Turn the Page: Automated Traversal of Paginated Websites*

Tim Furche, Giovanni Grasso, Andrey Kravchenko, and Christian Schallhart

Department of Computer Science, Oxford University,
Wolfson Building, Parks Road, Oxford OX1 3QD
firstname.lastname@cs.ox.ac.uk

Abstract. Content-intensive web sites, such as Google or Amazon, paginate their results to accommodate limited screen sizes. Thus, human users and automatic tools alike have to traverse the pagination links when they crawl the site, extract data, or automate common tasks, where these applications require access to the entire result set. Previous approaches, as well as existing crawlers and automation tools, rely on simple heuristics (e.g., considering only the link text), falling back to an exhaustive exploration of the site where those heuristics fail. In particular, focused crawlers and data extraction systems target only fractions of the individual pages of a given site, rendering a highly accurate identification of pagination links essential to avoid the exhaustive exploration of irrelevant pages. We identify pagination links in a wide range of domains and sites with near perfect accuracy (99%). We obtain these results with a novel framework for web block classification, BER$_y$L, that combines rule-based reasoning for feature extraction and machine learning for feature selection and classification. Through this combination, BER$_y$L is applicable in a wide settings range, adjusted to maximise either precision, recall, or speed. We illustrate how BER$_y$L minimises the effort for feature extraction and evaluate the impact of a broad range of features (content, structural, and visual).

1 Introduction

Pagination is as old as written information. On the web, no physical limitations force us to paginate articles or result lists. Nevertheless, pagination is just as ubiquitous – for traditional reasons (bookmarking), as well as technical (reducing bandwidth and latency), noble (avoiding information overload), and not quite so noble (maximising page views) ones.

If we are interested in the entire result, e.g., to search through a complete article or to count the number of matching products, pagination quickly becomes a nuisance. This is even more true for automated tools which are interested in extracting all results. Unfortunately, pagination is not a core concept of HTML or the web, but is simulated through links. The ability to distinguish such *pagination links* with *high accuracy* would be a significant advantage for focused crawlers and automated data extraction. Given

* The research leading to these results has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement DIADEM, no. 246858.

a reliable method for recognising pagination links, once we reach a page containing relevant, but paginated data, we can crawl, extract, or traverse the entire result set with minimal effort.

Nevertheless, existing crawlers and block classification tools do not incorporate high-accuracy identification of pagination links: **(1)** Traditional crawlers are unfocused and explore all pages of a site, therefore not requiring link or page classification. Focused crawlers [1–3, 6, 8, 11–13, 15] and data extraction tools [4, 7, 18] considered this a side issue, mostly addressed by manual supervision or simple heuristics combined with an exhaustive fallback strategy. For focused crawlers, this is also influenced by the fact that they already need a mechanism to distinguish relevant pages from irrelevant ones and can apply it also for pagination links. **(2)** Block classification approaches are more concerned with identifying large page segments, such as navigation menus or advertisements, and are typically tailored for speed over accuracy.

We introduce a novel method for *high accuracy identification of pagination links,* using a novel, flexible framework for block classification called BER_yL, an abbreviation for **B**lock classification with **E**xtraction **R**ules and machine **L**earning. BER_yL combines *logical rules* for *feature extraction* on one hand with *machine learning* for *feature selection and classification* on the other hand. This approach enables different trade-offs between speed, precision, and recall.

Aside of applications in web automation, pagination links are an interesting example for block classification: They appear in a large variety and have themselves a rather simple structure. Together, this makes it hard to identify them accurately with a single class of features. Here, we consider content features, e.g., whether a link text is a number, visual features, e.g., whether a block is in the upper third of a page, or structural ones, such as whether a link is the descendant of a `div`. Figure 1 shows the pagination links on the second page of a paginated result set. There are two *non-numeric* pagination links (‹ and ›) that lead to the next and previous page and 13 *numeric* links leading to the *i*-th page. For the identification of pagination links we are particularly interested in determining those links that lead to next page in the series, here › and 3 (immediate pagination links). Unfortunately, simple heuristics are not sufficient to achieve high accuracy for the identification of pagination links (as verified in Section 6):

1. *Content features:* Checking the presence of certain keywords or tokens, such as “next” and “›”, yields low precision and recall, unable to identify viable links among numeric or image-only pagination links. Since this is a simple baseline heuristic used in some focused crawlers, we investigate it in Section 6, extended with numeric pagination links and some other refinements. We show that it can achieve fairly high precision, but recall remains unacceptable.
2. *Structural features:* It is tempting to assume that such lists of pagination links will be contained in easily identifiable, repeatable HTML structures. But as almost every conceivable abuse of HTML structures indeed occurs in the web, any purely structural approach is limited in its accuracy. Again, we validate in Section 6 that, at least simple, heuristics based on structural features also fail (whether combined with content features or not).
3. *Visual proximity features:* To counter HTML abuse, many data extraction and block classification tools incorporate visual features. We could analyse the visual prox-

Fig. 1: Numeric (1, 3 – 14) and non-numeric (‹ and ›)

imity of links just as well, but although relatively sophisticated, such features fail to contribute significantly towards high accuracy results, either alone or combined with content or structural features, as discussed in Section 6.

