Multiple-Signal Duplicate Detection for Search Evaluation

Scott Huffman, April Lehman, Alexei Stolboushkin
Howard Wong-Toi, Fan Yang, Hein Roehrig
Google Inc.
Mountain View, CA
{huffman,alehman,aps,hwongtoi,fanyang,hein}@google.com

ABSTRACT

We consider the problem of duplicate document detection for search evaluation. Given a query and a small number of web results for that query, we show how to detect duplicate web documents with precision ~ 0.91 and recall ~ 0.77. In contrast, Charikar's algorithm, designed for duplicate detection in an indexing pipeline, achieves precision ~ 0.91 but with a recall of ~ 0.58. Our improvement in recall while maintaining high precision comes from combining three ideas. First, because we are only concerned with duplicate detection among results for the same query, the number of pairwise comparisons is small. Therefore we can afford to compute multiple pairwise signals for each pair of documents. A model learned with standard machine-learning techniques improves recall to ~ 0.68 with precision ~ 0.90. Second, most duplicate detection has focused on text analysis of the HTML contents of a document. In some web pages the HTML is not a good indicator of the final contents of the page. We use extended fetching techniques to fill in frames and execute Javascript. Including signals based on our richer fetches further improves the recall to ~ 0.75 and the precision to ~ 0.91. Finally, we also explore using signals based on the query. Comparing contextual snippets based on the richer fetches improves the recall to ~ 0.77. We show that the overall accuracy of this final model approaches that of human judges.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: [Information Search and Retrieval]

General Terms

Algorithms, Measurement

Keywords

Compression distance, duplicate detection, machine learning, search evaluation, similarity hash, web search

SIGIR'07, July 23–27, 2007, Amsterdam, The Netherlands.

1. INTRODUCTION

Detection of duplicate or near-duplicate web pages is both an important problem for web search engines, and a very difficult problem. Several algorithms have been proposed and evaluated in recent years [11, 8, 3, 9, 2, 3]. Most approaches can be characterized as different types of distance or overlap measures operating on the HTML strings fetched for pairs of URLs being compared. State-of-the-art algorithms, such as Broder et al.'s [2] and Charikar's [3], achieve reasonable precision and/or recall, but do not approach human-level accuracy.

The problem we address in this paper is duplicate detection as a part of search evaluation. Highly accurate duplicate detection is critical for obtaining accurate relevance measurements. To see this, suppose we are interested in measuring the aggregate relevance of results served by various search engines, for a set of sample queries. We obtain a relevance score from a human judge for each of the top N results of each query, from each engine being measured. We can roll these scores up to an aggregate relevance metric for each engine, using a technique such as nDCG [10]. Duplicates can affect our measurement in two ways. First, an engine may receive undue credit for returning duplicate results for a given query. Unaccounted for, an engine that returns four copies of the (highly rated) appropriate homepage for a navigational query will receive credit four times over. In reality, users of the engine will be annoyed, not four times happier, and the useless duplicates that are returned will push other potentially good results down in the results list. Second, rating noise can be introduced when two duplicate results from different engines are not detected, and thus are judged separately and potentially receive different relevance scores. Similarly, rating noise is introduced if two non-duplicates are wrongly detected as duplicates, assuming this means only one of them will be judged and they will receive an identical relevance score. Therefore, accurate relevance measurement in this framework requires duplicate detection with both high precision and high recall.

Duplicate detection approaches used by modern search engines are designed for very high scalability. As part of building an index of web pages, an engine must efficiently identify duplicates among the billions of URLs being crawled and indexed. Because of this, recent approaches have focused on transforming the string returned by a URL into a single compact hash or "fingerprint." Comparing these fingerprints is very inexpensive and gives a measure of similarity/distance between the content of the corresponding web pages. Typically, these algorithms operate on content

that is directly fetched via the URL (what is fetched by `wget` or `curl`) — which in many cases is very different from what a user will see in a browser, due to constructs such as frames, Javascript, and other dynamic content. Fully "rendering" the page like a browser does, by fetching embedded frames, executing Javascript, and so on would simply be too costly.

For search evaluation, however, we can afford to explore more expensive approaches. We only need to find duplicates among the top search results for a given query, from some small number of search engines being measured (for instance, the top five results from three engines, giving up to 15 total URLs); and, for search evaluation, we only need to do this for hundreds or thousands of sample queries. Even accounting for measurement over time (where we might see a new URL tomorrow that is a duplicate of one we rated today), the numbers are not large.

