Language Resources and Evaluation manuscript No.
(will be inserted by the editor)

Accurate and efficient general-purpose boilerplate detection for crawled web corpora

Roland Schäfer

Received: date / Accepted: date

Abstract Removal of boilerplate is one of the essential tasks in web corpus construction and web indexing. Boilerplate (redundant and automatically inserted material like menus, copyright notices, navigational elements, etc.) is usually considered to be linguistically unattractive for inclusion in a web corpus. Also, search engines should not index such material because it can lead to spurious results for search terms if these terms appear in boilerplate regions of the web page. In this paper, I present and evaluate a supervised machine-learning approach to general-purpose boilerplate detection for languages based on Latin alphabets using Multi Layer Perceptrons (MLPs). It is both very efficient and very accurate (between 95% and 99% correct classifications, depending on the input language). I show that language-specific classifiers greatly improve the accuracy of boilerplate detectors. The single features used for the classification are evaluated with regard to the merit they contribute to the classification. Furthermore, I show that the accuracy of the Multilayer Perceptron is on a par with that of a wide range of other classifiers. My approach has been implemented in the open-source *texrex* web page cleaning software, and large corpora constructed using it are available from the COW initiative, including the CommonCOW corpora created from CommonCrawl datasets.

Keywords corpus construction · web corpora · boilerplate · non-destructive corpus normalization

This work is supported by the German Research Council (Deutsche Forschungsgemeinschaft, DFG) through grant SCHA1916/1-1 *Linguistic web characterization and web corpus creation*.

Roland Schäfer
Freie Universität Berlin
Deutsche und niederländische Philologie
Habelschwerdter Allee 45
Tel.: +49-30-838-51374
Fax: +49-30-838-451374
E-mail: roland.schaefer@fu-berlin.de

1 The notion of boilerplate and previous approaches

In recent years, web corpora have been created by various research groups (Biemann et al, 2007; Baroni et al, 2009; Pomikalek et al, 2009; Schäfer and Bildhauer, 2012, 2013), and they have been actively used in Computational Linguistics and Corpus Linguistics. In order to turn web pages into a usable corpus (at least for most types of research), a high amount of processing is required in order to clean them from markup and scripts, to remove junk text and duplicates, and to separate real text written by humans from redundant automatically inserted material—usually called *boilerplate*—within single documents (for an overview, cf. Schäfer and Bildhauer, 2013). Boilerplate detection is among the crucial first steps in web corpus construction. It has direct consequences for the quality of further processing steps (such as linguistic annotation), and the work presented here is therefore highly relevant for anyone working on or with large crawled web corpora. I describe a general-purpose high-performance method of boilerplate removal, i. e., one that is intended to be used for the cleaning of web pages from unselected and unknown sources.[1] The first goal was to find a set of easily extractable features and a classifier that is very accurate and very fast. The second goal was to test whether the value of the features varies cross-linguistically, which is why I trained and compared classifiers for four languages, resulting in a language-specific ranking of the features.

An essential task to solve in the development of boilerplate detectors is the definition of the notion of boilerplate itself, and different corpus creators and corpus users might have different definitions of what is boilerplate. I define boilerplate as all material that remains after markup stripping, and which does not belong to one of those blocks of content on the web page that contain coherent text. Under this definition, headlines and abstracts relating directly to a block of connected text are not treated as boilerplate, but similar elements relating to the whole web site are because they are most likely repeated on many pages from the same site. Blocks just containing date strings, user IDs, etc., are classified as boilerplate. The method described here has been implemented in the current version of an open-source web page cleaning software called *texrex* (version *texrex-behindthecow*), and it is thus open to evaluations by others.[2] Furthermore, the training and evaluation datasets are made publicly available.[3] Multi-billion token corpora built using the software described here are available from the COW project and can be queried and downloaded (in the form of sentence shuffles) using the custom web front-end Colibri[2] (Schäfer, 2015).[4]

For general-purpose automatic boilerplate detection, many types of suitable classifiers have been used, such as Support Vector Machines (SVM) (Bauer et al, 2007), Conditional Random Fields (CRF) (Marek et al, 2007; Spousta et al, 2008), Naive Bayes (Pasternack and Roth, 2009), and Multilayer Perceptrons (MLP) (Schäfer and Bildhauer, 2012). Bauer et al (2007, 120) report $f_1 = 0.652$, Spousta et al (2008, 16) report f_1 scores around 0.8 and Pasternack and Roth (2009, 977–979) report f_1 scores above 0.9, even 0.95. In a more general approach (Kohlschütter et al, 2010), shallow text features (like word counts and link density) were used and combined with several classifiers (Decision Trees, SVMs), achieving maximal f_1 scores around 0.95. Recently, Neunerdt et al (2015) have shown that a CRF clas-

[1] I use the term *general-purpose boilerplate detection* in opposition to *template-based boilerplate detection*, where the markup structure of the web page is known beforehand because, for example, only pages generated by a limited number of content management systems is processed.

