 pages with only one result to merge, 2 more pages with only two result to merge, 1 page with four results to merge, and 1 page that could not be merged due to an error in our application[15]. We also excluded all these cases from further processing (cf. figure 3.3).

We continued to do a plausibility check for the submitted results: we computed a *tag-bias* for each user, where we compared each user's tendency to chose a tag for a node with the actual winning tags for nodes. This computation revealed 4 cases in which users showed strong biases towards certain tags[16]. We also excluded all results from these users.

[13] We also implemented another sanity check namely, to check whether the textual content in the nodes is identical but dropped this condition – mainly because the few encounters were false positives and it had negative impact on performance as well.

[14] 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.

[15] We fixed the error but this rendered the submitted pages unusable – newly submitted pages will be mergable.

[16] Manual inspection of these cases showed that the users obviously only wanted to raise their tagged-pages

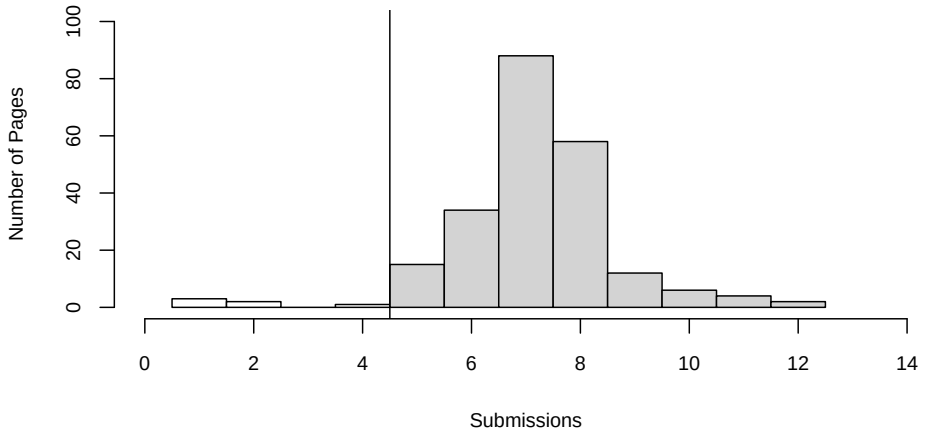


Figure 3.3: Number of Pages with x Submissions - the dividing line at *5 Submissions* shows the cut-off, i.e. pages with less then 5 Submissions were excluded from further processing. The observant reader may notice that we said the annotations were evenly distributed: this is the case, now. We had not turned on this feature when we started collecting the data, however.

The resulting and final Data Set: 219 Web pages, consisting of more than 420,000 words and over 2.5 million characters, were independently processed by 64 users who submitted 1595 results (re-submits for a page counted only once), which is an average of 7.28 submissions per page.

We continued our analyses of this new data at hand, and looked into the timestamps we collected for the submissions: therefore, we summed up all the deltas between two submissions for each user and calculated the duration each user *saw* a single page; then, we computed a reasonable upper bound for how log a submit action might take, i.e. the hypothesis was that page-view times longer than a certain amount of time were actually breaks. To this end, we detected outliers[17] and discarded *all* respective submissions (the calculated[R D08] result was 700s).

The calculated time data suggests that:

- the majority of users spent between 56 and 88 minutes on the assignment with an average of 71 minutes (cf. figure 3.4 for details),
- average per-page annotation time drops below three minutes (cf. figure 3.5), and
- the first pages after the tutorial are still more challenging than later ones (cf. 3.6).

count and therefore, just tagged very few nodes – typically high up in the DOM tree – which were then propagated downwards.

[17] This is quite standard: x values outside the range $Q1 - 1.5 * IQR < x < 1.5 * IQR + Q3$ were considered outliers.

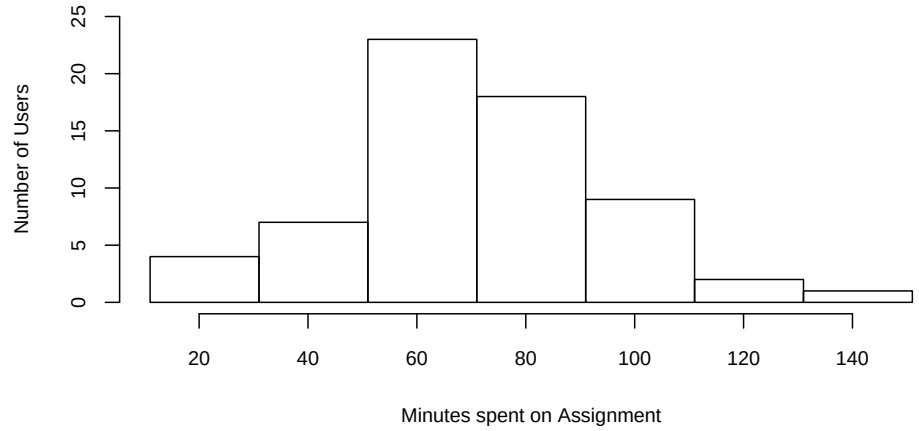


Figure 3.4: Time in Minutes spent by y Users on the Assignment, i.e. how much Time did a User interact with the Add-on to tag her share of the Canola Corpus.

For the overall inter-coder agreement of the remaining submissions we calculated Fleiss's multi-π as layed out in [AP08]: for each Web page the remaining submissions were set as coders, and the tagged DOM nodes as items – the three categories were fixed. This resulted in an average inter-coder agreement over all pages of 0.85 (cf. 3.8), which we think is – at least – substantial. Considering that these submissions were the basis for the merge process we believe that the Canola Gold Standard Corpus is a solid basis for further processing. Furthermore, this metric could be used for comparison of cleaning results in general – maybe normalised for the number of words or characters per DOM node.