We present an approach to duplicate detection for search evaluation that achieves precision and recall approaching human-level accuracy. The approach has three key elements:

- Use of multiple text-based signals, combined using a standard machine-learning classifier, as opposed to a single distance algorithm.
- Use of signals derived from both the directly-fetched content and the "rendered" content of each URL.
- Use of signals that are based on the *query* for which the URLs being compared were returned.

We evaluate performance using a collection of 34,577 URL pairs derived from major search engines' top results on 500 randomly-selected queries; each pair is labeled by multiple human judges as duplicate or non-duplicate. Because we want to maximize both precision and recall in a balanced way, we use a balanced F-score (2PR/(P+R)) as an overall accuracy measure, where P represents precision and R recall. Taking Charikar's [3] random-hyperplane hashing algorithm as a baseline, we find that using multiple signals combined via a decision-tree classifier improves the F-score by approximately 6 percent. Adding in signals derived from rendered content gives an additional 5 percent improvement, and adding query-based signals yields another percent. In aggregate, these methods reach an accuracy level that approaches the average accuracy of individual human judges.

Section 2 describes related work. Section 3 presents our duplicate detection method and describes several of the signals we use. Section 4 describes our evaluation methodology and the human-scored data we collected. Experimental results and a comparison of our method's accuracy to previous algorithms appear in Section 5. We provide conclusions and possible extensions in Section 6.

2. RELATED WORK

The majority of the work on duplicate document detection has focused on the problem of finding all duplicate document pairs without comparing all pairs of documents in the corpus. Along these lines, Manber [11] introduced the idea of considering all k-byte sequences in the text and fingerprinting the document with the sequences that hash to a bit sequence ending in c 0's where c is a parameter. This work was built on by Heintze [8] to develop a system for plagiarism detection. Broder et al. [2] applied a shingling approach similar to Manber [11] and Heintze [8] to the web. This approach was extended by Schleimer et al. [14] so that two documents that overlap for a long enough sequence are guaranteed to have at least one matching fingerprint stored.

Brin et al. [1] and Shivakumar and Garcia-Molina [15, 16] explored schemes for detecting plagiarism using sentence overlap and word frequency comparisons. These schemes compute multiple fingerprints per document and two documents are considered similar if a large fraction of their fingerprints agree. Shivakumar and Garcia-Molina looked at applying these schemes to the web where naively considering all pairs that have at least one fingerprint match is inefficient [17].

Charikar [3] developed a technique for fingerprinting documents that allows one to estimate the cosine similarity between documents quickly. The idea is to choose a set of random hyperplanes and then record for each document which half-space of each hyperplane contains the document term vector. We use an implementation of Charikar's algorithm as one of the signals for our system.

Recently, Henzinger [9] compared Broder et al.'s algorithm and Charikar's to determine the precision of each at a fixed recall. Henzinger adjusted the parameters of Charikar's algorithm to have the same storage needs as that of Broder et al.'s and achieve the same recall. The precision of each algorithm was then estimated. Henzinger found that both algorithms had higher precision when comparing documents from different sites. Since Charikar's algorithm identified a higher percentage of duplicate pairs from different sites, it also achieved higher precision than Broder's algorithm. Henzinger then used the algorithms in combination to produce a new algorithm that gets a better trade-off between precision and recall.

Calibrasi and Vitanyi [4] worked on clustering groups of similar documents using distance measures based on compression algorithms. Given a compression algorithm, the normalized compression distance between two documents A and B is defined as

$$ncd(A,B) = \frac{c(AB) - min\{c(A), c(B)\}}{max\{c(A), c(B)\}}$$

where AB is the concatenation of A and B and $c(X)$ is the compressed size of document X. Intuitively, if two documents are similar, the compression of their concatenation is similar in size to each of their individual compressed sizes, i.e., one document does not add much content beyond what is already in the other. For this study, one of our signals is based on the normalized compression distance.

Finally, in the area of query-specific duplicate detection, Gomes and Smith suggested using text around the query terms as a means for comparing documents [7]. This method provides another signal we use in our system.

3. APPROACH

Our approach to duplicate detection differs from the previous approaches in three respects. First, instead of relying on a single predictive signal we make use of multiple signals. The intuition is that each signal uses a different dimension to compare pages, and the combination of these signals is a better predictor than any signal alone. Second, we use richer fetching techniques to bring in a *rendered body* which in some cases is more representative of the page as rendered in a browser. Third, since our focus is on duplicate detec-

tion in the setting of search evaluation, we consider signals that estimate how similar the pages are with respect to a specific query.

