# Efficient High-precision Boilerplate Detection Using Multilayer Perceptrons (Extended abstract of unpublished paper)

**Roland Schäfer**

German Grammar
Freie Universität Berlin
Habelschwerdter Allee 45, 14195 Berlin
roland.schaefer@fu-berlin.de

**Abstract**

Removal of boilerplate is among the essential tasks in web corpus construction and web indexing. In this paper, we present an improved machine learning approach to general-purpose boilerplate detection for languages based on (extended) Latin alphabets (easily adaptable to other scripts). We keep it highly efficient (around 320 documents per single CPU core second) by using an optimized Multilayer Perceptron implementation while achieving around 95% correct classifications (*Precision*, *Recall*, and $F_1$ score over 0.95) by extracting suitable text block-internal features. We finally compare the performance of the Multilayer Perceptron to that of other classifiers such as Support Vector Machines.

**Keywords:** boilerplate detection, Multilayer Perceptron, web corpus construction, MLP vs. SVM

## 1. Previous Approaches and the Notion of Boilerplate

For 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), Multilayer Perceptrons [MLP] (Schäfer and Bildhauer, 2012). Bauer et al. (p. 120) report $F_1 = 0.652$, Spousta et al. report $F_1$ scores around $0.8$ or below (p. 16) and Pasternack and Roth report $F_1$ scores above $0.9$, even $0.95$ (p. 977–979).[1] Finally, Schäfer and Bildhauer, training the MLP on nine very simple text block-internal features, achieve an acceptable *Precision* of $0.83$ at a quite mediocre *Recall* of $0.68$, resulting in an $F_1$ score of $0.75$.[2] While the accuracy reported by Schäfer and Bildhauer is below the state of the art, the advantage of the MLP (especially when using the FANN library, cf. below) is that it is computationally very efficient. Section 2. will show how a large number of easily extractable features can be used to improve their MLP-based approach.

A major problem in designing and evaluating boilerplate detectors is the definition of boilerplate itself. We define boilerplate as all linguistic and non-linguistic material which remains after HTML stripping, and which does not belong to the (or one of many) coherent block(s) of text on the web page.[3] Under this definition, headlines and abstracts relating directly to a block of connected text are not treated as boilerplate, but similar blocks relating to the whole web site are. Blocks just containing date strings, user IDs, etc., are counted as boilerplate. Different setups might involve substantially different definitions of boilerplate as well as different definitions of the removal task itself. For example, in the CleanEval shared task (Baroni et al., 2008), minimal structure detection (headlines, lists, etc.) was required on top of the basic distinction between boilerplate and good text, which makes comparisons with setups like ours, where such distinctions are not required, more or less inappropriate. More details about our definition of boilerplate will be in the final paper, and the training data set will be made available publicly. The method described here is implemented in the current version of our open-source web page cleaning system `texrex`.[4]

## 2. Features for Very High Precision Boilerplate Detection

In principle, an MLP (Grossberg, 1973; Rumelhart et al., 1986; Minsky and Papert, 1988) is ideally suited for the task of learning a binary classification based

[1] We acknowledge that it is generally unfair to cite single scores, since all authors perform quite differentiated evaluations with diverse results, some higher than the ones given here. The values cited here are intended for a rough orientation only, and readers are advised to refer to the papers for full evaluations.

[2] The authors do not report all of these figures in the paper. They are reported in their book on web corpus construction, however (Schäfer and Bildhauer, 2013, p. 56).

[3] Splitting the page into blocks is done in a very simple way by interpreting the text contents of certain HTML containers like `<div>` and `<p>` as blocks.

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

on an array of input values. We extended Schäfer and Bildhauer's approach by adding a rich set of features, many inspired by Spousta et al.'s comprehensive feature set (Spousta et al., 2008). Like Schäfer and Bildhauer, we used the FANN library implementation (Nissen, 2003) and made per-block decisions. We thus do not use a single-window approach as used, for example, by the WaCky project (**?**). Since our implementation is part of a complete web page cleaning system, many of these features can be extracted in the HTML stripping process almost for free. Among the features are (where *text* always refers to the raw text stripped from HTML markup):