Remark: we looked into the pages at the lower end of the agreement spectrum and found that they tended to be quite long and were often discussion forum pages, i.e. with many alterations in the tags that were to assign. Given that similar shorter pages achieved better results, it seems that even our already quite low boundary of 6,000 words per page resulted in pages that were frustrating to process.

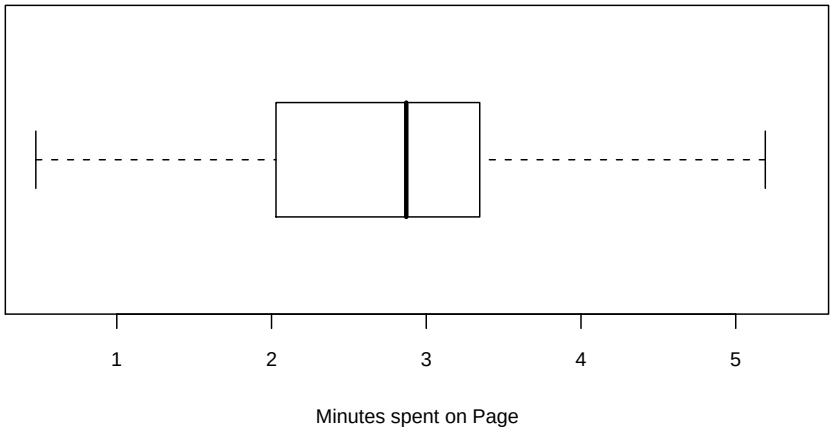


Figure 3.5: Minutes spent on a single Page accross all annotations of the Canola corpus.

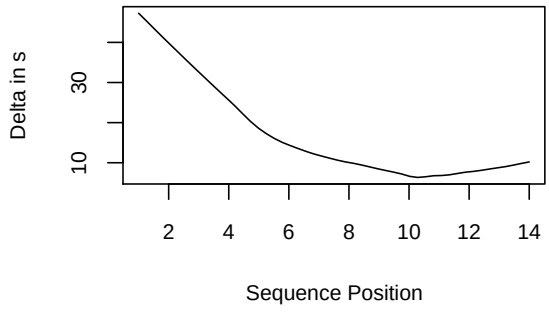


Figure 3.6: Smoothened average of differences in seconds between annotation times of all users at Position x in their specific sequences of Web Pages and the mean of all other users who processed identical pages at a later time in their respective sequences.

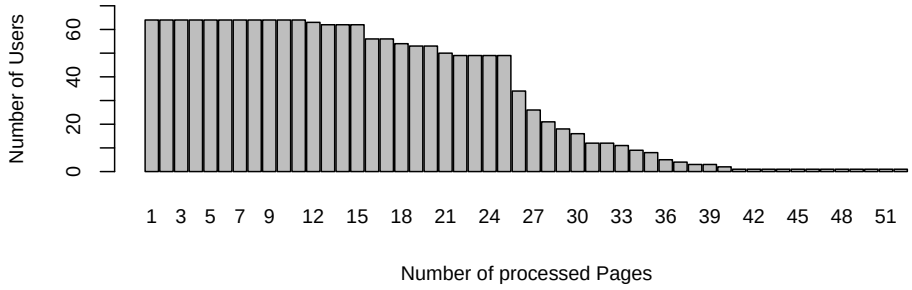


Figure 3.7: Aggregated counts for the Number of Users who processed *at least* x Number of Pages. Note the two steps at 15 and 25 pages, which correspond to the obligatory and the optional number of pages in the assignment. Also note that there were quite many students who went far beyond the requirement for the assignment.

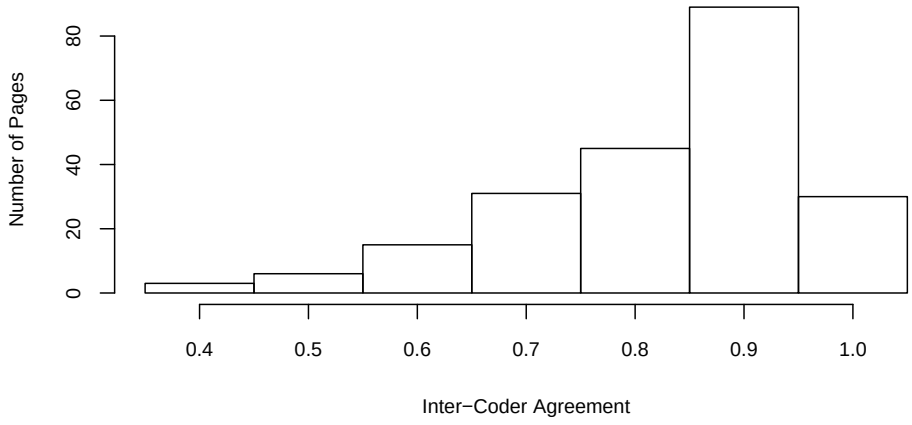


Figure 3.8: Inter-coder agreement between submissions for pages over the Canola corpus.

4 The KrdWrd Machine Learning Framework – Perceptually Driven Sweeping of Web Pages

4.1 Extraction Pipeline

Feature Extraction commences by running the KrdWrd Application Extraction Pipeline over the merged data obtained during annotation. For the Canola corpus, it took 2.5 seconds on average per page to generate text (2.5 million characters total), DOM information (46,575 nodes total), screen-shots (avg. size 997x4652 pixels) and a file with the target class for each text node.

We only used the stock KrdWrd features on the DOM tree and visual pipeline, with a simple JAMF graph as a showcase (c.f. figure 4.1). For computing textual features, we borrowed Victor's [SMP08] text feature extractor.