3.1 Rendered Bodies

When considering whether two pages are duplicates, previous approaches have focused on the HTML that can be fetched with a `wget` command. In a significant percentage of cases, the contents of the HTML bodies (fetches) do not reflect what is rendered in a browser. In these situations, algorithmic duplicate determination based on the original fetches is usually not accurate.

We address two situations in which the fetched HTML is not representative of the rendered appearance of the page:

Redirects: In some cases the initial page returned with a simple fetch quickly redirects to a final "landing" page.

A tool like `wget` can follow HTTP redirects, but it does not handle "meta refreshes" and page loads initiated from scripts on the page. Sometimes these techniques are used to branch between different versions of a site. In this case, very similar initial HTML pages can lead to widely differing landing pages. Conversely, redirects are often used to drive traffic to a single page. In this situation the initial pages tend to be stuffed with keywords for search engines, which do not see the redirect, and the landing page is polished for human users.

Frames: A fetch of a web page using frames returns the frameset of the page, i.e., a description of the frame layout and URLs for the frame content. Two web pages from a site using a common frame layout will differ only in the frame URLs; hence, content analysis needs to include the content of those frame URLs. The same holds for pages using inline frames (iframes).

Our approach is to render pages with a Mozilla-based fetcher similar to Crowbar [6]. After the page is loaded in the fetcher, we execute the equivalent of "view source" in Firefox to generate the effective HTML of the URL. We do not wait for redirects with long delays. We incorporate frames by grafting their contents into the surrounding document at the place of the frame declaration; this produces invalid HTML, but is a good base for our content analysis.

Fetching and rendering is much more expensive than simply fetching the content of the initial URL: with external frames and scripts many more URLs need to be retrieved and rendering a page in a browser is CPU intensive. To reduce bandwidth requirements, we do not fetch images. Moreover, we observe that for many URLs, the effective HTML does not differ much from the original HTML. Therefore an interesting area of future work is to develop heuristics for whether expensive rendering of a page is worthwhile or not.

3.2 Predictive Signals

For a pair of URLs, we compute a number of different signals based on the URLs, the HTML bodies, the rendered bodies and the text of the query used to score the relevance of the URLs. We use only one URL-based signal, namely an exact match on the domain of the URL. In addition, we compute *body-based* signals from both the fetched HTML and the rendered bodies as well as *query-specific* signals based on the text appearing near the query terms in the page. We begin by describing the body-based signals and then discuss the query-specific signals.

3.2.1 Body-based Signals

For each pair of URLs, we fetch both an HTML body and a rendered body. (See Section 3.1.) For each of these bodies, we compute a number of distance metrics. Some of these metrics have been explored in the literature as standalone duplicate detection algorithms but our focus here is on using them as one of many signals in a larger duplicate detection system.

Exact title match: The simplest signal we use is a check of whether the titles of the two documents exactly match. Despite the simplicity of this signal, it turns out be very useful for finding pairs of documents that are incorrectly considered duplicates by more sophisticated algorithms.

Similarity-hash distance: We use Charikar's algorithm for hashing document term vectors to a 64-bit hash [3]. This hashing scheme ensures that the fraction of bits that agree between two documents' hashes is, in expectation, equal to the cosine similarity between the documents' term vectors.

Tf-idf distance: The tf-idf score of a term, namely the product of the frequency of the term in a document divided by its frequency in the corpus, is a commonly used concept in information retrieval. For duplicate detection, we compare the top 100 tf-idf terms in each document, using a logarithmic idf table. Let L_X be the list of terms from document X and $pos(w, L_X)$ be the position of term w in L_X. For two documents A and B, we compute a distance between L_A and L_B which is biased towards the movements of top terms in the lists. In particular, for documents A and B and a term w, we define their *weight* as:

$$\text{weight}_{A,B}(w) = \frac{1}{\ln(pos(w,A)+e-1)\ln(pos(w,B)+e-1)}$$

Without loss of generality, assume that $|L_A| \geq |L_B|$. Then the tf-idf distance between A and B is given by:

$$\text{tf-idf distance}(A,B) = 1 - \frac{\sum_{w \in L_A} \text{weight}_{A,B}(w)}{\sum_{w \in L_A} \text{weight}_{A,A}(w)}$$

Note that if $|L_B| > |L_A|$ then we normalize the tf-idf distance based on L_B instead.

