# Adaptive Near-Duplicate Detection via Similarity Learning

Hannaneh Hajishirzi*
University of Illinois
201 N Goodwin Ave
Urbana, IL, USA
hajishir@uiuc.edu

Wen-tau Yih
Microsoft Research
One Microsoft Way
Redmond, WA, USA
scottyih@microsoft.com

Aleksander Kołcz
Microsoft
One Microsoft Way
Redmond, WA, USA
ark@microsoft.com

## ABSTRACT

In this paper, we present a novel near-duplicate document detection method that can easily be tuned for a particular domain. Our method represents each document as a real-valued sparse $k$-gram vector, where the weights are learned to optimize for a specified similarity function, such as the cosine similarity or the Jaccard coefficient. Near-duplicate documents can be reliably detected through this improved similarity measure. In addition, these vectors can be mapped to a small number of hash-values as document signatures through the locality sensitive hashing scheme for efficient similarity computation. We demonstrate our approach in two target domains: Web news articles and email messages. Our method is not only more accurate than the commonly used methods such as Shingles and I-Match, but also shows consistent improvement across the domains, which is a desired property lacked by existing methods.

## Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; I.2.6 [**Artificial Intelligence**]: Learning

## General Terms

Algorithms, Experimentation, Performance, Reliability

## Keywords

Near-duplicate Detection, Similarity Learning, Spam Detection

## 1. INTRODUCTION

Inspired by the needs of many real-world tasks when handling large document collections, near-duplicate detection (NDD) has been an important research problem for more than a decade [4, 6, 10, 20]. In many scenarios, two documents that are not exactly identical may still contain the same *content* and should be treated as duplicates. However, which portion of the documents should be considered important in such comparison depends on the final application and may vary from task to task. For example, Web pages from different mirrored sites may only differ in the header or footnote zones that denote the site URL and update time [7, 17]. News articles shown on different portals could come from the same source (e.g., Associated Press) and thus have identical content, but can be rendered in different site templates with advertisements [20]. In both cases, a search engine should not show these near-duplicate documents together since they carry identical information. In a plagiarism detection scenario, the definition of near-duplicate documents may be even looser. When a portion of one document, such as a sentence or a paragraph, is contained in another document, these two documents could be seen as near-duplicates. In contrast, perhaps the most extreme definition of a near-duplicate exists in an anti-adversarial scenario. Spam messages that belong to the same campaign may look very different because spammers often need to randomize the messages by obfuscating terms or adding unrelated paragraphs to pass the filter [13, 14, 16]. However, as long as the core payload text (e.g., a URL pointing to the spammer's site) is identical, two email messages are treated as near-duplicates.

There are two main considerations when solving an NDD problem: *efficiency* and *accuracy*. Applications of NDD typically need to handle a very large collection of documents. A practical algorithm will need to determine whether a document is a duplicate of some other documents in the repository in real-time [17]. As a result, *efficiency* has been the main focus of existing popular NDD approaches, where various techniques of generating short signatures using hash functions (e.g., [4, 2, 5]) and pruning inverted index searching (e.g., [20]) were invented. In contrast, the problem of how to improve NDD *accuracy* has received less attention. Most existing NDD methods encode documents as sets of token sequences (i.e., $k$-grams). In such binary vector representation, all the elements, no matter where the corresponding fragments come from (e.g., header, body or navigation block) of the document, are treated equally when comparing two documents. Although heuristic approaches have been developed for selecting important terms using IDF values (e.g., I-Match [6]) or special patterns based on stopwords (e.g., SpotSigs [20]), such approaches do not perform consistently across different domains. As a result, manually tuning the configuration or even selecting the *right* NDD method becomes a necessity to achieve acceptable accuracy for the target domain.

In this paper, we introduce a novel Adaptive Near-Duplicate Detection (ANDD) method to address the above issues. Given a small set of labeled documents that denote the near-duplicate clusters, ANDD learns a vector representation for documents in the target domain. Each element in the vector corresponds to a short token sequence (i.e., $k$-gram) associated with a real-valued weight that indicates its importance when determining whether two document are near-duplicate. Similarity scores, such as cosine or Jaccard, calcu-

*This work was done while the author was an intern at Microsoft Research.