[2] `http://texrex.sourceforge.net`

[3] `http://rolandschaefer.net/?p=88`

[4] `http://corporafromtheweb.org` and `https://webcorpora.org`

sifier trained on a domain-specific dataset (web pages containing comments) using several linguistic (token- and POS-related) features achieves up to $f_1 = 0.96$.[5]

All approaches mentioned so far are more or less general-purpose HTML web page boilerplate detectors using machine learning. Alternatively, researchers have used heuristics and hand-crafted algorithms or even different definitions of what should be treated as boilerplate. For example, Gallé and Renders (2014) define formulaic sequences (or *standardized text*, potentially spanning several sentences) in any type of corpus (not just HTML web pages) as boilerplate. Technically, they detect such material through a custom algorithm based on word co-occurrences. While this definition of boilerplate is more general than the one used in this paper, other researchers have employed a compatible definition, but without using machine learning. For example, boilerplate removal for the WaCky corpora (Baroni et al, 2009) was done by keeping one coherent block from each web page for which the text-to-markup ratio was maximized, a method that goes back to the Body Text Extraction algorithm by Finn et al (2001). Another state-of-the-art approach, which does not use machine learning, is the jusText algorithm by Pomikálek (2011). It is a hand-crafted algorithm using block length, link density, and stop word density as its basic inputs. Also, the context of blocks is used as a further cue to their boilerplate status. Depending on the dataset, the f_1 score is reported to range between 0.92 (Pomikálek, 2011, 51) and 0.98 (Pomikálek, 2011, 46).

The work presented here follows the machine learning approach, mostly because this allows me to perform the evaluations reported in Section 3 systematically and automatically for multiple data sets. It extends Schäfer and Bildhauer (2012), where we trained an MLP on just nine text block-internal features and achieved an acceptable precision of 0.83 at a quite mediocre recall of 0.68, resulting in an f_1 score of 0.75.[6] While the accuracy reported by Schäfer and Bildhauer (2012) is below the state of the art, the advantage of the MLP (especially when using the FANN library, cf. below) is that, once trained, it is computationally very efficient. At the same time, an MLP is highly suitable for the task of learning a binary classification based on an array of input values of all types, including non-linearly separable classifications (Grossberg, 1973; Rumelhart et al, 1986; Minsky and Papert, 1988).[7] In this paper, I go beyond Schäfer and Bildhauer (2012) by training the MLP on a much larger set of carefully engineered features, including many that have been used previously in the aforementioned publications. The thorough evaluation of the approach presented in Section 3 (including a feature ranking and an evaluation of the language-specificity of the classifiers and features) represents a major extension of my earlier work. Like Schäfer and Bildhauer (2012), I use the FANN library implementation (Nissen, 2003), which is optimized for classification speed. I classify single blocks as either boilerplate or non-boilerplate. Thus, I do not use a single-window approach as used, for example, by the WaCky project (Baroni et al, 2009). The main argument against such a single-window approach is that many web pages contain sequences of blocks alternating between boilerplate and non-boilerplate. In a forum

[5] I admit that it is generally unfair to cite single scores, since all authors perform quite differentiated evaluations with fine-grained results. The values cited here are intended for a rough orientation only, and readers are advised to refer to the papers for full evaluations.

[6] Due to the general nature and the space constraints of Schäfer and Bildhauer (2012), not all of these figures are reported in the original paper. They are, however, reported in our book on web corpus construction (Schäfer and Bildhauer, 2013, p. 56).

[7] In a two-dimensional space, two sets of points are linearly separable if they can be separated by a straight line. In higher-dimensional spaces, the straight line is generalized to a hyperplane. The n input features used to train the classifier define an n-dimensional space. A classifier that requires linear separability would impose the requirement that the boilerplate blocks and the non-boilerplate blocks be separable by a hyperplane in this space.

thread, for example, single posts are interleaved with blocks containing user names, timestamps, signatures, etc. Many pages (such as online newspapers) also contain ads in between text blocks.

To conclude this section, a short word is in order as to why the data from the CleanEval shared task (Baroni et al, 2008) was not used to evaluate the system described here. First of all, in the CleanEval shared task, a form of minimal structure detection (headlines, lists, etc.) was required on top of the basic distinction between boilerplate and good text, and that distinction is not required in the COW production system. Furthermore, the CleanEval data were prepared in very unfortunate ways.[8] Most regrettably, the crawl and HTML headers have been stripped, and thus the HTML documents are not intact, including missing doctype definitions. It is virtually impossible to adapt a complex and optimized production system to such artificially crippled documents (see Section 2 for the features, many of which require doctype definitions and intact HTML). Also, the CleanEval gold standard files come in various (original) encodings, and a clean NFC-normalized UTF-8 output (such as produced by my system) therefore cannot be matched properly against the gold standard by the CleanEval evaluation script. Conceptually, the CleanEval annotation guidelines are designed for destructive web page cleaning, while *texrex* is optimized for non-destructive cleaning (Section 4). To give just one example of the consequences, the CleanEval people annotated date strings (such as timestamps in forum posts) as non-boilerplate although they are obviously not human-generated content. Given such conceptual differences, an evaluation of the *texrex* system against CleanEval data would be meaningless. Finally, since technological standards and conventions on the web change quickly (e. g., markup and scripting standards, popular blog and forum software), training and evaluation data from shared tasks organized eight years ago (when, for example, HTML5 documents were non-existing) can only be considered outdated.

The remainder of the paper is structured as follows. Section 2 is dedicated to the feature engineering. In Section 3, I evaluate the classification results obtained with the MLP. I also test whether the method is sensitive to language-specific features and provide a ranking of the features in terms of their usefulness for the task. Turning to more technical evaluations of the method, I compare the MLP to alternative classifiers, and I evaluate the efficiency of the chosen MLP implementation in terms of computing time. I then briefly describe how *texrex* implements non-destructive web corpus cleaning in Section 4 before I sum up the results in Section 5.