Body-length distance: Let $len(X)$ be the length of the HTML body for document X. Then we define the *body-length distance* between documents A and B as

$$\text{bld}(A,B) = \begin{cases} 0 & \text{if } len(A) = len(B) = 0 \\ \frac{|len(A)-len(B)|}{max\{len(A),len(B)\}} & \text{otherwise} \end{cases}$$

Modified normalized compression distance (mcd): We apply the concept of compression distance to our duplicate detection problem using the `gzip` algorithm as the underlying compression algorithm. We modify the formula for compression distance to increase the accuracy for documents that are similar to each other. We define the *modified normalized compression distance* between documents A and B as

$$mcd(A,B) = \frac{max\{|c(AB)-c(AA)|, |c(AB)-c(BB)|\}}{max\{c(AA), c(BB)\}}.$$

Note that the modified normalized compression distance is 0 if and only if $c(AA) = c(AB) = c(BB)$. In contrast, the normalized compression distance may be *non-zero* for identical documents since $c(AA)$ is not guaranteed to equal $c(A)$. Our modification means that we can distinguish between document pairs that are identical and pairs that are very similar to each other at the cost of decreasing the accuracy of the measurement for very dissimilar pairs. This is acceptable because the difficult part of accurately detecting duplicate documents lies in distinguishing pairs that are duplicates from pairs that are merely very similar to each other.

3.2.2 Query-specific Signals

In search evaluation, the question of whether two URLs are duplicates is made in the context of the query for which these two URLs were returned as good search results. Two pages are duplicates if their content is essentially the same in light of the query. For each URL we extracted contextual *snippets* from the document that appeared near the query words. This text was similar to the text that users see on a search engine's results page as a description accompanying a search result. However we found that a few modifications allowed the snippets to be more accurate for duplicate detection. First, we used much longer snippets than would typically be presented to a user in a search result display. Second, search engines sometimes draw snippets from sources other than the visible text of the page, such as meta tags or ODP entries [12]. In our case, we restricted the snippets used as a duplicate detection signal to be from the visible text of the page.

4. METHODOLOGY

To generate training and test data, we first randomly sampled 500 English queries from Google's US querystream in mid 2006. Each query was run on three search engines, and the first five search results were recorded, yielding up to 15 unique URLs for each query. We then asked four human judges to indicate duplicate/non-duplicate for each pair of URLs associated with each query.

In reality, judging whether two URLs are duplicates is often very difficult; there are many "edge" cases. After some initial trials, so that we could get consistent data, we produced fairly detailed instructions to guide our human judges on some of the more common hard cases. Examples that we indicated should be considered duplicates included:

- Pages A and B contain the same content and appear to be from the same "site", but one is a "printer friendly" version of the other.
- Pages A and B are the same page, from different mirrors/archive sites.
- Pages A and B have essentially the same content and layout, with the exception of non-essential elements such as text advertisements, banner ads, or non-critical site "boilerplate" such as legal and copyright notices.
- Page A automatically redirects the browser to another page (even after a delay of a few seconds), and this page is a duplicate of page B.

Examples that we indicated should not be considered duplicates included:

- Pages A and B contain primarily the same content, but have different basic layouts and surroundings.
  - Example: Two pages that contain lyrics to the same song, but from completely different sites with different appearance, background and surrounding content.
  - Example: Two pages contain the same information about a hotel, restaurant, etc., but are different "aggregator" sites with different appearance, layout, etc.
- Pages A and B lead to the same content, but one page initially requires a user login or registration while the other does not.
- Pages A and B redirect to one another only under certain conditions. Example: `http://toolbar.yahoo.com` and `http://toolbar.yahoo.com/firefox` only redirect to one another if the user is using the Firefox browser.
- Pages A and B are essentially the same page, but contain different "sort orders" of the same list of items.

Using these instructions, four judges per URL pair had full agreement on 98.5% of pairs. The remaining 504 pairs were reviewed by additional judges (and in the hardest cases, by the authors), until we had a final judgment for every pair. Our data set included 537 duplicates out of 34,577 pairs. Note that the small number of duplicate pairs in the sample is partly due to the fact that we filtered out exact duplicate URLs returned from two different engines since these are trivial to label correctly. It should be noted that the rater agreement for the class of duplicate pairs was only 83.0% whereas for the non-duplicate pairs it was 98.8%. This demonstrates that the difficulty of detecting duplicate documents, even for human raters, lies in distinguishing the duplicate pairs from the pairs that are merely very similar to each other.