*SIGIR'10,* July 19–23, 2010, Geneva, Switzerland.

lated based on this new vector representation provide higher near-duplicate detection accuracy. Such improvement does not come at the cost of sacrificing the computational efficiency, as established signature generation techniques can be easily adapted to reduce the dimensionality of the original vector representation.

Our contributions in this work are twofold. First, by observing that existing NDD approaches consist of two main steps, raw vector construction and fast similarity computation, we identify the former is critical to achieve better accuracy. Hashing schemes or truncated inverted index provide fast ways to *approximate* the similarity score calculated using the raw vectors. Theoretically, the prediction performance is upper bounded by the quality of the original representation. As we show experimentally, the accuracy difference between different hashing schemes is often limited and much less than the gain brought by a better document representation. Second, we extend a recently proposed similarity learning framework [21] to learn the vector construction. This allows ANDD to easily capture the implicit and often vague notion of "near-duplicate" through the editorial data for the target domain. As a result, comparable and often higher accuracy can be achieved compared to existing NDD methods, and such advantage is consistent across domains.

# 2. RELATED WORK

In this section, we briefly describe the representative NDD approaches with focus on their unique characteristics.

## 2.1 Shingling

As one of the earliest NDD methods, the *Shingling* [4, 2] algorithm views each document as a sequence of tokens and first encodes a document as a set of unique $k$-grams (i.e., contiguous subsequences of $k$ tokens). For the ease of further processing, each $k$-gram is encoded by a 64-bit Rabin fingerprint [19] and is called a *shingle*. The similarity between two documents is measured using the Jaccard coefficient between the shingle vectors. Documents with high similarity scores are considered near-duplicate.

One issue of applying the Jaccard similarity directly is the variable and potentially large size of the shingle sets, as they grow linearly in the number of document tokens. Broder *et al.* [4] introduced the technique of *min-wise independent permutations* to solve this problem by mapping each set of shingles to an $m$-dimensional vector, with $m$ typically much smaller than the original number of tokens in a document. In this process, $m$ different hash functions $h_1, \cdots, h_m$ are created and applied to all shingles. Let the set of shingles of the target document $d$ be $S(d) = \{s_1, s_2, \cdots, s_n\}$. The $j$-th element in the final vector is defined as the minimum hash value of $h_j$. Namely, $\min_{l \in \{1..n\}} h_j(s_l)$.

Each $m$-dimensional vector can be further mapped to a smaller set of *super shingles* by first separating the elements into $m'$ disjoint subsets of equal size and then fingerprinting each subset of elements using a different hashing function. This process effectively reduces the dimensionality of each vector from $m$ to $m'$, and thus saves the storage space and also speeds up the computation.

Notice that in the standard shingling methods, the construction of document signature vectors is purely *syntactic* – all the $k$-grams in the documents are treated equally. Alternatively, Hoad and Zobel [11] experimented with various strategies of selecting $k$-grams when encoding shingles, such as based on their TFIDF scores.

## 2.2 I-Match

Unlike Shingling, which creates a sequence of hash values based on a random sample of $k$-grams of the original document, *I-Match* maps each individual document into a single hash value using the SHA1 hash algorithm. Two documents are considered near-duplicate if and only if their hash values are identical [6]. The signature generation process of I-Match views a document as a single bag of words (i.e., terms, unigrams). In addition, only the "important" terms are retained in the bag. It first defines an I-Match lexicon $L$ based on collection statistics of terms using a large document corpus. A commonly used option is the inverse document frequency (*IDF*), where $L$ consists of only terms with mid-range *IDF* values. For each document $d$ that contains the set of unique terms $U$, the intersection $S = L \cap U$ is used as the set of terms representing $d$ for creating the signature.