2 Feature engineering

In this section, I discuss the engineering of the input features that are fed into the MLP and are used to distinguish between boilerplate and non-boilerplate. Since my implementation is part of a complete web page cleaning system, many of the features can be extracted in the HTML stripping process almost without additional computing effort. The features are calculated for each block of text on the web page. Block segmentation is done in a straightforward way by splitting the content where opening or closing container tags such as `<div>` occur in the markup.[9] The complete set of per-block input features is given in Table 1.

[8] `http://cleaneval.sigwac.org.uk/`

[9] The full list is: `<article>`, `<blockquote>`, `<div>`, `<h1>` – `<h6>`, `<li>`, `<p>`, `<section>`, `<td>`, and their closing counterparts.

Table 1 Features extracted for block-wise boilerplate detection implemented in *texrex*; if not specified otherwise, counts are made within the block; characters are always UTF-8 characters

feature	description	scaling/range
AnchorProp	no. of HREF anchors ÷ no. of characters	$(0,r)$ for some $r<1$
ContArticle, ContBlock, ContDiv, ContH, ContLi, ContP, ContSection, ContTd	the type of enclosing HTML container (`<article>`, `<block>`, `<div>`, `<h>`, `<li>`, `<p>`, `<section>`, `<td>`)	$\{0,1\}$
ContClose	block text follows a closing container tag (= is probably outside a typical container)	$\{0,1\}$
Copy	block contains the copyright sign	$\{0,1\}$
DocMarkupProp	like MarkupProp, but for the whole document	$(0,1)$
DtHtml4, DtHtml5, DtXhtml	declared HTML document type (HTML4, HTML5, XHTML)	$\{0,1\}$
EmailProp	no. of literal email addresses ÷ no. of characters	$(0,r)$ for some $r<1$
EndsPunct	block ends in sentence-ending puncutation	$\{0,1\}$
HashProp	no. of Twitter hashtags ÷ no. of characters	$(0,r)$ for some $r<1$
Length	no. of text characters	clamped to $1,000$ and mapped linearly from $(0,1000)$ to $(0,1)$
LetterProp	no. of letters ÷ no. of characters	$(0,1)$
MarkupProp	no. of markup-encoding characters ÷ no. of all characters	$(0,1)$
NumberProp	no. of number characters ÷ no. of characters	$(0,1)$
OpenProp	no. of opening tags ÷ no. of tags	$(0,1)$
PageProp	block text length ÷ page text length	$(0,1)$
PercDiv	location of the block within the array of blocks of the web page	$(0,1)$
PercText	location of the block text within the text of the web page	$(0,1)$
PunctProp	no. of punctuation characters ÷ no. of characters	$(0,1)$
SentBogus	block contains no sentence-ending punctuation at all	$\{0,1\}$
SentCount	no. of sentences	clamped to 10 and mapped linearly from $(0,10)$ to $(0,1)$
SentLength	average sentence length	clamped to 100 and mapped linearly from $(0,100)$ to $(0,1)$
SkippedDivs	no. of empty blocks directly preceding this block	clamped to 20 and mapped linearly from $(0,20)$ to $(0,1)$
TagProp	no. of tags ÷ no. of characters	$(0,r)$ for some $r<1$
UpperProp	no. of upper-case letters ÷ no. of letters	$(0,1)$
UriProp	no. of literal URLs ÷ no. of characters	$(0,r)$ for some $r<1$
Window1	the same as MarkupProp for a window extended by one block in both directions	$(0,1)$
Window2	the same as MarkupProp for a window extended by two blocks in both directions	$(0,1)$
YearProp	no. of year strings ÷ no. of characters	$(0,r)$ for some $r<1$

All features have to be normalized to fit within either $(-1,1)$ or $(0,1)$, which are the admissible ranges of input values for MLPs. I chose to map all values to $(0,1)$ for purely practical reasons. **Binary features** are trivially represented as either 0 or 1 (Copy, EndsPunct, SentBogus), and **nominal features** (HTML container type and document type) are dummy-coded as arrays of binary features (ContArticle, ContBlock, ContDiv, ContH, ContLi, ContP, ContSection, ContTd, DtHtml4, DtHtml5, DtXhtml). All features that represent true **proportions** (i. e., the number of items of a certain type divided by the number of items of that type's supertype, such as the number of upper-case letters divided by the number of all letters) are intrinsically in $(0,1)$ and do not require further scaling (DocMarkupProp, LetterProp, MarkupProp, NumberProp, OpenProp, PageProp, PunctProp, UpperProp, Window1, Window2). For **count features**, I follow two scaling strategies: If the count intrinsically cannot exceed the number of characters in the block, I divide the count by the number of

Table 2 Numbers of boilerplate and non-boilerplate text blocks in the manually annotated training/evaluation data, including the resulting classification baseline

language	boilerp.	non-boilerp.	baseline
English	13,102	1,303	0.910
French	11,227	1,289	0.897
German	7,912	2,005	0.798
Swedish	9,745	1,505	0.866

characters in the block (AnchorProp, EmailProp, HashProp, TagProp, UriProp, YearProp). For example, because a year string (like *1999* or *2015*) always consists of more than one character, the number of characters in the block can never be lower than the number of year strings in the same block, even if the block exclusively consists of year strings. This is why, in Table 1, the interval for these features is correctly given as: $(0,r)$ for some $r < 1$. For all other counts, I clamp the values between 0 and a fixed constant c and map the results linearly from $(0,c)$ to $(0,1)$ (Length with $c = 1000$, SentCount with $c = 10$, SentLength with $c = 100$, SkippedDivs with $c = 20$). This is justified because it is reasonable to assume that extreme values (for example for block text length) are rare and do not help to differentiate between boilerplate and non-boilerplate.