4. *Page position features:* Pagination links usually appear on top or below the paginated information. Thus, a link's relative position on a page or whether it occurs in the first screen (at a typical resolution) might seem to constitute a promising feature. Unfortunately, advertisements or navigation headers and footers affect these features significantly (and reliably recognising those blocks is anything but easy). For simple page position features, Section 6 again shows that neither alone nor combined with either content or structural features high accuracy is achieved.

Fortunately, BER$_y$L makes it very easy to extract a large set of features through declarative (Datalog) extraction rules. On the extracted feature model, we employ standard machine learning techniques for automated feature selection and classification. Note that the classifier only needs to be trained *once for all* pages, as demonstrated in Section 6. Our approach is resilient against significant changes in the way HTML is used on the web, as a single re-training of the classifier will suffice for the system to keep functioning properly. With this combination, we achieve near perfect accuracy for identifying pagination links, yet remain comparable in performance to other block classification methods that incorporate visual features, identifying pagination links on most pages within a few seconds. All these approaches are dominated in performance by the underlying page rendering, which is necessary to extract visual features and becomes unavoidable even for content and structural features, as scripted pages reshape the web today. Furthermore, this is easily offset by the fact that a high-accuracy identification of pagination links avoids following many irrelevant links without missing any relevant data, e.g., in focused crawlers or data extraction.

To summarise, we present a novel method for identifying pagination links at a far higher accuracy than previous methods:

1. In BER$_y$L, a **flexible block classification framework**, we combine easy, declarative feature extraction with automatic feature selection and classification (Section 4).
2. Specifically, we define a *comprehensive set of features*, spanning content, structural, and visual features through declarative Datalog rules (Section 5) and BER$_y$L's flexible *feature template rules*.
3. With such a feature set, a *small training set* suffices, yet yields a classifier that uses nearly all these features for effective classification (Section 5.1). The training process is supported by BER$_y$L-Trainer, a visual tool for labeling blocks on web pages.
4. In an *extensive evaluation* (Section 6), we show that this approach achieves **near perfect accuracy** (99%), yet remains highly scalable.
5. We investigate the impact (Section 6) of *removing some features* from the full model and show that neither content, visual, nor structural features alone suffice.

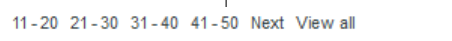
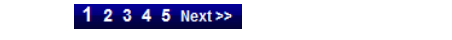
	Website	n	n_1	n_2	P	R	Screenshot
Real estate	FindAProperty	370	1	1	1	1	
	Zoopla	332	1	1	1	1	
	Savills	234	2	2	1	1	
Cars	Autotrader	262	2	2	1	1	
	Motors	472	2	2	1	1	
	Autoweb	103	2	2	1	1	
Retail	Amazon	448	1	1	1	1	
	Ikea	290	2	0	1	1	
	Lands' End	527	2	2	1	1	
Forums	TechCrunch	279	0	1	1	1	
	TMZ	200	2	2	1	1	
	Ars Technica	341	2	2	1	1	

Table 1: Sample pages

2 Pagination Links: A Survey

To motivate the need for pagination link detection and give an impression of some of the issues involved, Table 1 presents a selection of popular websites from the evaluation corpus (on all of which we happen to identify pagination links with 100% precision and recall). n is the number of links on the result page, n_1 (n_2) the number of immediate numeric (non-numeric) pagination links on the page, and P, R are precision and recall for our approach.[1] For each website we also present a screenshot of either its pagination links or a potential false positive. Even in this small sample of webpages, we can observe the diversity of pagination links: Only six of the twelve websites have a typical pagination link layout (non-numeric link containing a NEXT keyword and a list of numeric links with the current page represented as a non-link). Some of the challenges evident from this table are:

1. For FindAProperty and IKEA the index of the current page is a link and thus we need to consider, e.g., its style to distinguish it from the other links.
2. For Zoopla the "50" for the results per page can be easily mistaken for an immediate numeric pagination link.
3. For Savills, numeric links come as intervals. However, our NUMBER annotations also cover numeric ranges (as well as "2k" or "two").
4. For Amazon the result page contains a confusing scrollbar for navigating through related products (right screenshot).
5. For Lands' End the non-numeric pagination link is an image. However, our approach classifies it correctly, based on the context and attribute values.

[1] Precision is the percentage of true positives among the nodes identified as pagination links, recall the percentage of identified pagination links among all pagination links (and thus lower recall means more false negatives).

6. TechCrunch contains a single isolated non-numeric pagination link, that we are able to identify due to the keyword present in its text and the proximity to "Page 1".
7. TMZ has a pagination link that carries both a NEXT and a NUMBER annotation. From the context, we nevertheless identify it correctly as non-numeric.

3 Related Work

Our work is mostly related to web-block classification. Although not directly addressing our problem, fields such as focused (or topical) crawling and web data extraction try to recognise relevant links to follow, such as pagination links. *Focused crawlers* [2, 1, 13, 3, 8], tailored towards navigating through more restricted collections of pages, mainly aim at efficient exploration of pages relevant to a specific topic. To determine whether a page is relevant, several heuristics are used mostly based on the link context (i.e., the text appearing around the considered link), which is classified using machine learning techniques [1, 3, 6, 12, 13, 11, 15]. None of these specifically address pagination links, but rely on the general relevance metrics instead. Unfortunately, these approaches, though requiring significantly more training for the classifiers than our approach, usually reach accuracy below 70%, which is considerable for the general problem, but can be done for pagination links – as shown in this paper – at much higher accuracy.