1. **Markup-related features**
   (a) markup proportion in the block (bytes)
   (b) same as (1a) for a window extended by one block in both directions
   (c) same as (1a) for a window extended by two blocks in both directions
   (d) type of enclosing container tag (`<article>`, `<div>`, `<h>`, etc.)
   (e) block is outside any expected HTML container (`<p>`, `<div>`, etc.) {0,1}
   (f) count of empty blocks before this one
   (g) count of anchor tags in block
2. **Link-related features**
   (a) count of email addresses in block
   (b) count of literal URLs in block
   (c) count of Twitter hashtags in block
3. **Text length and position**
   (a) block text length in UTF-8 characters
   (b) block text length ÷ document text length (UTF-8)
   (c) position (percentile) in the text mass of the document
   (d) position (percentile) in the document's array of blocks
4. **Graphemic features**
   (a) proportion of punctuation characters
   (b) proportion of letter characters
   (c) proportion of numeric characters
   (d) proportion of uppercase letters
   (e) occurrence of the copyright character {0,1}
5. **Linguistic features**
   (a) average sentence length
   (b) sentence count
   (c) last character is punctuation character {0,1}
6. **Whole-document weighting features**
   (a) declared doctype for document {HTML 4, HTML 5, XHTML, undeclared}
   (b) overall proportion of markup, i. e., (1a) for the whole document

All features have to be scaled to fit within $(-1, 1)$ or $(0, 1)$, the range of input values expected by the MLP. For all proportions and the position/percentile values (3c), this is trivially the case. For count features, we follow two scaling strategies: If the count intrinsically cannot exceed the number of characters in the block, we simply divide the count by the number of characters (1g,2a,2b,2c). For all others, we divide the actual count by a constant and clamp values to a maximum of 1.0 (1f,1f,3a,5a,5b). Since by assumption the text in the middle of the web page tends to be the good (i. e., non-boilerplate) text, we scale the percentile values such that the 50th percentile is mapped to 0.0, and the 0th and the 100th percentile are both mapped to 1.0 (3c). The nominal features (1d,6a) were encoded as arrays of binary features.
Figure 1 shows how the good text is actually centered around a central percentile in the analysis of an actual web page.

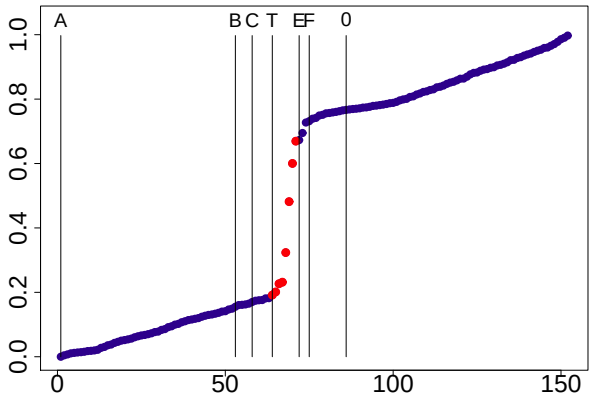


Figure 1: Cumulative distribution of the text mass of a web page over the HTML input by blocks; A, B, C, E, F, and 0 are boilerplate regions, T is the non-boilerplate text region (Schäfer and Bildhauer, 2013, 53).

## 3. Training and Evaluation

The MLP was trained on a manually annotated data set of 9,917 blocks from a large breadth-first crawl of the German top-level domain from late 2011, with a final set of 37 features. The pages were randomly sampled, but it was made sure that each web host was unique

in the sample to guarantee diversity. Also, the sample was stratified according to the distribution of declared document types in the crawl: HTML4 18.7%, HTML5 4.3%, XHTML 44.7% and undeclared 32.2% in the crawl. The documents contained 9,917 blocks in total. To control for overfitting, we used a 10-fold cross-validation, where in each fold 992 blocks were retained for testing, and training was done on 8,925 blocks. The best training algorithm and the best hidden and output activation functions were determined by brute-force permutation of all options on the whole data set (best mean square error [MSE] after 10,000 epochs). The RPROP algorithm as well as the Gaussian Symmetric (hidden) and Linear Piece Symmetric (output) activation functions were selected.[5] We used one hidden layer with 18 units by applying the default rule of choosing half as many hidden units as there are input units. The development of the training and test MSE over 50,000 training epochs is shown in Figure 2. All evaluations were performed using MLPs trained for 50,000 epochs, taking snapshots at intervals of 1,000 epochs, and finally using the snapshot with the lowest MSE on the training data.