Finally, the features PercDiv and PercText (where *Perc* stands for *percentile*) encode the **position of the block** in the array of all blocks of the page (PercDiv) and in the total (non-markup) text mass of the page (PercText). They are scaled in a special way because the text in the middle of the web page tends to be the good text, and the text at the margins tends to be boilerplate (Schäfer and Bildhauer, 2013, 53). In order to reduce the amount of nonlinearity, the percentile values are scaled such that the 50th percentile is mapped to 0, and the 0th and the 100th percentile are both mapped to 1.

It should be noticed that the features DtHtml4, DtHtml5, DtXhtml, and DocMarkupProp are features of the whole document, not of single blocks. They were added to allow the classifier to capture interactions between general document properties and features of single blocks.

3 Evaluation

3.1 Training and accuracy of the MLP

Using the FANN library, I trained MLPs on manually annotated datasets using the 37 input features shown in Table 1.[10] Since some of the input features might vary between languages (such as sentence length), evaluations were done separately for English (14,405 blocks), French (12,516 blocks), German (9,917 blocks), and Swedish (11,250 blocks). The data came from large breadth-first crawls performed in the national top-level domains *uk*, *fr*, *de*, and *sv* in 2011 and 2012. The pages used for training and evaluation were sampled randomly from the crawl, but it was made sure that each web host was unique in the sample to increase diversity. Also, the sample was stratified according to the distribution of declared document types in the whole crawl: HTML4 (18.7%), HTML5 (4.3%), XHTML (44.7%) and undeclared/undetected (32.2%). Table 2 shows the distribution of boilerplate and non-

[10] `http://leenissen.dk/fann/wp/`

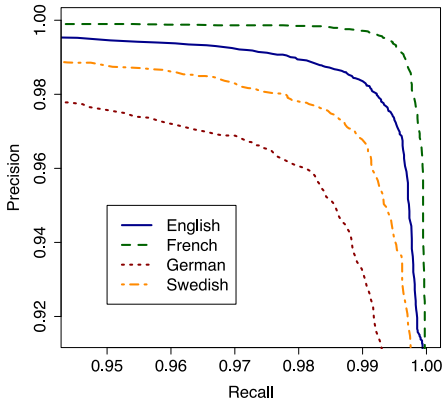

Fig. 1 Precision/recall plot for the four languages (10-fold CV) achieved by manipulating the cutoff used to transform the output of the MLP into a binary decision

Table 3 Thresholds (means over 10 folds) at which *precicion=recall=*f_1 for all languages, as well as the value of f_1 and the proportion of correct classifications at that threshold

language	threshold	f_1	correct
English	0.62	0.986	0.975
French	0.49	0.995	0.990
German	0.60	0.969	0.950
Swedish	0.57	0.979	0.963

boilerplate text blocks in the manually annotated datasets. The baseline accuracy, which can be reached by categorizing all blocks as boilerplate, is also given.

The best training algorithm and the best hidden and output activation functions for the MLP were determined by brute-force permutation of all options on the whole dataset (best mean square error [MSE] after 10,000 epochs). The RPROP algorithm as well as the Gaussian symmetric hidden and linear piecewise symmetric (= simple bounded linear) output activation functions were selected.[11] I used one hidden layer with 18 units by applying the default rule of using half as many hidden units as there are input units. To control for overfitting, I applied 10-fold cross-validation.

I now turn to the accuracy of the classifier when it is used to make a binary decision. In principle, MLPs output real numbers in $(-1,1)$, which means that in order to derive a binary decision (= the block is or is not boilerplate), a threshold has to be determined. Since in the training data, 0 was used for *is not boilerplate* and 1 for *is boilerplate*, actual predicted values are in $(0,1)$ with a minimal spill-over into the negative range. I determined the precision at given recall values achieved by manipulating the threshold in increments of 0.01. Figure 1 shows the results for all four languages.

A key metric is the threshold which results in the optimal trade-off between precision and recall, as well as the corresponding prediction accuracy. For each language, the threshold at which *precicion=recall=*f_1 is shown in Table 3. These thresholds vary within a large

[11] `http://leenissen.dk/fann/html/files/fann-h.html`

Table 4 Evaluation for all languages at a threshold of 0.5 (means over 10 folds) including the baseline (correct decisions achieved by classifying everything as boilerplate) and the proportional reduction of error (PRE) achieved by the MLP compared to the baseline

language	precision	recall	f_1	correct	baseline	PRE
English	0.983	0.990	0.990	0.976	0.910	0.733
French	0.995	0.994	0.994	0.990	0.897	0.903
German	0.963	0.977	0.977	0.952	0.798	0.762
Swedish	0.977	0.983	0.983	0.983	0.866	0.873

interval $(0.49, 0.62)$. However, the prediction accuracy is generally high and the precision and recall curves are very flat between thresholds of 0.3 and 0.7. Therefore, simply setting the threshold at 0.5 leads to the excellent results shown in Table 4.[12] The accuracy is virtually unchanged when moving from the balanced thresholds (Table 3) to the default threshold of 0.5 (Table 4). This indicates that the classifier separates cleanly between boilerplate and non-boilerplate (i. e., output values are close to 0 and 1) and is not biased (i. e., 0.5 is a near-perfect cutoff point). The MLP achieves a proportional reduction of error between 0.733 for English and maximally 0.903 for French. Given the excellent results shown in Table 4, a threshold of 0.5 is used by default in our production system.