Treating the final judgments as ground truth, an interesting accuracy baseline is the precision and recall of individual judges. We took the first, second and third rater who rated each pair and compared their rating to the final rating. The best F-score obtained was 85.5% with a precision of 77.4% and a recall of 95.4%. Since these ratings were also used to determine the final duplicate/non-duplicate judgement, this is possibly a higher score than an independent rater could achieve. Interestingly, humans do better at recall but have a lower precision than either Charikar's algorithm [3] or our machine-learning approach.

5. EXPERIMENTAL RESULTS

Based on preliminary results we did not feel that a single duplicate detection algorithm could effectively distinguish duplicate documents with both high precision and high recall. Our goal was to understand how much each of the following three techniques improve duplicate detection:

1. use many signals computed from standard fetches and the URL instead of a single duplicate detection algorithm.
2. do extended fetching to generate rendered bodies which are sometimes more indicative of how the page will appear in a browser. Compute signals from the standard fetched bodies and the rendered bodies.

Group Key	Signals
F	same domain same title fetched-body mcd similarity hash tf-idf distance
R	same rendered title rendered-body mcd rendered similarity hash rendered tf-idf distance
FS	snippets from fetched bodies
RS	snippets from rendered bodies

Table 1: Signals used for classification

3. use signals based on the query.

To study this question, we used the publicly available data-mining toolbench `Weka` [18] to analyze our data. This tool enabled us to experiment quickly with a large variety of classification algorithms. In our initial tests, we were able to obtain much better results with classifiers that were tree-based or rule-based. This matched our intuition that the class of duplicates was not inherently linear, and that there were probably subclasses of duplicates definable by boolean conditions over various combinations of signals. To compare various feature sets, we used the average F-score from two classifiers: Jrip, a rule-based classifier (`Weka`'s implementation of the Ripper algorithm [5]) and J48, a decision-tree classifier (a `Weka` implementation of the C4.5 algorithm [13]). Each classifier was run using 10-fold cross-validation a total of nine times each using different randomization seeds for learning and cross-validation splits. The numbers reported are for the models with median F-score.

In order to understand the usefulness of our three ideas for improving duplicate detection we considered the following groups of signals in Table 1. Group **F** alone represents a set of signals that can be easily computed from a standard fetch. We use this set to evaluate how much improvement can be made over a single duplicate detection algorithm by combining multiple signals. Group **R** are the signals that can be computed from rendered bodies. We added group **R** to group **F** to evaluate the combined effect of using multiple signals and richer fetching techniques. When evaluating query-based signals we discovered that snippets derived from the fetched and rendered bodies behave differently. Therefore they are listed separately as groups **FS** and **RS** respectively. Note that body-length distance was not included in sets **F** or **R**. It turns out that body-length distance is not useful when modified normalized compression distance is included in the set of signals. However, in practice, body-length distance is a much faster signal to compute and therefore in Section 5.3 we discuss the effects of replacing modified normalized compression distance with body-length distance.

We first show how much improvement can be gained from each of these three techniques for our US English-based dataset. We then discuss using this model on different languages and queries from outside the US.

5.1 Results

We first consider the predictive strength of the individual signals, with precision and recall calculated over data for which the appropriate bodies were available (in some cases, we were unable to fetch the rendered bodies). The results appear in Table 2, ordered by decreasing F-score. The strongest two signals are drawn from the rendered bodies. This matches our intuition that these rendered bodies more closely match what the user sees, and when available often provide better predictors than signals based on the direct fetches. However, it should be noted that since only 90% of the pairs had rendered bodies for both URLs, their recall and F-scores over the full set of URL-pairs would be lower.

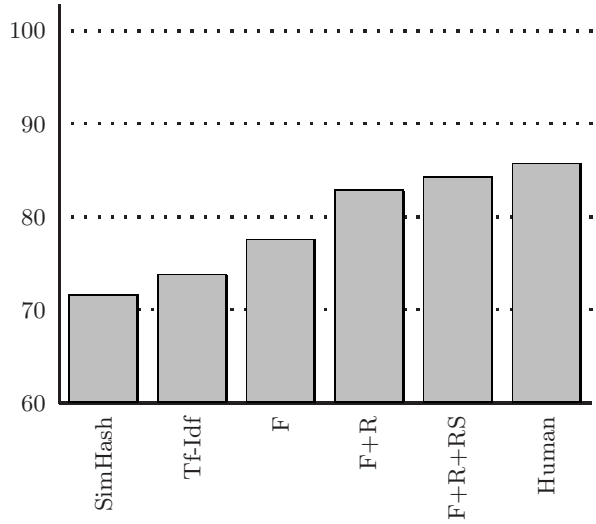

Figure 1: F-score of duplicate detection techniques: similarity hashing (simhash), Tf-Idf, F signals only, F + R signals, F + R + RS signals, and a human rater.