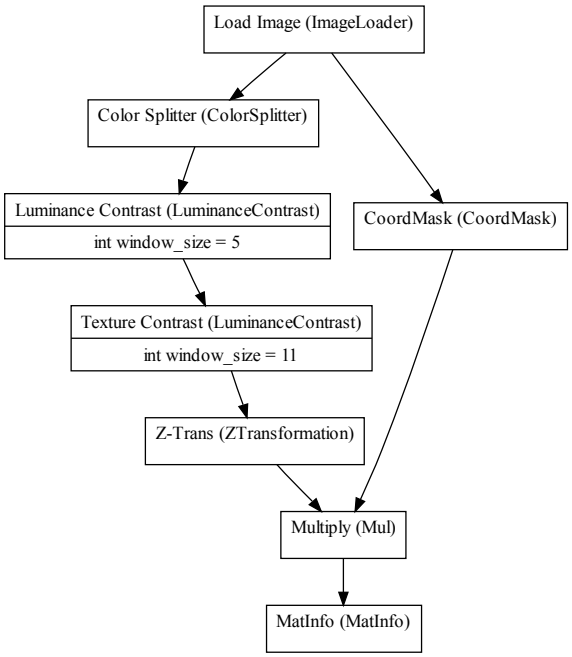


Figure 4.1: The simple JAMF graph used for the case study

Table 4.1: 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	Number of Features	Accuracy	Precision	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.

4.2 Experiment

We used the data gathered by feature extraction for training a Support Vector Machine [CL01]. 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 46,575. Vector lengths for the different pipelines and test results from 10-fold cross validation are shown in table 4.1.

Although the results for the single pipelines look quite promising – especially the visual pipeline performed surprisingly well 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.

4.3 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.

4.4 Further Work

The KrdWrd Application and supporting infrastructure are a reliable platform under a real-world usage scenario.

But 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.

Even though we showed the broad set of features for text, structure and imagery contribute to classification there is still much to be researched until the next CleanEval contest.

5 Summary and Outlook

A recent publication stated that:

> To date, cleaning has been done in isolation by each group using web data (and it has not been seen as interesting enough to publish on). Resources have not been pooled, and it has often not been done well. In CleanEval we put cleaning centre-stage. The goals of the exercise are to identify good strategies and to foster sharing of ideas and programs. [BCKS08]

Employing KrdWrd in the Canola case study showed that we achieved what we set out for and we also gained valuable experience for possible improvements: We delivered *improved Annotation Guidelines*, a *broad Annotation Set-up*, redefined the *Output Document Format*, created a solid *Development Set for ML based Web cleaning task*, and returned with the *Scoring Metric* to an often used – but still highly debated – metric.

We believe the work put forward in the KrdWrd Project is beneficial for research in the context of *Web as Corpus* but also in related areas, e.g. Content Extraction and Data Mining: they may both use our system for quick learning of extraction patters for single sites they want to harvest for content and thus, getting rid of unwanted clutter they usually need to fine-tune differently for every site.

Furthermore, the visual analysis of Web pages will certainly become more important and the KrdWrd System may be a good candidate for bridging the gap into this promising future.

A Appendix

A.1 Projekt Link

KrdWrd work is coordinated over a Trac[@TRAC] system reachable via:

http://krdwrd.org.

The system features documentation, bug tracking, source code and history. Source code metrics are provided via https://www.ohloh.net/ and show appx. 2500 relevant lines of code for KrdWrd.

A.2 Enclosed Documents

Following – begining on the next page – are the KrdWrd Homework Assignment and the KrdWrd Add-on Manual.

KrdWrd Homework

due Friday, 18.07.2008

The main objective of this assignment is to give you first hand experience on a competitive manual tagging task on pages from the World Wide Web, where you will use an online tool to tag the pages. It is *competitive* because each of your individual results will compete with others' results on identical pages and it is *manual* because you will actually have to do the task.

Pages from the Web because the Web is an unprecedented and virtually inexhaustible source of authentic natural language data and offers the NLP community an opportunity to train statistical models on much larger amounts of data than was previously possible. However, after crawling content from the Web the subsequent steps, namely, language identification, tokenising, lemmatising, part-of-speech tagging, indexing, etc. suffer from 'large and messy' training corpora and interesting regularities may easily be lost among the countless duplicates, index and directory pages, Web spam, open or disguised advertising, and boilerplate. Therefore, thorough preprocessing and cleaning of Web corpora is crucial in order to obtain reliable frequency data.

The preprocessing can be achieved in many different ways, e.g. a naïve approach might use finite-state tools with hand-crafted rules to remove unwanted content from Web pages. The KrdWrd project is heading for a quite different approach:

1. Use the visual presentation of Web pages
2. Have an initial training set of Web pages annotated – the Gold Standard
3. Extract the visual, structural, and linguistic information to train a model using machine learning algorithms
4. Use the model to automatically clean Web pages while building a corpus

The KrdWrd project homepage is available at `http://krdwrd.org` and this is also the place to get started.

Exercise 1. (2 points)

Your task is to complete the online tutorial of the KrdWrd system, i.e. you have to launch Firefox[1], install the KrdWrd Add-on, get a copy of the manual, and go through the online tutorial.

1. Use Firefox to visit `http://krdwrd.org` and follow the instructions, i.e. install the necessary certificate
2. Go to `https://krdwrd.org/trac/wiki/AddOn` and follow the installation steps for the add-on
3. Read through the manual – make sure to cover at least *Introduction* and *Getting Started*
4. Start the tutorial by selecting *Start Tutorial* from the KrdWrd broom status bar menu on the lower right of your browser window
5. Read through the page – thoroughly – and finish the tutorial

[1] If you have not installed Firefox, yet, visit `http://www.mozilla.com` and download your copy.