3.2 Language-specificity

Since I have used language-specific training data, the overall language-specificity of the approach can also be evaluated. This was done by applying MLPs trained on the full dataset from one language to the full dataset from the other languages as test data. Figure 2 shows the precision/recall graphs of the experiment. In all cases, the MLP trained on one language does not achieve the same precision for the other three languages (at the same recall values), and the drop is substantial, between 0.05 and 0.15, where especially the German data react badly to MLPs trained on other languages. It is thus clear that the COW approach of using language-specific classifiers increases the accuracy a lot (cf. Section 3.3 for a possible explanation in terms of features).

To test whether pooling larger amount of data from several languages could help to partially overcome language-specificity, I trained an MLP on all Germanic data (English, German, and Swedish) and applied it to all four languages, cf. Figure 2. The Germanic languages fare much better than in the language-pair evaluations, while the accuracy on French does not drop noticeably compared to the single language pairings. This is an encouraging result, because it shows that while there is a considerable amount of language-specificity, MLPs trained on large multilingual input datasets might achieve high accuracy on multilingual corpora.[13]

[12] Notice that considering the fact that input decisions were all 0s and 1s, an optimal classifier is expected to perform very well at a threshold of 0.5.

[13] As a reviewer pointed out, it appears that the compatibility between the three Germanic languages is much higher than the compatibility between French and any of the Germanic languages or the pooled Germanic dataset. Therefore, classifiers for language *families* also seem like a viable option to be explored in the future.

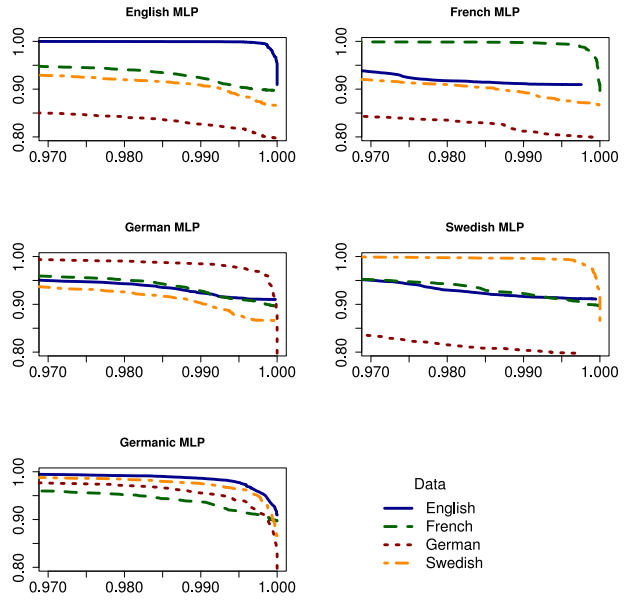

Fig. 2 Precision (y-axis) at recall (x-axis) for MLPs trained on one language and applied to the other languages; the *Germanic MLP* was trained on the pooled English, German, and Swedish data

3.3 Feature ranking

Some approaches to boilerplate detection have used only one feature, for example the text-to-markup ratio (Baroni et al, 2009). In my approach, there is quite a large number of features, many of them inspired by the previous literature. Designers of other production systems might be interested in selecting only a subset of features to increase the performance while keeping most of the accuracy. Therefore, I evaluated and ranked the usefulness of the features by using the Weka toolkit (Hall and Witten, 2011), which offers a similar but less efficient and feature-rich MLP implementation compared to FANN, but which has extensive functionality for experimenting with datasets and different classifiers. (Cf. also Section 3.4 on the otherwise similar performance of the Weka and the FANN MLP). I used Weka's scheme-specific attribute selection and ranking (i. e. the evaluation relative to a given classifier). I used the *ClassifierAttributeEval* evaluator with the *Ranker* search method for a straightforward ranking of the features, always with 10-fold cross-validation.[14]

The results are shown in Figure 3, where the merit average between all languages (plotted as the gray line) is used to sort the features in descending order, and the values for the single languages are plotted as dots. There is a clear split between some very useful features (primarily block length, average sentence length, and text-to-markup ratio, also in extended windows—results which are mostly in line with Kohlschütter et al, 2010 and also Pomikálek,

[14] As the evaluation criterion for calculating the merit, the default was chosen (accuracy for discrete classes and root mean square error [RMSE] for numeric classes). The *leaveOneAttributeOut* option was not used, i. e., each feature was evaluated in isolation. Using *leaveOneAttributeOut* was infeasible because even a single run without cross validation running 16 parallel threads took 15 hours on a powerful machine, and the full evaluation would have taken approximately 25 days.