Figure 1 summarizes the improvement in duplicate detection accuracy as we cumulatively add groups of signals corresponding to our three techniques. We use the similarity hash distance as a baseline since it is commonly used in search-engine indexing systems. We also show the F-score for tf-idf distance since it is actually 2.2 points higher and represents the best F-score obtained by an individual signal computed from a standard fetch. Taking a combination of all fetched-body signals provides a 3.8 point improvement beyond tf-idf distance. Further adding signals computed from the rendered bodies adds another 5.3 points to the F-score. Finally, the addition of signals based on the rendered snippets (text surrounding the query terms in the rendered body) also appears to help ($\sim$1.4 points). Table 3 gives the breakdown of precision and recall for each of these signal sets. One interesting thing to note is that the improvement in F-score comes largely as a result of improving the recall while maintaining a high precision.

In Section 5.3, we repeat this analysis with the signals ordered by increasing computational cost.

The precision-recall curve for the model including all the signals in **F**, **R**, and **RS** is shown in Figure 2. While the overall F-score our algorithm achieves is close to that of a human rater, it does so with noticably higher precision but lower recall. For our algorithm to match human-level recall, its precision would be below 20%. We suspect this may be because our algorithms do not detect bolierplate content au-

Signals	Jrip			J48			Average
	P	R	F	P	R	F	F
rendered snippet mcd	85.6	70.3	77.2	90.3	66.6	76.9	77.0
rendered tf-idf distance	81.5	70.7	75.7	84.5	67.6	75.1	75.4
tf-idf distance	84.4	65.9	74.0	86.1	63.4	73.0	73.5
snippet mcd	88.9	61.4	72.6	91.9	59.3	72.1	72.3
similarity hash	90.2	59.9	72.0	91.4	57.6	70.7	71.3
body mcd	89.9	58.4	70.8	90.5	58.4	71.0	70.9
rendered body mcd	86.1	55.6	67.6	90.5	54.2	67.8	67.7
rendered body-length distance	84.3	55.6	67.0	89.6	52.7	66.4	66.7
body-length distance	86.6	52.1	65.0	90.0	52.2	66.1	65.5
same rendered title	52.2	80.9	63.4	52.2	80.9	63.4	63.4
rendered similarity hash	82.3	50.1	62.3	86.4	47.9	61.6	61.9
same title	50.9	70.7	59.2	50.3	63.6	56.2	57.7

Table 2: Individual signal strength. Rendered-body data were computed over the 31,231/34,587 (90%) pairs for which rendered bodies were available.

Signals	Jrip			J48			Average
	P	R	F	P	R	F	F
similarity hash	90.2	59.9	72.0	91.4	57.6	70.7	71.3
tf-idf distance	84.4	65.9	74.0	86.1	63.4	73.0	73.5
F	88.7	68.7	77.4	89.9	67.8	77.3	77.3
F + R	86.7	79.1	82.8	91.2	75.2	82.4	82.6
F + R + RS	87.8	81.4	84.4	91.3	77.3	83.6	84.0
Human Rater	P = 77.4			R = 95.4			85.5

Table 3: Precision and recall

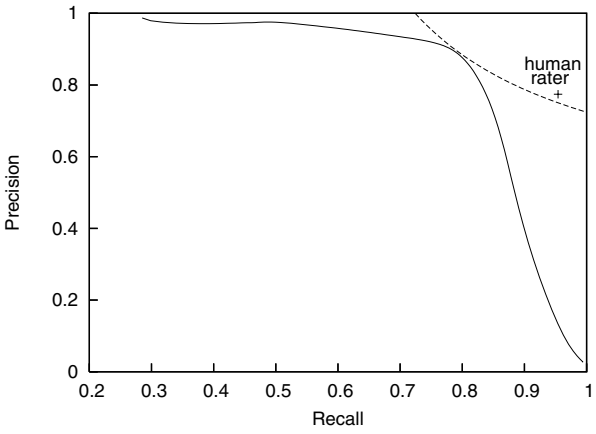

Figure 2: The solid line is the Precision-Recall curve for F + R + RS signals. The dotted line is the F-contour for the max achieved F-score of 84.0, just below that of the best human rater.

tomatically whereas humans can quickly eliminate irrelevant sections of a web page and determine that the underlying core content of two pages is the same.