Exercise 2. (8+10 points)

Your task is to tag pages from the Canola corpus: 15 well tagged pages will be worth 8 credits[2]; well tagged additional 10 pages will be worth 10 *extra* credits.

1. Select the *Canola* corpus from the *Corpus* menu and start tagging
2. Visit the *My Stats* page from time to time to see how many pages you have already tagged...

Notes:

- You can always interrupt tagging and continue at a later time: just select the Canola corpus and continue.
- In case where something goes wrong go to `https://krdwrd.org` and use the search feature to look for a solution.
- Iff you have not found a solution write a mail to `krdwrd@krdwrd.org` with a detailed problem description, i.e.:
  - What is the problem?
  - What were the steps that led to the problem?
  - Include the last lines of information from the *Help/About Mozilla Firefox* menu, i.e. the ones that start with `Mozilla/...` **and** include the first line of information from the *About* menu item in the add-on, i.e. the line `krdwrd version 0.x.y`.
  * We know that well-written bug reports are an extra effort and encourage them. Every unique substantial bug report leding to a fix in the add-on is worth a maximum of three extra credits[3].
- In case you have no other hardware to use you can use the computers in B10 aka. 31/412; however, make sure to substitute every occurrence of Firefox with Iceweasel, i.e. `s/Firefox/Iceweasel/g`, in all documentation.

[2] Together with Task 1 this corresponds to 100% of this assignment.
[3] However, you cannot exceed the 10 *extra credits* hard limit for this assignment.

The KrdWrd Add-on Manual

1 Introduction

"The availability of large text corpora has changed the scientific approach to language in linguistics and cognitive science" [M&S]. Today, the by far richest source for authentic natural language data is the World Wide Web, and making it useful as a data source for scientific research is imperative.

Web pages, however, can not be used for computational linguistic processing without filtering: They contain code for processing by the Web browser, there are menus, headers, footers, form fields, teasers, out-links, spam-text – all of which needs to be stripped.

The dimension of this task calls for an automated solution, the broadness of the problem for machine learning based approaches. Part of the KrdWrd project deals with the development of appropriate methods, but they require hand-annotated pages for training.

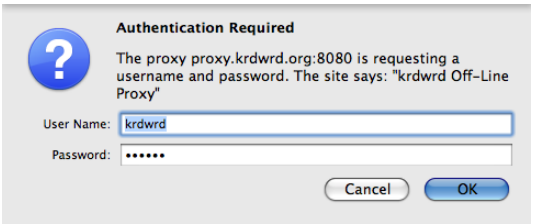
The *KrdWrd Add-on* aims at making this kind of tagging of Web pages possible. For users, we provide accurate Web page presentation and annotation utilities in a typical browsing environment, while preserving the original document and all the additional information contained therein.

2 Getting Started

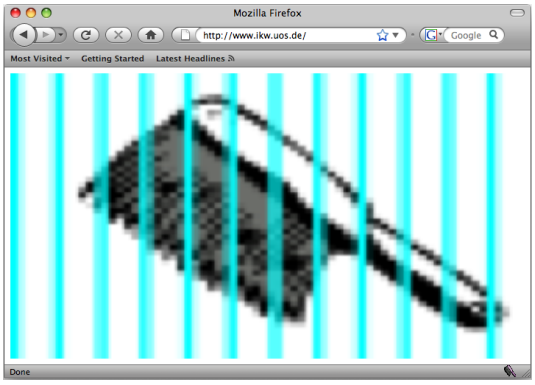
In this section, we will give you information about how to use the tool. If you have not installed it yet, go to krdwrd.org and get it. Of course, you will need Firefox, too.

- Since the add-on depends on a special proxy server to connect to the Internet - you can only grab and submit Web pages from the KrdWrd corpora - it may be a good idea to create a separate profile just for working with the add-on. If you want to create a profile but have no idea how to do that, have a look here.

- When grabbing a page for the first time, or selecting a corpus for the first time you will be asked to authenticate for the krdwrd Off-Line Proxy. The username and password in the dialog box are already filled in and it is save to leave the "Use Password Manager to remember this password" checked.



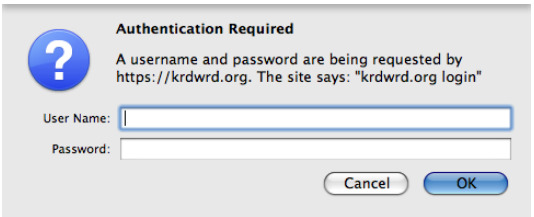
- The proxy server will deny all requests that are not part of the normal add-on operation. If you ever see something like



it is most likely because you tried to surf the Web with the wrong Firefox profile.

- You will be asked for authentication a second time. This authentication is for the KrdWrd Web site and requires your *RZ Account*[1]

[1]This is the same login as for *Stud.IP* and *WebMail*; in case you want to "Use Password Manger" please also "Use a master password" to protect your sensitive information.



- When you request a page from the corpus for the first time, Firefox will popup a security warning. The warning says *"you have requested an encrypted page that contains some unencrypted data"*. The warning is issued because the corpus page are issued unencrypted. Your login credentials are never send to the server unencrypted, there is no reason not to ignore this warning.