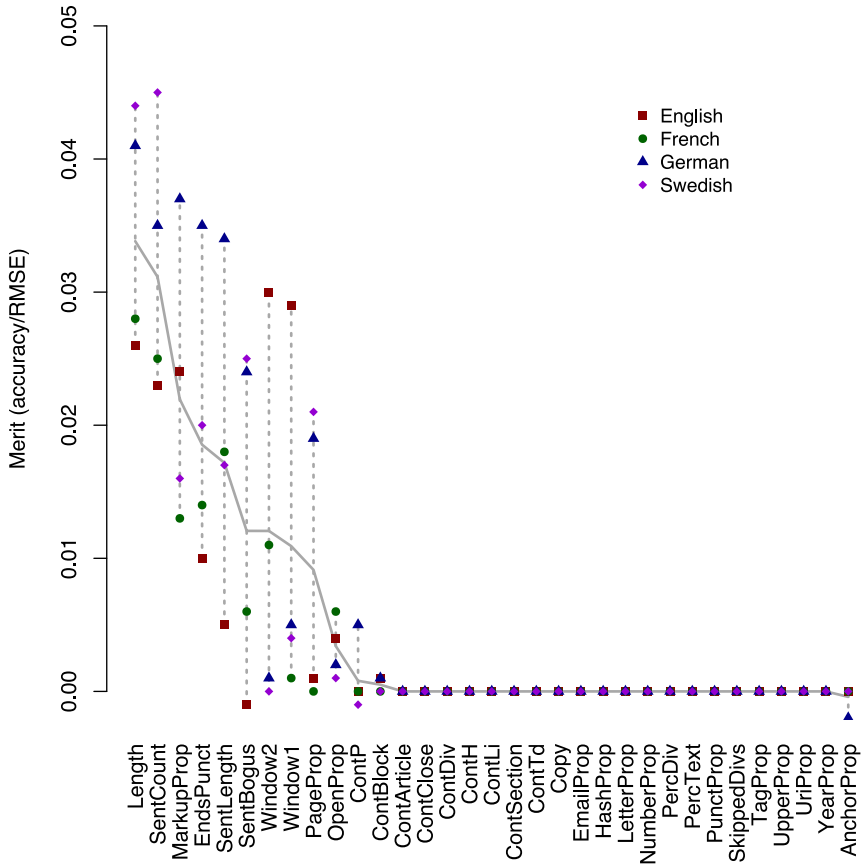

Fig. 3 Average merit (gray line) of the features used and the merit calculated for each language using Weka's *ClassifierAttributeEval* with the MLP

2011), and some marginally useful (mostly markup-related) ones. While the different languages agree with regard to the marginally useful features, there is considerable variance in the merit of the useful features. For example, the syntax-related features Length, SentCount, EndsPunct, and SentBogus are much less informative in the English dataset, while Window1 and Window2 have increased informativity. The opposite is the case for Swedish and less extremely for German. This is a potentially interesting finding from a linguistic perspective because the reason might be that text on English-speaking web pages really has different linguistic properties compared to Swedish and German, such as decreased sentence length (expected to a certain degree) and shorter paragraph length. While it is beyond the scope of this paper to determine whether this is actually the case, the variance in the merit of the important features obviously explains for the language-specificity of the profiles reported in Section 3.2.

3.4 Comparison with other classifiers

In order to verify that the MLP is an optimal choice as a classifier, I compared its performance on all four datasets to that of nine other popular classifiers available in Weka with appropriate mathematical properties. Especially MLPs and Support Vector Machines (SVM) are often treated as classifiers with similar capabilities (including non-linear mappings) but different strengths and trade-offs, cf., e. g., Cortez (2011). The best f_1 scores achieved are shown in Table 5, using the MLP as the reference. 10-fold CV was applied for each combination of a dataset and a classifier. In order to be able to balance both accuracy and time complexity, Table 6 shows the user CPU time consumed for the classification by each algorithm in a similar fashion.[15] The only classifiers which significantly improve the f_1 score on at least three of the datasets are the random forest and the SVM with a PUK kernel. The improvement is minimal (between 0.01 and 0.02). However, both come with significantly increased run times (Table 6). For the SVM, the increase is dramatic while it is modest for the Random Forest. In conclusion, the MLP appears to be an optimal choice with no systematic drawbacks in accuracy or time complexity, which makes it an ideal choice for processing very large datasets.

Table 5 f_1 achieved with different classifiers on all four datasets, 10-fold CV (weighted average); ○/● statistically significant improvement/degradation (corrected two-tailed t-test, $\alpha = 0.05$) over the MLP, which was used as the reference; MLP=Multi-Layer Perceptron, LR=Logistic Regression, IBk=K-Nearest Neighbor, J48=C4.5 Decision Tree, RF=Random Forest, NB=Naive Bayes, SVM/NP=SVM with normalized polynomial kernel, SVM/P=SVM with polynomial kernel, SVM/PUK=SVM with PUK kernel (Üstün et al, 2006), SVM/RBF=SVM with RBF kernel (Chang and Lin, 2011)

dataset	MLP	LR	IBk	J48	RF	NB	SVM/NP	SVM/P	SVM/PUK	SVM/RBF
English	**0.97**	0.96 ●	0.97	0.97 ●	0.98 ○	0.92 ●	0.97 ●	0.96 ●	0.98 ○	0.96 ●
French	**0.96**	0.95 ●	0.97 ○	0.96	0.97	0.91 ●	0.95 ●	0.95 ●	0.97 ○	0.94 ●
German	**0.94**	0.90 ●	0.95	0.94	0.96 ○	0.84 ●	0.92 ●	0.90 ●	0.96 ○	0.89 ●
Swedish	**0.96**	0.95 ●	0.97	0.96	0.97 ○	0.90 ●	0.95 ●	0.95 ●	0.97	0.94 ●