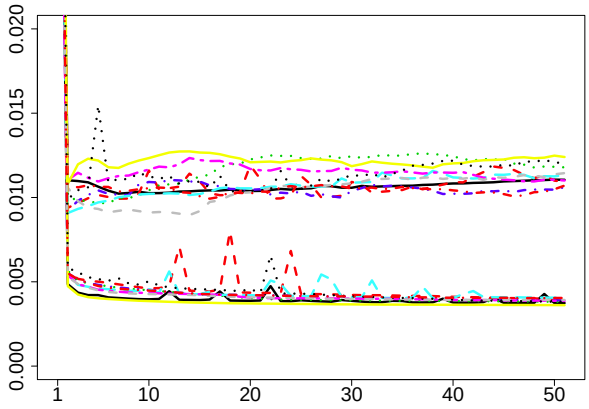


Figure 2: Training (lower lines) and testing (upper lines) mean square error (y axis) for the 10 folds after $n$ thousand epochs (x axis). After the 50,000th epoch, the mean training MSE across the folds is 0.00387, the mean testing MSE is 0.01128.

The MLP outputs reals in $(-1, 1)$, so in order to derive a binary decision, 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 mostly in $(0, 1)$. We determined the values for *Correct (Predictions)*, *Precision*, *Recall*, and $F_1$ for thresholds in $(-0.1, 1)$ at threshold increments of 0.01. Figure 3 plots the means of the results (network trained on the 8,925 training blocks applied to the retained 992 testing blocks) of all folds. Table 1 shows the mean results at the threshold where *Precision*=*Recall*=$F_1$.

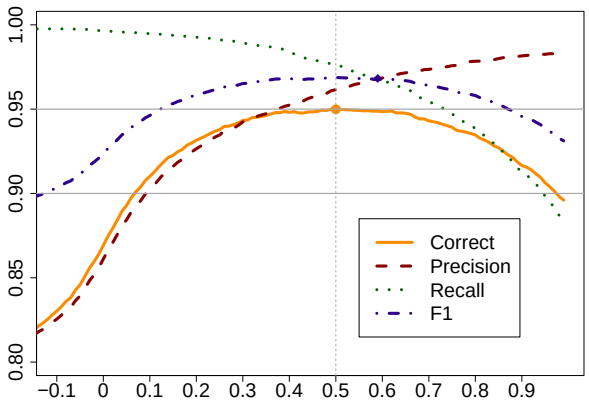


Figure 3: Mean (arithmetic) of *Correct*, *Precision*, *Recall*, and $F_1$ (y axis) over all 10 folds at thresholds in $(-0.1, 1)$ (x axis). The diamond marks the threshold where *Precision*=*Recall*=$F_1$=0.968 (threshold 0.59), the dot marks the threshold with the highest prediction accuracy of 0.951 (threshold 0.5).

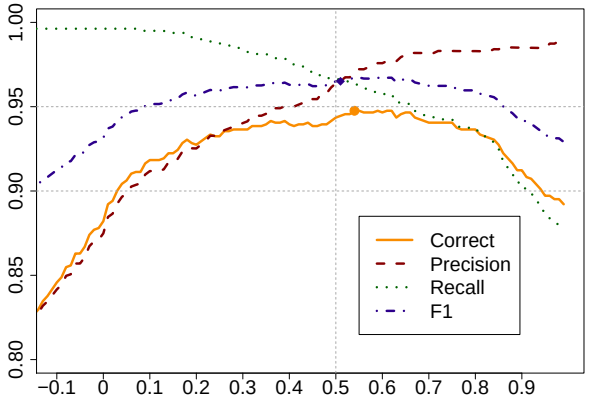


Figure 4: Same as Figure 3 for fold 9.