Somewhat surprisingly, the addition of snippets generated from the fetched HTML of the documents does not improve the F-score. Results appear in the second column of Table 4, where each row shows the average F-score first with and then without signals based on fetched-body snippets.

Signals	F	F (when **FS** added)
similarity hash	71.3	—
F	77.3	77.3
F + R	82.6	82.6
F + R + RS	84.0	83.5

Table 4: Effect of fetched-body snippets

5.2 Cross-location performance

Our models were built from data based on queries that originated in the US and were made in English. A natural question is how well such models extend to queries for other locations and in other languages.

We tested these classifiers on two data sets from Google's United Kingdom and Chinese querystreams. We obtained our test sets in the same way we built our US training data, except that we sampled only 200 queries for each test set. For UK data we had 204 duplicates out of 14,328 pairs, and for Chinese data a significantly smaller 120 duplicates out of 14,377 pairs. Our results appear in Table 5. The average F-score of 83.8 for UK queries indicates that the models trained over US English queries fit well for UK English queries. However, these models do not translate as well to Chinese-query duplicate results, producing a maximum F-score of only 74.8. The small sample size in this dataset (only 120 duplicates) may be having an effect. In any event, more analysis is required to explain the discrepancy. Perhaps for double-byte languages, the results can be improved by explicitly training on double-byte language training sets.

Test set	Jrip			J48			Average
	P	R	F	P	R	F	F
UK	83.2	79.9	81.5	96.4	77.9	86.2	83.8
Chinese	64.8	56.7	60.4	75.4	74.2	74.8	67.6

Table 5: Test results for Chinese and UK queries

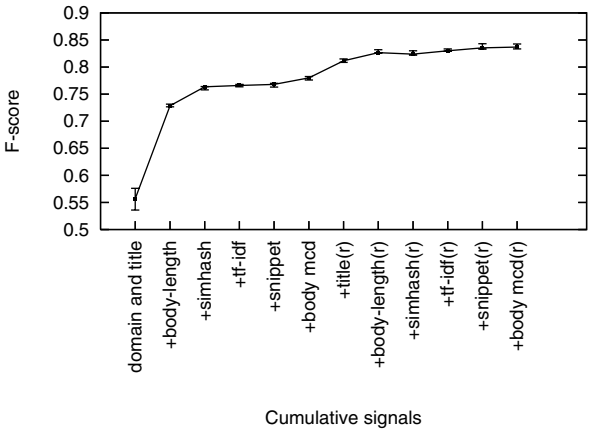

Figure 3: Performance gain while adding signals of increasing computational complexity. Signals computed from the rendered body are indicated by '(r)'. Errors bars denote the lower and upper quartile of F-scores per signal set.

5.3 Practical efficiency

The increased performance of our duplicate detection algorithm comes at a computational cost. In a real implementation, there are three classes of computationally intensive tasks, given below:

1. Obtaining the page content, which is made up of fetching and rendering. The rendering task is very CPU intensive and therefore has the dominant cost. Fetching tasks add latency because they can result in subsequent fetches to pull in secondary documents and the contents of frames. The cost of fetching and rendering scales linearly with the number of documents. In practice, tens of thousands of URLs per minute can be handled with a moderate number of servers.

2. Processing individual pages, which includes the cost of computing all signals that are based on a single page (tf-idf terms, snippets, similarity hashes, etc). This step is dominated by the cost of parsing each document since computing the signals is relatively cheap once the document is parsed. This step is considerably cheaper than step 1.

3. Duplicate determination. We first consider the simple task of determining whether two given URLs are duplicates. For this step we compute all the signals based on pair of pages (such as compression distance and tf-idf distance) and then apply our algorithms to these pair-based signals to determine whether the two pages are duplicates. The complexity of this process is dominated by compression distance computations. In practice, depending on the scale of the application, it may be wise to abandon computing compression distances on the entire bodies, but still use compression distances for the snippets. This change reduces the F-score to 83.5 which is not a large decrease from the 84.0 achieved when including compression distances over bodies. We now consider the cost of computing all duplicate pairs in a corpus of documents. The complexity of this step is proportional to the number of pairs of documents. In our case, this is proportional to the number of queries and the square of the number of results per query, which greatly reduces the number of pairs of pages that need to be compared.