Table 6 CPU time taken for classification with different classifiers on all four datasets, 10-fold CV (weighted average); ○/● statistically significant shorter/longer runtime (corrected two-tailed t-test, $\alpha = 0.05$) compared to the MLP, which was used as the reference; for labels cf. Table 5

dataset	MLP	LR	IBk	J48	RF	NB	SVM/NP	SVM/P	SVM/PUK	SVM/RBF
English	**0.01**	0.00 ○	1.34 ●	0.00 ○	0.03 ●	0.01	0.80 ●	0.00 ○	0.84 ●	0.92 ●
French	**0.01**	0.00 ○	3.05 ●	0.00 ○	0.08 ●	0.02 ●	0.68 ●	0.00 ○	0.75 ●	0.65 ●
German	**0.01**	0.00 ○	0.66 ●	0.00 ○	0.03 ●	0.01	0.99 ●	0.00 ○	0.99 ●	0.72 ●
Swedish	**0.01**	0.00 ○	2.74 ●	0.00 ○	0.07 ●	0.02 ●	0.69 ●	0.00 ○	0.36 ●	0.45 ●

3.5 Real-life efficiency

Taking up the experimental results from the previous section, I now show that the optimized MLP implementation in the FANN library is suitable for the efficient processing of huge datasets in production runs. Results reported here are part of a benchmark of the whole *texrex*

[15] Because training is a one-time process, training times are irrelevant in applications.

system (Schäfer, 2015). The benchmark was performed on a low-end Intel Core i5 processor with four physical cores at 2.3 GHz. I measured the performance (in ms) on 11,781 documents using one thread for reading, four threads for the processing of the documents, and one thread for writing the cleansed output as XML. Over the five runs, the boilerplate detector consumed 3.15 single CPU milliseconds per document on average, resulting in 317 documents per single CPU second or an estimated 27.5 million documents per single CPU day. The whole web page cleaning system with all processors switched on—such as UTF-8 encoding, text quality evaluation, w-shingle creation for near duplicate detection (Broder, 2000), and IP-based geolocalization—consumes 39 CPU milliseconds per document and can thus process 2,215,296 documents per CPU day, even on low-end systems.

4 Non-destructive web corpus cleaning

If we achieve (in the worst case) roughly 95% correct decisions (for example for German, cf. Table 4), we are still in error for 5% of the text blocks. Furthermore, a precision of 0.96 (German, Table 4) means that 4% of our decisions to remove a text block as boilerplate are incorrect. If boilerplate is actually removed from the documents, this leads to fragmented documents in the corpus, i. e., documents where text blocks are missing. For linguistic applications of web corpora that require fully intact documents (e. g., research on information structure, where *givenness* of discourse referents is a crucial notion), this is not acceptable.[16]

To deal with such problems of data integrity, the current version of *texrex* implements non-destructive cleaning wherever possible. For boilerplate removal, this means that the software can be configured to leave all detected boilerplate material in the output and just annotate each text block with the value calculated by the MLP as its *boilerplate score*. Users of the final corpus can then decide to restrict corpus queries to blocks with a low boilerplate score but still see the full context of the matches. Since some corpus query engines such as the IMS OCWB (Evert and Hardie, 2011) cannot efficiently restrict queries to regions annotated with real-valued attributes within a certain interval, *texrex* adds yet another annotation. It is a quantized boilerplate score, a mapping of the real-valued scores from $(0,1)$ to ten equally sized intervals represented alphabetically as letters between *a* and *j*. In systems like the OCWB, queries can be restricted to non-boilerplate regions efficiently by formulating regular expressions on these quantized scores.

5 Summary and outlook

I have shown that efficient and very accurate boilerplate detection for unselectively crawled collections of web pages is possible based on a large set of easily extractable features where some linguistically relevant features such as sentence length play the most important role. A language-specific comparison and evaluation in terms of global accuracy and the role played by single features has—to the best of my knowledge—not been reported in a similar way before. Since I have shown that maximal state-of-the-art accuracy requires language-specific models, this aspect should be considered in the design of production systems. The Multi-Layer Perceptron was shown to be one of the classifiers that make optimal use of the features while allowing for very fast implementations. Classifiers with minimally (albeit significantly) improved performance require significantly higher CPU time. Because features

[16] See also the evaluation of document fragmentation in the dissertation where the jusText algorithm was introduced (Pomikálek, 2011, 52–54).

were also ranked with respect to their usefulness with a clear split between highly useful and relatively irrelevant features, other implementations can benefit from results obtained here by focusing on features that are most informative for the task of boilerplate detection.

In future work, the COW project will continue to use, evaluate, and improve the technology introduced in this paper. For example, we are currently processing CommonCrawl snapshots using efficient *texrex* technology including the boilerplate detector discussed here, which will likely result in corpora of English larger than 100 billion tokens (Schäfer, 2016, to appear). Furthermore, a co-author and I have used distributional semantics as a form of extrinsic evaluation of the classifier presented here. We show that text blocks classified as boilerplate are not as good material for detecting word similarities using distributional semantics as text blocks classified as non-boilerplate. These results will be published elsewhere.

Acknowledgements First of all, I would like to thank the two anonymous LREV reviewers, whose comments greatly contributed to the quality of the paper. I am very grateful to the former student assistants Sarah Dietzfelbinger and Lea Helmers for their work on the annotation of the training data. I would also like to thank Felix Bildhauer for ongoing collaborative work on the COW corpora since 2011 and Stefan Müller for his support of the COW project since 2011.