These "balanced" thresholds show quite some variance between the folds, they are in $(0.51, 0.74)$. The thresholds at which the maximal number of correct predictions is achieved (to be shown in tabular form in the final paper) vary even to a greater extent and are in $(0.39, 0.72)$. However, since the prediction accuracy is generally high and the *Precision* and *Recall*

[5] For the definitions of these functions as used by the FANN library, cf. `http://leenissen.dk/fann/html/files/fann-h.html`.

| | Thresh. | $F_1$ | *Corr.* |
|---|---|---|---|
| 1 | 0.66 | 0.968 | 0.950 |
| 2 | 0.52 | 0.973 | 0.956 |
| 3 | 0.51 | 0.966 | 0.945 |
| 4 | 0.59 | 0.971 | 0.953 |
| 5 | 0.53 | 0.967 | 0.947 |
| 6 | 0.58 | 0.966 | 0.946 |
| 7 | 0.74 | 0.963 | 0.941 |
| 8 | 0.51 | 0.969 | 0.951 |
| 9 | 0.62 | 0.968 | 0.952 |
| 10 | 0.62 | 0.966 | 0.946 |
| $AM$ | 0.59 | 0.968 | **0.948** |
| $HM$ | 0.58 | 0.968 | **0.948** |

Table 1: Means (AM = arithmetic, HM = harmonic) of the "balanced" thresholds (= thresholds where *Precision* = *Recall* = $F_1$) across the 10 folds including accuracy values at this threshold.

| | *Prec.* | *Rec.* | $F_1$ | *Corr.* |
|---|---|---|---|---|
| 1 | 0.950 | 0.957 | 0.981 | 0.969 |
| 2 | 0.958 | 0.971 | 0.978 | 0.974 |
| 3 | 0.944 | 0.964 | 0.966 | 0.965 |
| 4 | 0.957 | 0.967 | 0.980 | 0.973 |
| 5 | 0.944 | 0.960 | 0.970 | 0.965 |
| 6 | 0.950 | 0.960 | 0.977 | 0.969 |
| 7 | 0.956 | 0.961 | 0.985 | 0.973 |
| 8 | 0.952 | 0.968 | 0.971 | 0.969 |
| 9 | 0.945 | 0.955 | 0.974 | 0.964 |
| 10 | 0.954 | 0.963 | 0.981 | 0.972 |
| $AM$ | 0.962 | 0.976 | 0.969 | **0.951** |
| $HM$ | 0.962 | 0.976 | 0.969 | **0.951** |

Table 2: Evaluation for all folds at a threshold of 0.5.

curves are not very steep, setting the threshold at 0.5 (where the average maximal prediction accuracy lies) gives the excellent results in Table 2.

Considering that the input decisions were all 0s and 1s, the classifier is expected to perform very well at a threshold of 0.5. Even if the system should turn out to perform slightly less well in actual production runs compared to the performance suggested by Table 2, we have shown that state-of-the-art high precision general-purpose boilerplate detection depends highly on the selection of good features, cf. the substantial improvement over earlier results using the same classifier (Schäfer and Bildhauer, 2012). The MLP outperforms Spousta et al.'s CRF implementation (Spousta et al., 2008) by around almost 0.15 on all evaluation metrics.

| Method | *Prec.* | *Rec.* | $F_1$ | *Corr.* |
|---|---|---|---|---|
| SVM (PUK)[6] | 0.957 | 0.957 | 0.957 | **0.957** |
| MLP | 0.944 | 0.945 | 0.945 | **0.945** |
| SVM (RBF)[7] | 0.901 | 0.905 | 0.901 | **0.905** |
| Log. Reg. | 0.901 | 0.904 | 0.902 | **0.904** |

Table 3: Overview of best achievable results with different classifiers.

## 4. Comparison with other Classifiers

For various reasons, Support Vector Machines (SVM) are sometimes regarded as a superior option compared to MLPs. To test whether the MLP chosen for our tool kit is a good choice (given the available features), we tested several alternative popular classifiers, including SVM, on the data set. We used the Weka tool kit (Hall and Witten, 2011), always applying 10-fold cross validation. The maximal correct predictions achieved are summarized in Table 3 (full details about the configurations will be in the final paper). First of all, we confirm the accuracy of the FANN evaluation by the similar result from the Weka MLP (a full comparison of configurations will be in the final paper). Furthermore, the SVM delivers roughly the same accuracy as the MLP, at least if a Pearson VII universal kernel is available, which helps the SVM to even slightly outperform the Weka MLP, although not the FANN MLP. Again, this is a confirmation that it is rather the selection of the right features that helps to achieve around 95% correct predictions, and that the choice of either MLP or SVM (with the right kernel) is secondary.