One potential issue with I-Match occurs when the retained word bag $S$ is small (i.e., $|S| \ll |U|$). Because the documents are effectively represented using only a small number of terms, different documents could be mistakenly predicted as near-duplicates easily. To deal with such cases, a constraint is placed on the minimum length of a document for which a valid signature can be produced. To make I-Match more robust to such false-positive errors, Kolcz et al. propose using $m$ randomized lexicons concurrently instead of one, following the same I-Match signature generation process [15, 14]. As a result, an $m$-dimensional vector is used to present a document. Two documents are considered near-duplicate only if they have enough number of signatures matched.

## 2.3 Random projection

As pointed out by Charikar [5], the *min-wise independent permutations* method used in Shingling is in fact a particular case of a locality sensitive hashing (LSH) scheme introduced by Indyk and Motwani [12]. The probability that the two hash values match is the same as the Jaccard similarity of the two $k$-gram vectors. In contrast, the random projection based approach proposed by Charikar [5] and later applied to the web document domain by Henzinger [10] is another special LSH scheme for the cosine similarity based on term (i.e., unigram) vectors. According to the implementation described in [10], this algorithm generates a binary vector with $m$ bits to represent documents using the following steps. First, each unique term in the target document is projected into an $m$-dimensional real-valued random vector, where each element is randomly chosen from $[-1, 1]$. All the random vectors generated from the terms in this document are then added together. The final $m$-dimensional binary vector representing this document is derived by setting each element in the vector to 1 if the corresponding real value is positive and 0 otherwise.

Noticing that the Shingling method often generates more false-positive cases, Henzinger invented a hybrid approach that applies Charikar's random projection method to the potential near-duplicate pairs detected by Shingling [10]. As a result, the precision is significantly improved without sacrificing too much recall.

## 2.4 SpotSigs

In determining which $k$-grams in a document should be used for creating signatures, Theobald *et al.*'s SpotSigs method is perhaps the most creative and interesting one [20]. When developing near-duplicate detection methods for clustering news articles shown on various Web sites, they observe that stopwords seldom occur in the unimportant template blocks such as navigation sidebar or links shown at the bottom of the page. Based on this observation, they first scan the document to find stopwords in it as anchors. $k$ tokens right after an anchor excluding stopwords are grouped as a special $k$-gram, or so called a "spot signature" in their terminology. The raw representation of each target document is therefore a set of spot signatures. To some extent, the construction of spot signatures can be viewed as a simple and efficient heuristic to filter terms in template blocks so that the $k$-grams are extracted from the main

content block only. Once the spot signatures have been extracted, the same techniques of using hash functions as seen in other NDD methods can be directly applied to reduce the length of the spot signature vectors. In addition, Theobald *et al.* propose an efficient algorithm for directly computing the Jaccard similarity measures on the raw spot signature vectors, with the help of a pruned inverted index data structure [20].

# 3. ADAPTIVE NDD

Although their algorithm designs seem different, all the NDD methods surveyed in Sec. 2 can be described within a unified framework that consists of two main steps: (1) generating $k$-gram vectors from documents and (2) computing the similarity score efficiently based on the desired function operating on the vectors. Two documents are considered near-duplicate if their similarity score is above a predefined threshold. In order to have an efficient NDD method with improved prediction accuracy, our strategy is to follow the same pipeline but to ensure that the raw $k$-gram vector is a good representation of the document for computing reliable similarity scores. It will be a real-valued vector where the weights of the active $k$-grams are learned from labeled document pairs.

There are two main differences in our approach, compared to most existing NDD algorithms. First, each $k$-gram in the vector is associated with a real weight, which is used in computing document similarity scores. In contrast, existing methods typically decompose documents into bags of $k$-grams (i.e., binary vectors). Although some weighting schemes (e.g., removing terms based on *idf* values) may be used to retain only a subset of $k$-grams, all the active $k$-grams are treated equally during signature generation and/or similarity computation. As in many NDD applications, not all terms in the same document are equally important (e.g., some may occur in the title; others may appear in the navigation sidebar), treating terms indistinguishably important will obviously impact the accuracy of the NDD predictions [6].