References

Baroni M, Chantree F, Kilgarriff A, Sharoff S (2008) CleanEval: A competition for cleaning webpages. In: Calzolari et al (2012), pp 638–643

Baroni M, Bernardini S, Ferraresi A, Zanchetta E (2009) The WaCky Wide Web: A collection of very large linguistically processed web-crawled corpora. Language Resources and Evaluation 43(3):209–226

Bauer D, Degen J, Deng X, Herger P, Gasthaus J, Giesbrecht E, Jansen L, Kalina C, Krüger T, Märtin R, Schmidt M, Scholler S, Steger J, Stemle E, Evert S (2007) Filtering the internet by automatic subtree classification. In: Fairon et al (2007), pp 111–122

Biemann C, Heyer G, Quasthoff U, Richter M (2007) The Leipzig Corpora Collection – monolingual corpora of standard size. In: Proceedings of Corpus Linguistic 2007, University of Birmingham, Birmingham

Broder AZ (2000) Identifying and filtering near-duplicate documents. In: Giancarlo R, Sanko D (eds) Proceedings of Combinatorial Pattern Matching, Berlin, pp 1–10

Calzolari N, Choukri K, Declerck T, Doğan MU, Maegaard B, Mariani J, Odijk J, Piperidis S (eds) (2012) Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC '12), European Language Resources Association (ELRA), Istanbul

Chang CC, Lin CJ (2011) LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2:1–27

Cortez P (2011) Data mining with multilayer perceptrons and support vector machines. In: Holmes DE, Jain LC (eds) Data Mining: Foundations and Intelligent Paradigms. Volume 2: Statistical, Bayesian, Time Series and other Theoretical Aspects, Springer, Berlin, pp 9–23

Evert S, Hardie A (2011) Twenty-first century corpus workbench: Updating a query architecture for the new millennium. In: Proceedings Corpus Linguistics 2011, University of Birmingham, Birmingham

Fairon C, Naets H, Kilgarriff A, de Schryver GM (eds) (2007) Building and Exploring Web Corpora: Proceedings of the 3rd Web as Corpus Workshop (Incorporating CLEANEVAL), Presses universitaires de Louvain, Louvain

Finn A, Kushmerick N, Smyth B (2001) Fact or fiction: Content classification for digital libraries. In: DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries

Gallé M, Renders JM (2014) Boilerplate detection and recoding. In: de Rijke M, Kenter T, de Vries A, Zhai CX, de Jong F, Radinsky K, Hofmann K (eds) Advances in Information Retrieval – 36th European Conference on IR Research, ECIR, Springer, Berlin, pp 462–467

Grossberg S (1973) Contour enhancement, short-term memory, and constancies in reverberating neural networks. Studies in Applied Mathematics 52:213–257

Hall M, Witten IH (2011) Data mining: practical machine learning tools and techniques, 3rd edn. Kaufmann, Burlington

Kohlschütter C, Fankhauser P, Nejdl W (2010) Boilerplate detection using shallow text features. In: Davison BD, Suel T, Craswell N, Liu B (eds) WSDM '10: Proceedings of the Third ACM International Conference on Web Search and Data Mining, ACM, New York, pp 441–450

Marek M, Pecina P, Spousta M (2007) Web page cleaning with Conditional Random Fields. In: Fairon et al (2007), pp 155–162

Minsky ML, Papert SA (1988) Perceptrons. MIT Press

Neunerdt M, Reimer E, Reyer M, Mathar R (2015) Enhanced web page cleaning for constructing social media text corpora. In: Kim KJ (ed) Information Science and Applications, Springer, Berlin, pp 665–672

Nissen S (2003) Implementation of a Fast Artificial Neural Network Library (FANN). Tech. rep., Datalogisk Institut Københavns Universitet, Copenhagen

Pasternack J, Roth D (2009) Extracting article text from the web with maximum subsequence segmentation. In: Quemada J, León G, Maarek Y, Nejdl W (eds) WWW '09: Proceedings of the 18th International Conference on World Wide Web, ACM, Madrid, pp 971–980

Pomikalek J, Rychly P, Kilgarriff A (2009) Scaling to billion-plus word corpora. Research in Computing Science 41, special issue: Advances in Computational Linguistics

Pomikálek J (2011) Removing boilerplate and duplicate content from web corpora. PhD thesis, Masaryk University Faculty of Informatics, Brno, URL `http://is.muni.cz/th/45523/fi_d/phdthesis.pdf`

Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323:533–536

Schäfer R (2015) Processing and querying large web corpora with the COW14 architecture. In: Bański P, Biber H, Breiteneder E, Kupietz M, Lüngen H, Witt A (eds) Proceedings of Challenges in the Management of Large Corpora 3 (CMLC-3), UCREL, Lancaster

Schäfer R (2016, to appear) CommonCOW: massively huge web corpora from CommonCrawl data and a method to distribute them freely under restrictive EU copyright laws. In: Calzolari N, Choukri K, Declerck T, Doğan MU, Maegaard B, Mariani J, Odijk J, Piperidis S (eds) Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC '16), European Language Resources Association (ELRA), Portorož

Schäfer R, Bildhauer F (2012) Building large corpora from the web using a new efficient tool chain. In: Calzolari et al (2012), pp 486–493

Schäfer R, Bildhauer F (2013) Web Corpus Construction. Synthesis Lectures on Human Language Technologies, Morgan and Claypool, San Francisco

Spousta M, Marek M, Pecina P (2008) Victor: The web-page cleaning tool. In: Evert S, Kilgarriff A, Sharoff S (eds) Proceedings of the 4th Web as Corpus Workshop, European Language Resources Association (ELRA), Marrakech, pp 12–17

Üstün B, Melssen WJ, Buydens LM (2006) Facilitating the application of Support Vector Regression by using a universal Pearson VII function based kernel. Chemometrics and Intelligent Laboratory Systems 81:29–40