2.1 First Steps

- ### How to Use the Mouse

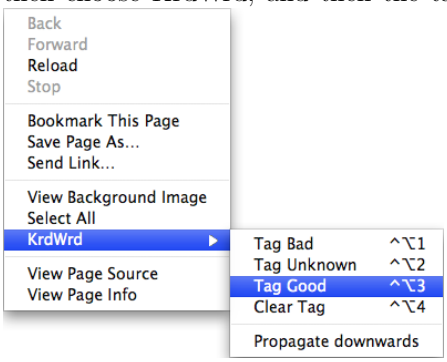
  When moving the mouse over a Web page, you will notice that certain areas are highlighted in pink. These are the blocks of text that you can tag. Sometimes the pink areas are fairly small (single words or lines of text), sometimes they are pretty large (whole paragraphs, or even whole pages). Thus it makes sense to move the mouse around a little before you actually start tagging because sometimes you want to tag big areas as, say, 'bad', and it saves you a lot of time if you do not have to tag every single line or paragraph. As a rule of thumb, it often makes sense to tag *everything* in red ('bad'), from top to bottom, and only then start tagging smaller pieces in yellow or green ('uncertain' or 'good', respectively) (see also **Examples, described on page 3.2, Tips & Tricks, page 4**).

- ### How to Choose the Tag

  This section deals with assigning tags. If you want information on how to choose the *right* tag to assign, go to the Annotation Guidlines on page 3.1.

For tagging a pink-highlighted section as 'good', 'bad', or 'uncertain', you have two options: You can use (1) keyboard shortcuts (hotkeys) or you can use (2) the context menu (rightclick).

1. Keyboard Shortcuts
   - *bad*: ctrl+alt+1
   - *uncertain*: ctrl+alt+2
   - *good*: ctrl+alt+3
   - *clear annotation*: ctrl+alt+4
2. Context Menu
   - Rightclick when you are over the section you want to tag, then choose KrdWrd, and then the tag you want to assign.

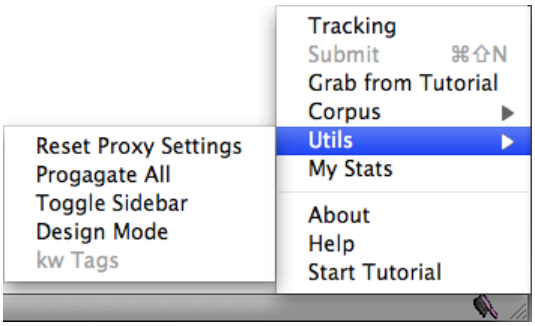


   - Using the context menu is not recommended, however. It is much more time-consuming to navigate the menu than to use the keyboard shortcuts.
   - Note also that if the mouse cursor leaves the menu area, a possibly different part of the page will be highlighted for tagging (namely the part that is now 'under' your mouse).

- **Cookies and Certificates**
  When some of the pages are being loaded, your Web browser will ask you whether you want to accept cookies (of course, depending on your browser settings: If you use a separate profile for the KrdWrd add-on, just allow all cookies, see Tips & Tricks, page 4). Actually, you *do not* have to accept any cookies; however, nothing bad will happen if you do accept them.

2.2 The Statusbar Menu

This section describes the status bar menu, depicted below.



- **Tracking**
  Here you can turn on or off whether sections of the Web page you are currently viewing are highlighted in pink. Usually there is no need to disable tracking. However, in some rare cases, this might help you to get a better view on a page before tagging it.

- **Submit**
  When you are done tagging a page, i.e. when everything on the page is green, red, or yellow, you can submit the page with this menu option - and the next page will load automatically. For your convenience, you can also use the keyboard shortcut *options+shift+N*.

- **Grab Page**
  Clicking here loads a new, un-annotated page. Once you annotated the whole corpus, you will be redirected to your personal statistics page.

- **Corpus**
  Here you can select one of the (predefined) available corpora. But you should stick to the Canola corpus for now.

- **Utils**
  The options in this menu make your life easier when tagging pages.

  - **Propagate**
    Here you can explicitly propagate a given tag down to all sibling nodes. This is helpful when you have a large portion that should be tagged red but all its siblings should be tagged green. You can then tag the parent node green, propagate, and re-tag the parent node as red. This way you do not need to tag all the siblings separately. (Try this on the Examples, described on page 3.2 and check the Tips & Tricks, page 4.2).
  - **Toggle Sidebar**
    Clicking here opens the sidebar. In the sidebar you can see all of the text in the current page and how it is tagged. A given tag is usually propagated down to lower nodes in the DOM tree automatically, but sometimes it may be unclear (i.e. not directly visible in the page) how a particular portion of text is tagged. In the sidebar you can easily see whether it is tagged red, green, or yellow.
  - **Design Mode**
    This is a debugging feature and you must not use it while tagging pages.

- **My Stats**
  This menu option will send you to your KrdWrd account. There you can see how many pages you have already tagged, and you can view, re-submit and delete your tagged pages.

3 How to Tag Pages

3.1 Annotation Guidelines

In the previous section, we described how to use the tool and how to assign tags. In the following, we give you guidelines regarding *which* tag should be assigned to a particular kind of text.

- Everything that is *boilerplate* is tagged **red**. *Boilerplate* is ...

  1. all navigation information,

2. copyright information,
3. hyperlinks that are not part of the text,
4. all kinds of headers and footers.

→ Generally speaking, boilerplate is everything that can be used interchangeably with any other Web page or could be left out without changing the general content of the page.

- The following types of text are also tagged **red**:
  1. incomplete sentences, or text in telegraphic style,
  2. text containing 'non-words' such as file names,
  3. off-site advertisements (i.e. advertisement from an external page),
  4. text in any other language than English,
  5. lists and enumerations of any kind.
- All captions are tagged **yellow**. And also everything that does not belong in the red or green category is tagged **yellow**.
- All text that is left is tagged **green**, i.e. ...
  1. text made up of complete sentences, **even if it is in a list or enumeration**,
  2. text that makes use of 'normal' words.
  3. text that is written in English.