While focused crawlers are driven by a given topic, *web data extraction* systems often automatise the extraction of data from specific web sites. In this respect, pagination links are fundamental. However, existing unsupervised approaches such as [18, 4, 7] mainly focus on the extraction of records on a page, exploiting repetitive structure, and do not address the task of automatically retrieving pagination links.

Web-block classification approaches typically start with a web-page's corresponding DOM-tree and identify sub-trees or sub-forests of this tree that correspond to particular block types (e.g. advertisements or navigation menus). Most of them employ machine learning techniques. However, to the best of our knowledge, none of these approaches addresses pagination links or even similar block types, but rather focus on larger (multi-node) block types. BER_yL is also able to classify such blocks, but in this paper we focus on recognising single-node block types such as pagination links.

Nevertheless, it is worth reviewing briefly how different feature types, content, visual, and structural, are used in these block classification systems. Most of the systems combine content and visual features. For example, [16] focuses on determining two block types (title and body) for one specific domain (news articles) and involves both content and visual features. In a similar way, [14] distinguishes between content and spatial features. In [10] the authors distinguish between stylistic features based on the DOM-tree and visual appearance of nodes and lexical features based on annotation texts of the DOM-nodes, quite similar to the content and visual features in our approach. The authors list site navigation elements among their classification labels, but the F_1 value they achieve for this block (82%) is significantly lower than our result for pagination links (99%) and the definition includes navigation menus and similar blocks. The same is true for [17] and [9]. They also consider navigation blocks, but with a wider definition

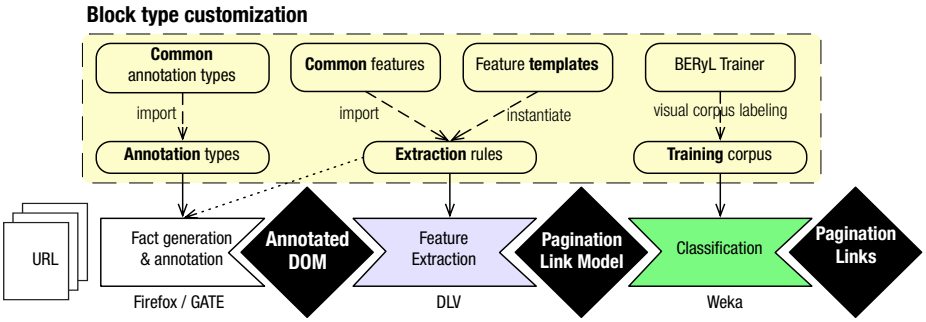


Fig. 2: BER$_y$L: Overview

and significantly lower accuracy (87% and 53%). They are also not able to distinguish pagination links leading to immediate next pages from others.

It may be surprising that we do not incorporate tags or tag paths of pagination links. In [19] the broomstick data-structure is proposed, which employs both the sub-tree rooted at a node (the broom head) and the tag path to it (the broom stick). However, in the case of pagination links, we have observed a high variation among the relevant tags and tag paths.

4 Block Classification with BER$_y$L

Before turning to the specific problem of identifying pagination links, we introduce BER$_y$L, a novel framework for block classification on web pages. BER$_y$L's novelty is the combination of extraction rules formulated in Datalog (with stratified negation and aggregation) and machine learning. Through this combination BER$_y$L allows a user **(1)** to introduce new block types with minimal effort, **(2)** to intermix highly selective, block type specific features alongside more common ones. When we configure BER$_y$L for a specific block type, we decide which of the predefined, *common features* should be considered and define new, block type specific features, mostly through instantiating *feature templates*. Features are specified over the facts on a page's DOM, content, or visual appearance. However, BER$_y$L automatically adjusts the extraction pipeline for a given block type to avoid generating, e.g., visual information if that is not needed by the specific extraction rules. This allows us to tailor BER$_y$L for block types that require fast extraction (possibly through sacrificing in precision or recall).

More specifically, Figure 2 shows an overview of block classification with BER$_y$L. The bottom illustrates the typical stages of classifying the blocks on an individual web page: **(1)** We generate facts about the page's DOM, content, or visual structure depending on the needs of the actual extraction rules. We extract content facts with a set of GATE [5] annotators for entity extraction. These can be customised to the specific block type. Other facts are extracted from a customised Firefox. **(2)** Based on the annotated DOM obtained in the preceding step, we extract a feature model through a set of feature extraction rules (Datalog rules with stratified negation and aggregation). The features for a specific block type are typically a mix of common features, instantiations of *feature templates*, and ad-hoc features. **(3)** Finally, given this feature model, we classify

Structural:	
`dom::node(N, T, P, Start, End)`	DOM node N has tag T, parent P, and start and end label `Start` and `End` (position of the start and end tag among the combined list of all start and end tags).
`dom::clickable(N)`	N is a clickable target (link or has `onClick` handler).
`dom::content(N, v, O, L)`	N has (textual) content v, starting at document offset O and having length L.
Visual:	
`css::box(N, Left, Top, Right, Bottom)`	bounding box of a DOM node N.
`css::page(Left, Top, Right, Bottom)`	dimensions of the page.
`css::resolution(Horizontal,Vertical)`	screen resolution.
`css::font_family(N, Family)`	N is rendered with a font from the given family.
Content:	
`gate::annotation(N, A, v)`	holds if $\mu(A,N,v)$ holds.