The second difference of our approach is that we leverage labeled data in the target domain to learn these weights. Unlike existing NDD methods, which are more or less *static*, the learning approach provides a principled way to adjust the model to better fit the target domain. Specifically, we extend a recently proposed term-weighting learning framework [21] by using general $k$-gram vectors instead of just unigrams and by optimizing for different similarity functions such as the Jaccard coefficient. In what follows, we describe the learning approach in detail and demonstrate how the better document representation can be encoded to signatures that support efficient similarity computation.

## 3.1 Learning real-valued $k$-gram vectors

We first formally define the raw document representation in our approach, the real-valued $k$-gram vector. Let $\mathcal{V} = \{g_1, g_2, \cdots, g_n\}$ be the vocabulary that contains all possible $k$-grams occurring in all documents. Each document $d$ is mapped to a sparse vector $\mathbf{v}$, which consists of all the $k$-grams $\mathcal{G} \subseteq \mathcal{V}$ that can be found or selected in $d$. For each $k$-gram $g \in \mathcal{G}$, its score is decided by a function that depends on the $k$-gram $g$ and/or the document $d$: $gw_{\bar{\lambda}}(g, d)$, where $\bar{\lambda}$ is the model parameters learned from the labeled data.

Conceptually, this weighting function indicates how important the $k$-gram is with respect to the document, when computing the similarity between two vectors. While there are many choices of the functional form, we use a simple linear combination of features extracted for each $k$-gram $g_i$ occurring in document $d$:

$$gw_{\bar{\lambda}}(g_i, d) = \sum_j \lambda_j \cdot \phi_j(g_i, d), \quad (1)$$

| Feature | Note |
|---|---|
| Bias | 1 for all examples |
| TF | Frequency of the $k$-gram in the document |
| DF | Document frequency of the $k$-gram |
| DF-Avg | Average of document frequencies of terms in the $k$-gram |
| DF-Med | Median of document frequencies of terms in the $k$-gram |
| QF | Query log frequency of the $k$-gram |
| Loc | The location the $k$-gram in the document |
| Len | Length of the document |
| Cap | Whether the first term of the $k$-gram is capitalized |
| InTitle | Whether the $k$-gram appears in the title or email subject |
| InURL | Whether the $k$-gram is part of the URL of the document |

**Table 1: Features used in ANDD. DF-Avg and DF-Med are only used when $k > 1$. InURL is used only for HTML documents.**

where $\phi_j$ is the $j$-th feature function and $\lambda_j$ is the corresponding model parameter. The goal of the training procedure is thus to determine $\bar{\lambda}$ so that two near-duplicate documents can have a high similarity score.

The learning framework enables a principled way of incorporating additional information regarding the raw text in *features*. For each $k$-gram, the features used are listed in Table 1. Note that these features are all very easy to compute. Some are statistics derived from scanning the whole document once (e.g., Loc and Len). Others can be retrieved from a direct table look-up (e.g., DF and QF).

Given two documents $d_p$ and $d_q$, their similarity score is given by a specified similarity function $f_{sim}$ operating on their corresponding $k$-gram vectors $\mathbf{v}_p$ and $\mathbf{v}_q$. We develop models for two commonly used similarity functions in the task of NDD, *cosine* and the (extended) *Jaccard coefficient* in this work.

$$cos(\mathbf{v}_p, \mathbf{v}_q) = \frac{\mathbf{v}_p \cdot \mathbf{v}_q}{\|\mathbf{v}_p\|\|\mathbf{v}_q\|} \quad (2)$$

$$jacc(\mathbf{v}_p, \mathbf{v}_q) = \frac{\mathbf{v}_p \cdot \mathbf{v}_q}{\|\mathbf{v}_p\|^2 + \|\mathbf{v}_q\|^2 - \mathbf{v}_p \cdot \mathbf{v}_q} \quad (3)$$

Having human subjects score the *importance* of each $k$-gram that leads to a robust similarity measure is a difficult annotation task. Instead, we assume that we are given clusters of documents as the labeled data. Documents belonging to the same clusters are near-duplicate and unrelated otherwise. How to use such labeled documents to train the model parameters depends on the learning setting. We choose the setting of learning the *preference ordering* because of its slightly superior performance reported previously [21]. In this setting, the absolute similarity score of two documents is not important. Instead, we would like the model to assign higher scores to near-duplicate documents, compared to other unrelated pairs of documents. Due to the lack of space, we omit the derivations of gradients for both cosine and Jaccard functions here.