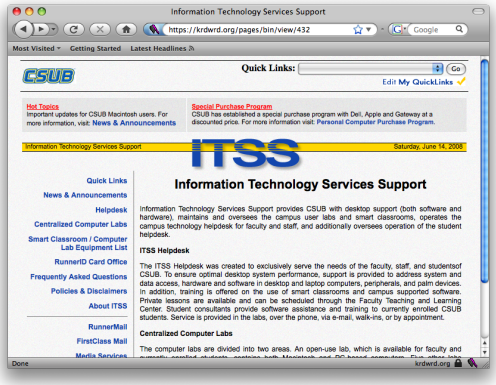
Simple, isn't it? You will notice that on some pages you can only highlight very large areas, on others the choices are less restricted. If you tag an element, the tag assigned is propagated to all elements that are contained in this area. However, if you are not sure whether a specific element is entailed, just tag it too to be on the safe side (remember the *sidebar option* mentioned in the previous section!).

In a previous section, we said that as a rule of thumb, it often makes sense to tag *everything* in red ('bad'), from top to bottom, and only then to start tagging smaller pieces in yellow or green ('uncertain' or 'good', respectively). The easiest way to tag a whole page red is to tag the outermost rim of the page and tag that as 'bad'. Due to the tag propagation, the whole page is now tagged as 'bad'. If you want to make sure that this is so, check the sidebar (see above).

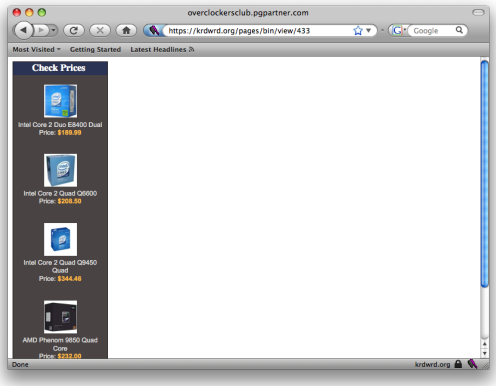
This may all be a bit confusing now. But fear not, in the next sections you will have the possibility to check whether you understood everything.

3.2 Examples - Easy

- Example 1
  This is a fairly standard Web page. Advertisements and boilerplate should be easy to spot and easy to tag.



- Example 2
  This should be easy, too.



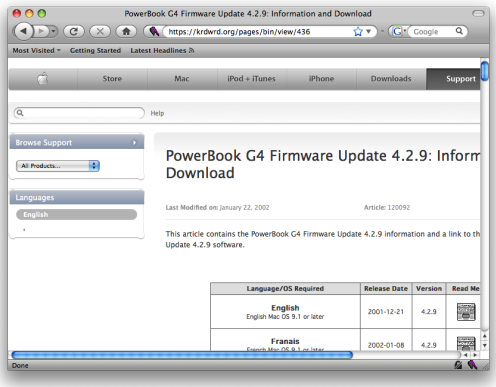
- Example 3
  Similar to *Example 1* but you will have to invest a little more time, since the layout is not as clean. Is there something that is not 'good' in the text portion? How should you treat the headlines? What about the headlines' subtitles?

- Example 4
  Somehow similar to *Example 2*. Why is even the text portion not 'good'?

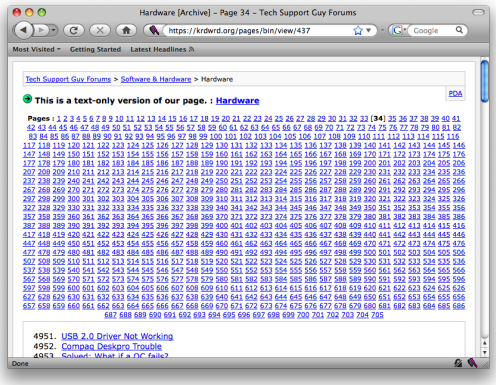
3.3 Examples - Medium

- Example 5
  Remember wich language you should tag (and that all text in another

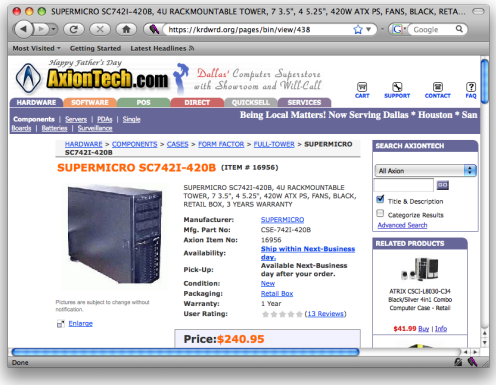
language is bad). How should you tag the enumerations?



- Example 6
  This one is all about enumerations.

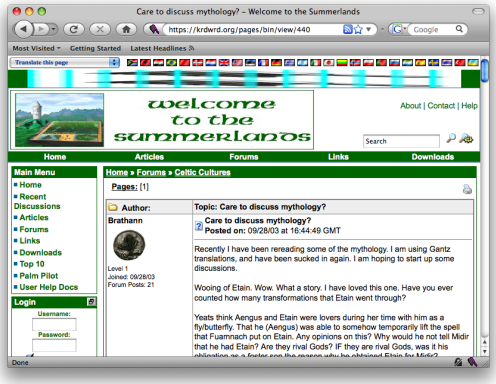


- Example 7
  Once you have decided how much of the text is junk, this is fairly easy. Propagate is you friend.



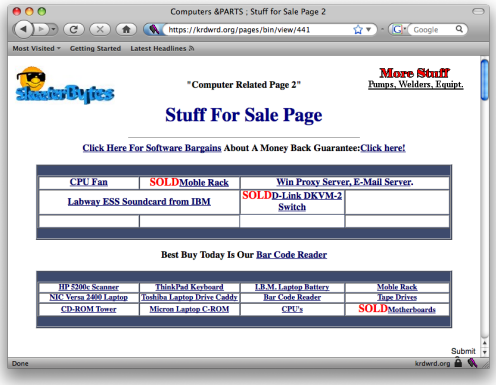
- Example 8
  This can be easy with the right strategy. One of the rare pages where it is easier if you don't start with tagging everything red first.