Table 2: BER$_y$L Annotated DOM

the blocks on the page. The classifier is learned from a training corpus, whose labelling is supported by the BER$_y$L-Trainer, a visual interface for labelling web page blocks.

Definition 1. *A* BER$_y$L ***feature model*** *is a set of features with values of boolean, integer, string, or tuple types. Each feature is defined through one or more* ***feature extraction rules****. An instance of a feature model is the result of evaluating those rules on a fact representation of a page. A* classifier *for a* BER$_y$L *feature model is a ML classifier learned from a set of a training instances for such a model.*

In the following, we briefly describe BER$_y$L, as needed for the identification of pagination links, and in Section 5, we show how to instantiate BER$_y$L for that purpose.

4.1 BER$_y$L Extraction Rules

Fundamentally, BER$_y$L's extraction rules are Datalog$^{\neg,\text{Agg}}$ (i.e., with stratified negation and aggregation) rules on top of an annotated DOM. To ease the definition of new features, we extend this with a template language that allows us to implement many features as simple template instantiations (without affecting the data complexity).

Definition 2. *An* ***annotated DOM*** *is a DOM tree P decorated with annotation types from an* annotation schema $\Lambda = (\mathscr{A}, \mu)$ *where $\mathscr{A}$ is a set of annotation types and $\mu : \mathscr{A} \times \mathscr{N} \times \mathscr{U}$ a relation on $\mathscr{A}$, the set of DOM nodes $\mathscr{N}$ and the union of all attribute domains $\mathscr{U}$. $\mu(A,n,v)$ holds if n is a text node containing a representation of a value v of attribute type A.*

For `<b>0xford,£2k</b>`, we obtain, e.g., $\mu(\text{NUMBER}, t, 2000)$, indicating that there is a number with value 2000 in t, assuming that t is the text node within the `<b>...</b>`.

An annotated DOM is represented in BER$_y$L through three sets of facts, structural, visual, and content (or annotation). Depending on the extraction rules, BER$_y$L automatically detects which of these sets are required and only generates the corresponding facts. In Table 2, we give a sample of the facts from each fact set, as needed for the

```
std::preceding(X, Y) ⇐ dom::node(X, _, _, _, End),
  dom::node(Y, _, _, Start, _), End < Start.
std::proximity(X, Y) ⇐ proximity_dimension(DHor, DVert),
  css::box(X, LeftX, TopX, _, _), css::box(Y, LeftY, TopY, _, _),
  TopY − DVert ≤ TopX ≤ TopY + DVert, LeftY − DHor ≤ LeftX ≤ LeftY + DHor.
std::left_proximity(X, Y) ⇐ std::proximity(X, Y),
  css::box(X, _, _, RightX, _), css::box(Y, LeftY, _, _, _),
  RightX ≤ LeftY.
std::first_screen(Left, Top, Right, Bottom) ⇐
  css::page(Left, Top, _, _), css::resolution(H, V),
  Right = Left + H, Bottom = Top + V.
```

Fig. 3: BER$_y$L standard predicates

following discussion. In BER$_y$L, we add namespaces (prefix followed by `::`) to separate facts from different fact sets.

Based on these facts, BER$_y$L provides a set of standard predicates and features that a specific block type may import and a set of feature templates for easy instantiation of new features. As for fact sets, BER$_y$L also uses namespaces for different rule sets. Standard predicates are in the `std` namespace. When instantiating a block type, we use a namespace to separate feature predicates from ordinary ones. Feature predicates are predicates that represent a feature to be used for classification. They come in two varieties, unary for boolean features, and binary for features with a value. Values may be integers, strings, or (flat) tuples of those.

Figure 3 shows some of the standard predicates in BER$_y$L: These range from structural relations between nodes (similar to XPath relations) over visual relations (such as proximity or if a node is to the north-west of another one) to information about the rendering context such as the dimensions of the first and last screen.

BER$_y$L provides a set of standard features such as a nodes's number of characters:

```
<Model>::char_num(X, Num) ⇐ node_of_interest(X), dom::content(X, _, _, Num).
```

`node_of_interest` is a predicate that specifies which nodes are to be considered for classification (e.g., all nodes, only links, only images). It is provided with the block specific extraction rules, see Section 5.

When we instantiate BER$_y$L for a specific block type, we import only those features that are actually relevant through an import statement such as

```
IMPORT <Model>::char_num INTO <plm>
```

This binds the template variable `<Model>` to `plm` and replaces all occurrences of that variable in the rule.

4.2 BER$_y$L Feature Templates

Different block types often have overlapping features, but it is even more common for them to differ only slightly, e.g., with respect to what DOM nodes to consider.

For that reason, we introduce a small template language atop of Datalog. This language allows us to specify a common pattern for features, factoring out constants or predicates in which these features may differ.