A training set of $N$ examples in this setting is formally denoted as $\{(y_1, (x_{a_1}, x_{b_1})), (y_2, (x_{a_2}, x_{b_2})), \cdots, (y_N, (x_{a_N}, x_{b_N}))\}$, where $x_{a_k} = (d_{p_{a_k}}, d_{q_{a_k}})$ and $x_{b_k} = (d_{p_{b_k}}, d_{q_{b_k}})$ are two pairs of documents and $y_k \in \{0, 1\}$ indicates the pairwise order preference, where 1 means $x_{a_k}$ should be ranked higher than $x_{b_k}$ and 0 otherwise. The loss function we used for this setting is:

$$L(\bar{\lambda}) = \sum_{k=1}^{N} \log(1 + \exp(-y_k \cdot \Delta_k - (1 - y_k) \cdot (-\Delta_k))), \quad (4)$$

where $\Delta_k$ is the difference of the similarity scores of two document pairs, computed based on the corresponding vectors. Namely,

$$\Delta_k = f_{sim}(\mathbf{v}_{p_{a_k}}, \mathbf{v}_{q_{a_k}}) - f_{sim}(\mathbf{v}_{p_{b_k}}, \mathbf{v}_{q_{b_k}})$$

The loss function (Eq. 4) is regularized by adding a term $\frac{\alpha}{2}||\mathbf{w}||^2$.

We use L-BFGS to minimize the loss function for its guarantee to find a local minimum. We omit the derivations of gradients for both cosine and Jaccard functions due to lack of space.

## 3.2 Generating document signatures

In many applications, the NDD method needs to handle a large collection of documents. Therefore, being able to *efficiently* determine whether the similarity score of two documents is high enough to declare they are near-duplicate is not only crucial but also a requirement to a practical NDD algorithm. Based on the traditional vector space model and cosine similarity, one simple way to detect near duplicates is to sequentially submit each document in the collection as a query to search for highly similar documents in the collection. By the use of inverted index as adopted in most search engines, this kind of document search is sub-linear (to the number of documents in the collection) in time complexity in practice, however it can be $O(n^2)$ in the worst case.

Recall that the weights in each vector can be interpreted as the importance scores of the corresponding $k$-grams, the straightforward way to shrink the vector size is thus by eliminating $k$-grams with low weights. Efficient similarity computation can be supported by techniques like pruned inverted index [20]. While this could be an effective strategy as shown in our experiment (see Sec. 4.2.2), having variable sizes of document representation is less appealing in terms of engineering. Meanwhile, although low-weight $k$-grams are not as important, they still contain information that could affect the similarity measure. In contrast, signature generation techniques map vectors into strings of a fixed number of bits. Even with the same complexity order, the basic bit comparison operation is much faster than comparing whether two strings (i.e., terms) or two integers (i.e., term id) are identical. Moreover, efficient sub-linear algorithms exist for similarity search in the signature space [5, 12, 1].

In this paper, we rely on locality sensitive hashing (LSH) schemes to map the raw vector to a sequence of hash values as the document signature. An LSH scheme has the following defining property:

DEFINITION 3.1 ([5, 12]). *Let $f_{sim}(\cdot,\cdot)$ be a given similarity function defined on the collection of objects $\mathcal{O}$. A distribution on a family $\mathcal{H}$ of hash functions operating on $\mathcal{O}$ is a locality sensitive hashing scheme if for $x, y \in \mathcal{O}$,*

$$\mathbf{Prob}_{h\in\mathcal{H}}[h(x) = h(y)] = f_{sim}(x, y)$$

Using this scheme, hash functions $h_1, h_2, \cdots, h_m$ drawn from $\mathcal{H}$ are applied to raw vectors to encode them into signatures of $m$ hash values. The similarity score of two documents is derived by counting the number of identical hash values, divided by $m$. As $m$ increases, this scheme will approximate asymptotically the true similarity score given by the specific function $f_{sim}$. Since the similarity functions that our learning method optimizes for are cosine and Jaccard, we apply the corresponding LSH schemes when generating signatures.