- Example 9
  Sometimes there are no technical difficulties.

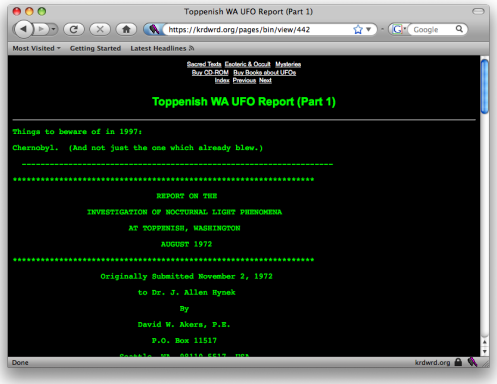


3.4 Examples - Hard

- Example 10
  By now this should be easy for you.



- Example 12
  This one is a bit like example 9 but with technical difficulties. You might rather want to go to your dentist. This is really as bad as it can be. If you can do this all other pages are a piece of cake.



4 Tips & Tricks

4.1 Keyboard Shortcuts

The default shortcuts for tags are

- *bad*: ctrl+alt+1
- *uncertain*: ctrl+alt+2
- *good*: ctrl+alt+3
- *clear annotation*: ctrl+alt+4

Depending on the size of your keyboard and your hands, this may really hurt after some pages. But you can change the shortcuts. All you need is the keyconfig add-on.

- Install keyconfig.
- Bring up the keyconfig menu by pressing *Ctrl+Shift+F12*. (Mac users press *Command+Shift+F12*).
- The commands you want to change are named **Tag Bad**, **Tag Good** and **Tag Unknown**.
- Close your Firefox window and reopen it; otherwise, the newly set shortcuts will not work.

4.2 How and When To Use Propagate

There are two main uses for the propagate utility. Either there are many good text portions embeded in bad text portions (or vice versa). Or there are many small chunks of text cluttered around the page.

With propagate you can often get around tagging each chunk individually.

- Remember to check each text portion's tag to be correct.
- It is important that text is tagged right, you don't have to care about the background color (and you really shouldn't).
- Propagate will tag text and text only. And you really should not care about the color of the background where the text is written on.
- Most pages in Examples - Medium, page 3.3 are significantly faster to tag when using the propagate utility.

4.3 How to *Undo*

Currently the add-on has no readily available *Undo* function (which might come handy in cases where you *propagated* the wrong tag). However, the *My Stats* page lets you *Delete* certain, committed pages.

A.3 Trivia

[ˈkæːr̆d ˌvɪr̆d]

Logo and Name are in reference to Kehrd Wärd (http://www.asn-spatz.de/), where citizens clean their part of a Franconian town for the greater good. For the full experience a Franconian has to mumble something along the lines of *"gekehrt wird!"*.

Bibliography

The Web sites referred to below were last accessed on March 20, 2009. In case of unavailability at a later time, we recommend visiting the Internet Archive.

[AP08] Ron Artstein and Massimo Poesio. Inter-coder agreement for computational linguistics. *Computational Linguistics*, 34(4):555–596, 2008. Available from: http://www.mitpressjournals.org/doi/abs/10.1162/coli.07-034-R2.

[BB04] Marco Baroni and Silvia Bernardini. BootCaT: Bootstrapping corpora and terms from the web. In (ELRA) [EL04], pages 1313–1316. Available from: http://sslmit.unibo.it/~baroni/publications/lrec2004/bootcat_lrec_2004.pdf.

[BCKS08] Marco Baroni, Francis Chantree, Kilgarriff, and Serge Sharoff. CleanEval: A competition for cleaning web pages. In (ELRA) [EL08]. Available from: http://clic.cimec.unitn.it/marco/publications/lrec2008/lrec08-cleaneval.pdf.

[BDD+07] Daniel Bauer, Judith Degen, Xiaoye Deng, Priska Herger, Jan Gasthaus, Eugenie Giesbrecht, Lina Jansen, Christin Kalina, Thorben Krüger, Robert Märtin, Martin Schmidt, Simon Scholler, Johannes Steger, Egon Stemle, and Stefan Evert. FIASCO: Filtering the Internet by Automatic Subtree Classification, Osnabrück. In Fairon et al. [FNKdS07]. Available from: http://purl.org/stefan.evert/PUB/BauerEtc2007_FIASCO.pdf.

[BNC] The British National Corpus (BNC) user licence. Online Version. Available from: http://www.natcorp.ox.ac.uk/docs/licence.pdf [cited 032009].

[CL01] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[DW05] Pernilla Danielsson and Martijn Wagenmakers, editors. *Proceedings of Corpus Linguistics 2005*, volume 1 of *The Corpus Linguistics Conference Series*, 2005. ISSN 1747-9398.

[EKS08] Stefan Evert, Adam Kilgarriff, and Serge Sharoff, editors. *Can we beat Google? (WAC4 - 2008) – Proceedings of the 4th web as corpus workshop*, 06 2008. Available from: http://webascorpus.sourceforge.net/download/WAC4_2008_Proceedings.pdf.

[EL04] European Language Resources Association (ELRA), editor. *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal, May 2004. Available from: http://www.lrec-conf.org/lrec2004/.

[EL08] European Language Resources Association (ELRA), editor. *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008)*, Marrakech, Morocco, May 2008. Available from: http://www.lrec-conf.org/lrec2008/.