Based on the above considerations, the signals can be ordered by increasing computational cost. When building a system, a natural approach is to choose only enough signals to reach a desired F-score. Figure 3 shows the marginal gain in F-score obtained by adding increasingly expensive signals to our model. We grouped same title and same domain together because individually they each provide very low F-score. Interestingly, combining just the three cheapest signals — same domain, same title and body-length distance — already gives an F-score above 70 which is as good as many of the more expensive signals alone. Adding similarity hash brings the F-score above 75. Computing the compression distance for fetched bodies does improve the F-score but a bigger increase is realized by adding signals based on the rendered content. This again matches our intuition that the rendered content is more indicative of what users see, and that it is beneficial to combine different kinds of signals (in this case, those from both fetched bodies and rendered bodies).

A practical embodiment of our methodology handling hundreds of queries and tens of thousands of URLs a minute is feasible. Figure 4 shows an example model when replacing the expensive modified normalized compression distances on bodies with body-length distance in the signal set $\mathbf{F} + \mathbf{R} + \mathbf{RS}$.

6. CONCLUSIONS

Our experimental results show that the accuracy of duplicate detection algorithms can be significantly improved by combining multiple text-based signals and computing those signals over both fetched and rendered bodies. We believe the quality of our models could be increased further by (1) detecting and removing boilerplate from document bodies, (2) more fine-grained feature selection, (3) using more sophisticated URL-matching signals, and (4) training over larger data sets.

The increased performance we gained comes at a computational cost. The resultant algorithm is only feasible over modest-sized problems. It would be interesting to explore the idea of combining multiple signals in other duplicate detection domains with stricter computational limits, such as web crawling and indexing.

```
(rendered_snippet_mcd <= 0.23888) and (same_rendered_title = true) and
         (bodylengthdistance <= 0.01325) => class=dup
(rendered_tfidfdistance <= 0.51996) and (rendered_snippet_mcd <= 0.16307) and
         (same_rendered_title = true) => class=dup
(tfidfdistance <= 0.45709) and (simhashdistance <= 4) and (same_title = true) => class=dup
(rendered_tfidfdistance <= 0.53986) and (rendered_bodylengthdistance <= 0.03841) and
         (rendered_simhashdistance <= 2) => class=dup
(rendered_tfidfdistance <= 0.51996) and (rendered_snippet_mcd <= 0.09619) and
         (rendered_simhashdistance <= 12) => class=dup
(tfidfdistance <= 0.53986) and (rendered_snippet_mcd <= 0.21829) => class=dup
(tfidfdistance <= 0.53986) and (bodylengthdistance <= 0.0015) and (same_domain = false) => class=dup
=> class=notdup
```

Figure 4: Example Jrip classification rules

7. REFERENCES

[1] S. Brin, J. Davis, and H. Garcia-Molina. Copy detection mechanisms for digital documents. In M. J. Carey and D. A. Schneider, editors, *SIGMOD Conference*, pages 398–409. ACM Press, 1995.

[2] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13):1157–1166, 1997.

[3] M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

[4] R. Cilibrasi and P. Vitanyi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005.

[5] W. W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Proc. of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, July 9–12, 1995. Morgan Kaufmann.

[6] Crowbar. `http://simile.mit.edu/wiki/Crowbar`.

[7] B. Gomes and B. Smith. Detecting query-specific duplicate documents, 2003. U.S. Patent 6,615,209.

[8] N. Heintze. Scalable document fingerprinting. In *USENIX Workshop on Electronic Commerce*, 1996.

[9] M. R. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In E. N. Efthimiadis, S. T. Dumais, D. Hawking, and K. Järvelin, editors, *SIGIR*, pages 284–291. ACM, 2006.

[10] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[11] U. Manber. Finding similar files in a large file system. In *USENIX Winter*, pages 1–10, 1994.

[12] Open directory project. `http://www.dmoz.org`.

[13] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[14] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: Local algorithms for document fingerprinting. In A. Y. Halevy, Z. G. Ives, and A. Doan, editors, *SIGMOD Conference*, pages 76–85. ACM, 2003.

[15] N. Shivakumar and H. Garcia-Molina. Scam: A copy detection mechanism for digital documents. In *Proceedings of the Second Annual Conference on the Theory and Practice of Digital Libraries*, 1995.

[16] N. Shivakumar and H. Garcia-Molina. Building a scalable and accurate copy detection mechanism. In *Proceedings of the First ACM Conference on Digital Libraries*, 1996.

[17] N. Shivakumar and H. Garcia-Molina. Finding near-replicas of documents on the web. In *WEBDB: International Workshop on the World Wide Web and Databases*, 1998.

[18] I. H. Witten and E. Frank. *Weka: Practical machine learning tools and techniques with Java implementations*. Morgan-Kaufmann, 1999.