## 5. Efficiency

Finally, we show that the optimized MLP implementation in the FANN library is suitable for efficient processing of huge data sets. We performed a simple benchmark on a standard Intel Core i5 processor with four physical cores at 2.3 GHz running a plain GNU/Linux kernel (3.2.0-53-generic SMP) under Ubuntu 12.04 LTS. We switched off all configurable algorithms of our web page cleaning tool, leaving only basic processing (HTML stripping, conversion to UTF-8, conversion of HTML entities) on. We measured the performance (in ms) over 11,781 documents read from an ARC file using one thread for reading, four threads for the processing of the documents, and one thread for writing the cleansed output as XML (five runs with at least two minute cooling off in between). Then, switching only the boilerplate detector, we measured the difference in processing time. Over the five

[6]Using Weka's PUK kernel (Üstün et al., 2006).

[7]Using libSVM's RBF kernel (Chang and Lin, 2011).

runs, the boilerplate detector (on average) consumed 3.15 single CPU core milliseconds per document, resulting in an efficiency of $\sim$ 317 documents per single CPU core second or an estimated 27.5 million documents per single CPU core day.

## 6. Summary and Outlook

We have shown that efficient and very high-precision boilerplate detection based on easily extractable but numerous document internal features is possible. Since the method was tested only on German web pages so far, future work will center around an evaluation of the method on English, French, Spanish, and Swedish web pages. Especially, we are interested to see how well MLPs trained on documents in one language perform on other graphemically similar languages. Also, integrating the software in an Apache Hadoop-based massively parallel architecture for web corpus construction (Biemann et al., 2013) is among our future goals.

## 7. References


Marco Baroni, Francis Chantree, Adam Kilgarriff, and Serge Sharoff. 2008. CleanEval: A competition for cleaning webpages. In *Proceedings of LREC 06*, pages 638–643, Marrakech. ELRA.

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. 2007. Filtering the internet by automatic subtree classification. In Fairon et al. (Fairon et al., 2007), pages 111–122.

Chris Biemann, Felix Bildhauer, Stefan Evert, Dirk Goldhahn, Uwe Quasthoff, Roland Schäfer, Johannes Simon, Leonard Swiezinski, and Torsten Zesch. 2013. Scalable construction of high-quality web corpora. *Special issue of JLCL*. To appear.

Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27.

Cédrick Fairon, Hubert Naets, Adam Kilgarriff, and Gilles-Maurice de Schryver, editors. 2007. *Building and Exploring Web Corpora: Proceedings of the 3rd Web as Corpus Workshop (Incorporating CLEANEVAL)*, Louvain. Presses universitaires de Louvain.

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

Mark Hall and Ian H. Witten. 2011. *Data mining: practical machine learning tools and techniques*. Kaufmann, Burlington.

Michal Marek, Pavel Pecina, and Miroslav Spousta. 2007. Web page cleaning with Conditional Random Fields. In Fairon et al. (Fairon et al., 2007), pages 155–162.

Marvin L. Minsky and Seymour A. Papert. 1988. *Perceptrons*. MIT Press.

Steffen Nissen. 2003. Implementation of a Fast Artificial Neural Network Library (FANN). Technical report, Datalogisk Institut Københavns Universitet, Copenhagen.

J. Pasternack and D. Roth. 2009. Extracting article text from the web with maximum subsequence segmentation. In *Proceedings of the 18th international conference on World Wide Web*, pages 971–980.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature*, 323:533–536.

Roland Schäfer and Felix Bildhauer. 2012. Building large corpora from the web using a new efficient tool chain. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, pages 486–493, Istanbul. ELRA.

Roland Schäfer and Felix Bildhauer. 2013. *Web Corpus Construction*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool, San Francisco.

Miroslav Spousta, Michal Marek, and Pavel Pecina. 2008. Victor: The web-page cleaning tool. In Stefan. Evert, Adam Kilgarriff, and Serge Sharoff, editors, *Proceedings of the 4th Web as Corpus Workshop*, pages 12–17, Marrakech.

Bülent Üstün, Willem J. Melssen, and Lutgarde M.C. Buydens. 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.