Figure 4 shows a sample of feature templates in BER$_y$L. The first template defines boolean features for any annotation type `AType` indicating whether a certain node of interest is annotated with `AType`. We instantiate it for a NUMBER in the `plm` pagination link model in the following way:

```
INSTANTIATE annotated_by<Model, AType> USING <plm, NUMBER>
```

In a similar way, the second template defines a boolean feature that holds for nodes of interest, if there is another node in their proximity for which `Property(Close)` is true. To instantiate it to nodes that are annotated with PAGINATION, we write

```
INSTANTIATE in_proximity<Model,Property(Close)>
USING <plm, plm::annotated_by<PAGINATION(Closest)>
```

Observe, that BER$_y$L templates thus allow for two forms of template parameters: variables and predicates. More formally,

Definition 3. *A* **BER$_y$L *template*** *is an expression* `TEMPLATE` $N\langle D_1,\ldots,D_k\rangle\{p \Leftarrow expr\}$ *such that N is the template name, $D_1,\ldots,D_k$ are template parameters, p is a template atom, $expr$ is a conjunction of template atoms and annotation queries. A template parameter is either a variable or an expression of the shape $p(V_1,\ldots,V_l)$ where p is a predicate variable and $V_1,\ldots,V_n$ are names of* required *first order variables in bindings of p.*

A template atom $p\langle C_1,\ldots,C_k\rangle(X_1,\ldots,X_n)$ *consists of a first-order predicate name or predicate variable p, template variables $C_1,\ldots,C_k$, and first-order variables $X_1,\ldots,X_n$. If $p(V_1,\ldots,V_l)$ is a parameter for N, then $\{V_1,\ldots V_l\} \subset \{X_1,\ldots,X_n\}$.*

An instantiation always has to provide bindings for all template parameters. We extend the usual safety and stratification definitions in the obvious way to a BER$_y$L template program. Then it is easy to see that the rules derived by instantiating a safe and stratified template program are always a safe, stratified Datalog$^{\neg,\text{Agg}}$ program.

5 Pagination Links with BER$_y$L

For identifying pagination links with high accuracy, we create a small feature model in BER$_y$L that consists of content, page position, visual proximity, and structural features. In Section 6, we show that not only BER$_y$L achieves almost perfect accuracy with this feature model for a wide range of domains and pages, but that these four feature types contribute notably to the overall performance.

Definition 4. *A* pagination sequence *is a sequence of web pages from the same domain, that is the result of paginating some information such as an article or a result set of a search. Given a DOM P of a page, the (immediate)* ***pagination link identification problem*** *is the problem of identifying those nodes in P that must be clicked to get to the following page in any pagination sequence the page is part of. The pagination links should be distinguished into numerical and non-numerical (such as "Next").*

```
TEMPLATE annotated_by<Model,AType> {
  <Model>::annotated_by<AType>(X) ⇐ node_of_interest(X),
    gate::annotation(X, <AType>, _). }
TEMPLATE in_proximity<Model,Property(Close)> {
  <Model>::in_proximity<Property>(X) ⇐ node_of_interest(X),
  std::proximity(Y,X), <Property(Close)>. }
TEMPLATE num_in_proximity<Model,Property(Close)> {
  <Model>::in_proximity<Property>(X,Num) ⇐ node_of_interest(X),
  std::proximity(Close,X), Num = #count(N: <Property(Close)>). }
TEMPLATE relative_position<Model,Within(Height,Width)> {
  <Model>::relative_position<Within>(X, (PosH, PosV)) ⇐ node_of_interest(X),
    css::box(X, LeftX, TopX, _, _), <Within(Height,Width)>,
    PosH = 100·LeftX / Width, PosV = 100·TopX / Height. }
TEMPLATE contained_in<Model,Container(Left,Top,Bottom,Right)> {
  <Model>::contained_in<Container>(X) ⇐ node_of_interest(X),
    css::box(X,LeftX,TopX,RightX,BottomX), <Container(Left,Top,Right,Bottom)>,
    Left < LeftX < RightX < Right, Top < TopX < BottomX < Bottom. }
TEMPLATE closest<Model,Relation(Closest,X),Property(Closest),Test(Closest)> {
  <Model>::closest<Relation>_with<Property>_is<Test>(X) ⇐ node_of_interest(X),
    <Relation(Closest,X)>, <Property(Closest)>, <Test(Closest)>,
    ¬(<Relation(Y,X)>, <Property(Y)>, <Relation(Y,Closest)>). }
```

Fig. 4: BER$_y$L feature templates

With BER$_y$L we reduce this problem to a block classification task over the set of clickable nodes (`DOM::clickable`). To do so, we need

1. to define appropriate annotation types if necessary,
2. to specify an appropriate feature model, as discussed in Section 4.1, and
3. to train a classifier on a small training set.

Annotation Types In addition to the standard annotation type NUMBER, we introduce two annotation types specific to pagination link identification: NEXT and PAGINATION. NEXT collects typical keywords used to indicate immediate non-numerical pagination links, e.g., “next”, “»”, or “›”, PAGINATION includes all those, but also keywords related to previous pagination links, e.g., “previous”, and to the number of results, e.g., “page”, “results”.

Feature Model Table 3 shows the features used in our approach to pagination link identification. They are split into four types: content, page position, visual proximity, and structural. The corresponding extraction rules are given in Figure 5.