[Eve08] Stefan Evert. A lightweight and efficient tool for cleaning web pages. In (ELRA) [EL08]. Available from: http://purl.org/stefan.evert/PUB/Evert2008_NCleaner.pdf.

[FNKdS07] Cédrick Fairon, Hubert Naets, Adam Kilgarriff, and Gilles-Maurice de Schryver, editors. *Building and Exploring Web Corpora (WAC3 - 2007) – Proceedings of the 3rd web as corpus workshop, incorporating CLEANEVAL*, Louvain-la-Neuve, July 2007. Presses universitaires de Louvain.

[GHHW01] Ben Goodger, Ian Hickson, David Hyatt, and Chris Waterson. Xml user interface language (xul) 1.0. Recommendation, Mozilla.org, 2001.

[GN00] Gregory Grefenstette and Julien Nioche. Estimation of english and non-english language use on the WWW. In *In Recherche d'Information Assistée par Ordinateur (RIAO)*, pages 237–246, 2000.

[HHW+04] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document object model (dom) level 3 core specification. Recommendation, W3C, 2004.

[KB06] Adam Kilgarriff and Marco Baroni, editors. *11th Conference of the European Chapter of the Association for Computational Linguistics – Proceedings of the 2nd International Workshop on Web as Corpus*, Trento, Italy, April 2006. Available from: http://www.aclweb.org/anthology-new/W/W06/W06-1700.pdf.

[KG03] Adam Kilgarriff and Gregory Grefenstette. Introduction to the special issue on the web as corpus. *Computational Linguistics*, 29:333–347, 2003.

[Kil07] Adam Kilgarriff. Googleology is bad science. *Comput. Linguist.*, 33(1):147–151, 2007.

[Krd] The KrdWrd Project. *Add-on Manual*. Available from: https://krdwrd.org/manual/manual.pdf. Online Version at https://krdwrd.org/manual/html/.

[R D08] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. Available from: http://www.R-project.org. ISBN 3-900051-07-0.

[RHJRWY04] Song Ruihua, Liu Haifeng, Wen Ji-Rong, and Ma Wei-Ying. Learning important models for web page blocks based on layout and content analysis. *SIGKDD Explor. Newsl.*, 6(2):14–23, 2004.

[SMP08] Miroslav Spousta, Michal Marek, and Pavel Pecina. Victor: the web-page cleaning tool. In Evert et al. [EKS08]. Available from: http://webascorpus.sourceforge.net/download/WAC4_2008_Proceedings.pdf.

[SWW+08] Johannes Steger, Niklas Wilming, Felix Wolfsteller, Nicolas Höning, and Peter König. 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, 2008. Available from: http://dblp.uni-trier.de/db/conf/wapcv/wapcv2008.html#StegerWWHK08.

[@ADDR] W3C. Naming and addressing: URIs, URLs, ... [online, cited 032009]. Available from: http://www.w3.org/Addressing/.

[@CEan] Marco Baroni and Serge Sharoff. CLEANEVAL: Guidelines for annotators [online, cited 032009]. Available from: http://cleaneval.sigwac.org.uk/annotation_guidelines.html.

[@CGI] The Common Gateway Interface (CGI) – a standard for external gateway programs to interface with information servers such as http servers [online, cited 032009]. Available from: http://hoohoo.ncsa.uiuc.edu/cgi/overview.html.

[@CL] The computational linguistics group of the Institute of Cognitive Science at the University of Osnabrück [online, cited 032009]. Available from: http://www.ikw.uni-osnabrueck.de/CL/.

[@DEB] Debian GNU/Linux: The free operating system for your computer [online, cited 032009]. Available from: http://www.debian.org/.

[@FF] Firefox: The browser that has it all [online, cited 032009]. Available from: http://www.mozilla.com/firefox/.

[@HTTP] The apache HTTP server project [online, cited 032009]. Available from: http://httpd.apache.org/.

[@KRDW] Johannes M. Steger and Egon W. Stemle. The KrdWrd project web site [online, cited 032009]. Available from: https://krdwrd.org/.

[@POTA] A perl module intended to perform "boilerplate" stripping and other forms of filtering [online, cited 032009]. Available from: http://sslmitdev-online.sslmit.unibo.it/wac/post_processing.php.

[@PYTH] Python: An interpreted, interactive, object-oriented programming language [online, cited 032009]. Available from: http://www.python.org/.

[@SVN] An open source version control system [online, cited 032009]. Available from: http://subversion.tigris.org/.

[@SZAG] Süddeutsche Zeitung Archiv – Allgemeine Geschäftsbedingungen [online, cited 032009]. Available from: http://www.sz-archiv.de/sueddeutsche-zeitung-archiv/onlinearchive/sz-aboarchiv-uebersicht/sz-aboarchiv-agb.

[@TRAC] An enhanced wiki and issue tracking system for software development projects [online, cited 032009]. Available from: http://trac.edgewall.org/.

[@UNIC] Unicode home page [online, cited 032009]. Available from: http://www.unicode.org/.

[@URL] URI working Group. Uniform resource locators: A syntax for the expression of access information of objects on the network [online, cited 032009]. Available from: http://www.w3.org/Addressing/URL/url-spec.txt.

[@W3ba] HTML 4.01 specification – path information: the BASE element [online, cited 032009]. Available from: http://www.w3.org/TR/html401/struct/links.html#h-12.4.

[@WC] The wc command [online, cited 032009]. Available from: http://www.bellevuelinux.org/wc.html.

[@WOFF] Andrew M. Bishop. A simple proxy server with special features for use with dial-up internet links [online, cited 032009]. Available from: http://gedanken.demon.co.uk/wwwoffle/.

[@YAHO] The Yahoo! internet search engine [online, cited 032009]. Available from: http://www.yahoo.com.