For defining these features, we use a small number of auxiliary predicates as shown below. The first is required in any BER$_y$L feature model and specifies the domain of discourse, here all `dom::clickable` nodes (links and other click targets). The second is required in feature models that use proximity predicates and specify what we consider to be “in the proximity” of a node.

```
node_of_interest(X) ⇐ dom::clickable(X).
proximity_dimension(Width,10) ⇐ css::page(_, _, Width, _).
```

		Description	Type	Predicate
Content	1	Annotated as NEXT	bool	`plm::annotated_by<NEXT>`
	2	Annotated as PAGINATION	bool	`plm::annotated_by<PAGINATION>`
	3	Annotated as NUMBER	bool	`plm::annotated_by<NUMBER>`
	4	Number of characters	int	`plm::char_num`
Page position	5	Relative position on page	int^2	`plm::relative_position<css::page>`
	6	Relative position in first screen	int^2	`plm::relative_position<std::first_screen>`
	7	In first screen	bool	`plm::contained_in<std::first_screen>`
	8	In last screen	bool	`plm::contained_in<std::last_screen>`
Visual proximity	9	Pagination annotation close to node	bool	`plm::in_proximity<plm::annotated_by<PAGINATION>>`
	10	Number of close numeric nodes	int	`plm::num_in_proximity<numeric>`
	11	Closest numeric node is a link	bool	`plm::closest<std::left_proximity>_with <numeric>_is<non_link>`
	12	Closest numeric node has different style	bool	`<numeric>_is<different_style>`
	13	Closest link annotated with NEXT	bool	`<dom::clickable>_is<plm::annotated_by<NEXT>`
	14	Ascending w. closest numeric left, right	bool	`plm::ascending-numerics`
Structural	15	Preceding numeric node is a link	bool	`plm::closest<std::preceding>_with <numeric>_is<non_link>`
	16	Preceding numeric node has different style	bool	`<numeric>_is<different_style>`
	17	Preceding link annotated with NEXT	bool	`<dom::clickable>_is<plm::annotated_by<NEXT>`

Table 3: PLM: Pagination Link Model

```
numeric(X, Value) ⇐ gate::annotation(X, NUMBER, Value).
numeric(X) ⇐ numeric(X,_).
different_style(X,Y)⇐css::font_family(X,FX), css::font_family(Y,FY), FX ≠ FY.
```

The *content features* $1-4$ from Table 3 specify whether a node is annotated with one of the three annotation types (NEXT, PAGINATION, and NUMBER) and how many characters it contains. They are defined in Figure 5 by an instantiation and an import of the standard feature `char_num`. The instantiation creates three instances of the `annotated_by` template, one for each of the three annotation types.

The *page position features* are the relative position on the first page and on the first screen, as well as whether a node is on the first or last screen. They are defined by two instantiations in Figure 4, one for relative positions (using `css::page` and `std::first_screen`, resp.) and one for the presence in the first or last screen.

The *visual proximity features* are the most involved ones. They include a feature on whether there is a node in visual proximity that is annotated with PAGINATION (9), a feature on the number of `numeric` nodes in the proximity (10), and a feature that specifies whether the node and the closest numeric nodes in the proximity to the left and right form an ascending sequence (14). $11-13$ ask if the closest node with a certain property passes a given test, e.g., whether the closest numeric node is a link. Accordingly, $11-13$ are instantiations of `closest`, 9 and 10 are instantiations of `in_proximity` and

```
1 – 3 : INSTANTIATE annotated_by<Model, AType>
        USING <plm, { NEXT, PAGINATION, NUMBER>
  4 :   IMPORT <Model>::char_num INTO <plm>
5 – 6 : INSTANTIATE relative_position<Model,Within(Height,Weight)>
        USING <plm, {css::page(_,_,Height,Width), std::first_screen(_,_,Height,Width) }>
7 – 8 : INSTANTIATE contained_in<Model,Container(Left,Top,Bottom,Right)>
        USING <plm, {std::first_screen(Top,Left,Bottom,Right),...}>
  9 :   INSTANTIATE in_proximity<Model,Property(Close)>
        USING <plm, plm::annotated_by<PAGINATION(Closest)>
 10 :   INSTANTIATE num_in_proximity<Model,Property(Close)>
        USING <plm, numeric(Close)>
11 – 12 : INSTANTIATE closest<Model,Relation(Closest,X),Property(Closest),Test(Closest)>
        USING <plm, std::left_proximity(Closest,X), numeric(Closest),
                    { non-link(Closest), different_style(Closest,X) }>
 13 :   INSTANTIATE closest<Model,Relation(Closest,X),Property(Closest),Test(Closest)>
        USING <plm, std::left_proximity(Closest,X), dom::clickable(Closest),
                      plm::annotated_by<NEXT(Closest)>
 14 :   plm::ascending-numerics(X) ⇐ node_of_interest(X), numeric(X, ValueX),
          std::left-proximity(Left,X), std::right-proximity(Right,X),
          numeric(Left, ValueLeft), numeric(Right, ValueRight),
          ValueLeft < ValueX < ValueRight,
          ¬(std::left-proximity(Left,LeftN), std::left-proximity(LeftN,X), numeric(LeftN)),
          ¬(std::left-proximity(Left,RightN), std::right-proximity(RightN,X), numeric(RightN)).
```

Fig. 5: Extraction rules for pagination link identification

`num_in_proximity`, and 14 is the only feature in this model that is defined entirely from scratch.

The *structural features* are similar to $11-13$, but use XPath's preceding instead of visual proximity, E.g., 15 tests if the numeric node, immediately preceding the given node, is a link. Those are omitted from Figure 5 as they are similar to $11-13$.

5.1 Training the Classifier

With this feature model, BER_yL derives a classifier based on a small training set. For pagination link classification, a training corpus of only two dozen pages suffices to achieve the high accuracy demonstrated in Section 6. We expect a trade-off between accuracy, corpus size, and complexity of the feature model, but their relation in this triangle remains an open issue.

Figure 6 shows the classification tree derived on a training corpus for real-estate and car websites in the UK and detailed in Section 6. The tree employs almost all features, though the only structural feature considered is 15. For 5, we use both horizontal and vertical position, but at different points in the tree ($5.h$ and $5.v$). Visual proximity ($9--14$) and 15 are clearly very distinctive features of pagination links. Feature 1 has a key role in distinguishing numeric and non-numeric pagination links, but 11, 12, and 14 are also required to give an almost perfect distinction, as evident in Section 6.

Our classifier employs almost all of the features we have pre-selected. It places a strong emphasis on the visual features, such as relative vertical and horizontal positions

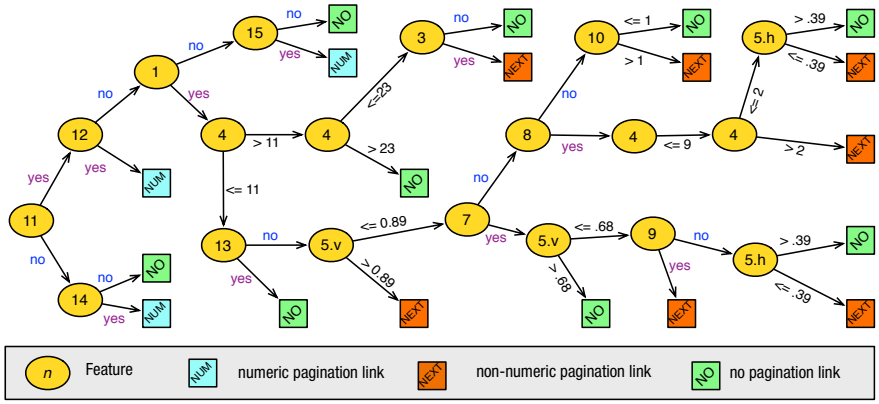


Fig. 6: Classification tree

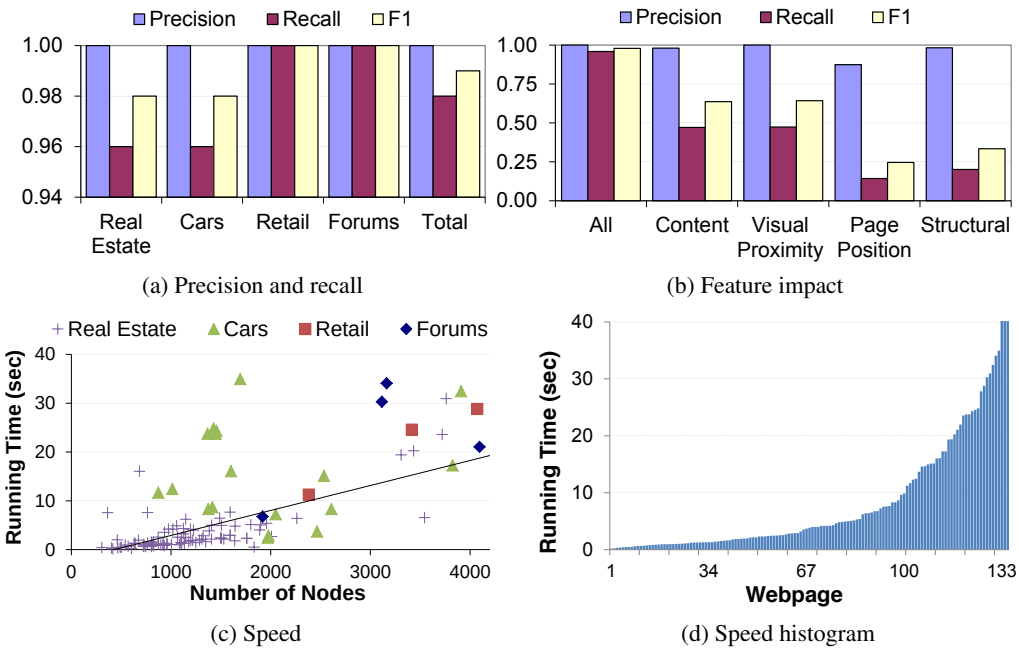
and whether the link to be classified is in the first or last screens. Manually finding rules that correspond to this classifier would be a very error-prone and time consuming task, in particular where thresholds and complex features are involved. This justifies the use of machine learning to obtain the precise classifiers. We further simplify this process by offering a visual tool, BER$_y$L-Trainer . With BER$_y$L-Trainer one can visually label elements on a page very quickly with the correct classification (only numeric and non-numeric are necessary, of course). It is configured with the annotation types and thus can support the user by offering the relevant types and performing a basic validation.

Given this specific classifier, BER$_y$L is able to quickly identify pagination links. To that end, we first compute the basic content, visual, and DOM facts, then apply the extraction rules and finally classify the feature model instance. Through this approach BER$_y$L achieves almost perfect accuracy, as shown in Section 6.

6 Evaluation

We evaluate our BER$_y$L pagination link classifier on a corpus of 145 websites from four domains (real-estate, used cars, on-line retail, and forums). For each domain, we selected the pages randomly from a listings page (such as `yell.com`) or a Google search. The latter favours popular sites, but that should not affect the results presented here. For example pages from the evaluation corpus, consider again Table 1 from Section 2.

In Figure 7, we show the results of evaluating quality, feature impact, speed, and per-page speed. Figure 7a illustrates that for all four domains our approach achieves 100% precision with recall never below 96%. This high accuracy means that our approach can be used for crawling or otherwise navigating paginated pages with a very low risk of missing information or retrieving unrelated pages. *Numeric pagination links* are generally harder to classify than non-numeric ones due to their greater variety and the larger set of candidates. Though precision is 100% for both cases, recall is on average slightly lower for numeric pagination links (98% vs. 99%) and in some domains quite notable (e.g., real estate with 96% vs. 99%). Figure 7b shows the overall precision, recall, and F_1 score compared with those for each of the basic feature sets. For space



(a) Precision and recall

(b) Feature impact

(c) Speed

(d) Speed histogram

Fig. 7: Evaluation results

reasons, we do not show the individual impact of all features, but note that all features included in the classification tree contribute at least several percentages to the overall precision and recall. Figure 7b also shows that content and visual proximity features are significantly more important for recall than page position and structural features.

Speed The speed of feature extraction is crucial for the scalability of our approach. As discussed above, the use of visual features by itself imposes a certain penalty, as a page needs to be rendered for those features to be computed. Figure 7c shows that the performance is highly correlated to page size with most reasonably sized pages being processed in well below 10 seconds (including page fetch and rendering). Our experiment was performed on a 3.4 Gz Intel Core i7 machine with 16 GB of RAM, running a 64-bit version of Linux Ubuntu. It is interesting to observe, that those domains where we used Google for generating the corpus and where the corpus is thus biased towards popular websites, seem to require more time than the real estate domain where the corpus is randomly picked from `yell.com`. Figure 7d shows the distribution of processing time over web pages, sorted by processing time. It is worth noting, that these results were achieved with an early prototype where the extraction rules had been implemented with DLV, a far more powerful reasoning language than what would suffice for our purposes.

7 Conclusion

Identifying pagination links with high accuracy is beneficial for many types of automated processing on the web. The approach taken in this paper shows that nearly perfect

accuracy is achievable through the use of our flexible block classification framework BER$_y$L. In contrast to previous approaches, BER$_y$L can be easily extended with block specific features. Not only is that essential for high accuracy but also allows us to keep the feature model small. There are two main open issues: Building an optimised version of the current system with sub-second classification time, which we believe is possible if the extraction rules are implemented, e.g., by a first-order rewriting. Another interesting problem would be the further automation of adding new blocks for classification, e.g., by automatic feature selection and gazetteer acquisition.

References

1. G. Almpanidis, C. Kotropoulos, and I. Pitas. Combining text and link analysis for focused crawling - an application for vertical search engines. *Inf. Syst.*, 32(6):886–908, 2007.
2. P. D. Bra and R. D. J. Post. Information retrieval in the world-wide web: Making client-based searching feasible. *Computer Networks and ISDN Systems*, 27(2):183–192, 1994.
3. S. Chakrabarti, M. V. D. Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. In *Computer Networks*, 1623–1640, 1999.
4. V. Crescenzi and G. Mecca. Automatic information extraction from large websites. *J. ACM*, 51(5):731–779, 2004.
5. H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, N. Aswani, I. Roberts, G. Gorrell, A. Funk, A. Roberts, D. Damljanovic, T. Heitz, M. A. Greenwood, H. Saggion, J. Petrak, Y. Li, and W. Peters. *Text Processing with GATE (Version 6)*. 2011.
6. M. Diligenti, F. M. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused crawling using context graphs. In *VLDB*, 527–534, 2000.
7. B. Fazzinga, S. Flesca, and A. Tagarelli. Schema-based web wrapping. *Knowledge and Inf. Sys.*, 26:127–173, 2011.
8. M. Hersovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur. The shark-search algorithm. an application: tailored web site mapping. *Computer Networks and ISDN Systems*, 30(1–7):317–326, 1998.
9. J. Kang and J. Choi. Block classification of a web page by using a combination of multiple classifiers. In *NCM*, 2008.
10. C. H. Lee, M. Y. Ken, and S. Lai. Stylistic and lexical co-training for web block classification. In *WIDM*, 2004.
11. H. Liu, J. Janssen, and E. Milios. Using HMM to learn user browsing patterns for focused web crawling. *DKE*, 59(2), 2006.
12. G. Pant and P. Srinivasan. Learning to crawl: Comparing classification schemes. *TOIS*, 23(4):430–462, 2005.
13. G. Pant and P. Srinivasan. Link contexts in classifier-guided topical crawlers. *TKDE*, 18(1):107–122, 2006.
14. R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma. Learning block importance model for web pages. In *WWW*, 2004.
15. P. Srinivasan, F. Menczer, and G. Pant. A general evaluation framework for topical crawlers. *Inf. Retrieval*, 8:417–447, 2005.
16. J. Wang, C. Chen, C. Wang, J. Pei, J. Bu, Z. Guan, and W. V. Zhang. Can we learn a template-independent wrapper for news article extraction from a single training site? In *KDD*, 2009.
17. X. Yang and Y. Shi. Learning web page block functions using roles of images. In *ICPCA*, 2008.
18. Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *WWW*, 2005.
19. S. Zheng, R. Song, J-R. Wen, and C. L. Giles. Efficient record-level wrapper induction. In *CIKM*, 2009.