For the cosine function, we use the random hyperplane based hash function, which is the essential part of Charikar's random projection algorithm [5]. For a given collection of vectors in $\mathbf{R}^d$, each hash function is created by first choosing a random vector $\bar{r}$ from the $d$-dimensional Gaussian distribution. When applied to a vector $\bar{u} \in \mathbf{R}^d$, the corresponding binary hash function $h_{\bar{r}}$ returns 1 if the dot product $\bar{r} \cdot \bar{u} \geq 0$; otherwise it's 0. For vectors $\bar{u}$ and $\bar{v}$, this LSH scheme has the following property:

$$\mathbf{Prob}[h_{\bar{u}} = h_{\bar{v}}] = 1 - cos^{-1}(\bar{u}, \bar{v})/\pi,$$

which has a monotonic mapping of the cosine function.

When applying this scheme to our $k$-gram vectors, each $k$-gram in the vocabulary is associated with $m$ different random numbers drawn from the Gaussian distribution. The signature of each vector/document is a bit-string of $m$ bits. The value of the $i$-th bit is decided by the sign of summing the product of the $i$-th random number and the weight of each $k$-gram. Notice that this scheme works for both binary and real vectors, and the number of bits (i.e., $m$) does not need to increase when handling real vectors.

For the Jaccard function, the LSH scheme that we use is the min-hash [12, 8] function, which are designed originally for binary vectors. To handle our real $k$-gram vectors, we first transfer each real-valued weight to a binary vector as suggested by Gionis *et al.* [8]. The weight of each $k$-gram in the vector is multiplied by a big integer and then the number is bucketed and mapped to a bit-string. The original real vector thus becomes a binary vector by concatenating these bit-strings.

It is impractical to choose a hash function $h$ uniformly among all the possible functions. Therefore, we limit the search among a specific class of functions (linear in our experiments) as suggested by [3, 20]. Each hash function $h_i$ in this family is defined by two random numbers $\alpha_i$ and $\beta_i$ that are smaller than the length of the mapped binary vectors. Let $X$ be the set of indices of the "1" bits in vector $\bar{u}$. The $i$-th hash value of this vector is defined as $h_i(\bar{u}) = \min_{x\in X}(\alpha_i \cdot x + \beta_i \bmod p)$ where $p$ is the first prime number bigger than the length the mapped binary vector. Similarly, a complete document signature consists of $m$ such min-hash values, and the Jaccard coefficient is approximated by the fraction of identical hash values in the corresponding signature vectors.

# 4. EXPERIMENTS

In this section, we compare our near-duplicate detection methods with other state-of-the-art approaches on two important tasks: *De-duping Web news articles* and *email campaign detection*. Although in both tasks, the goal to identify semantically identical documents is the same, the syntactic presentation of near-duplicates in fact differs significantly from one domain to the other. As we demonstrate, our approach can adapt to the target domain easily while existing methods often suffer from such domain change and their predictions cannot be consistently reliable. Other practical issues when applying our method, such as the number of training examples it needs and whether it increases the computational cost during runtime, will be discussed at the end of this section.

## 4.1 De-duping Web news articles

In this set of experiments, we focus on determining whether news articles shown on different sites are identical and thus are potentially from the same source. As observed by Theobald *et al.* [20], Web sites showing these articles have their own unique style or template. These news pages may differ substantially in their presentation and only the core content can decide whether they are near-duplicate articles. As a result, correctly distinguishing the important portion of the text and encoding such information in the document representation become a key to improve the accuracy for this task. Below, we start from describing the data used for this set of experiments and the detailed experimental settings. We then compare our method with others in detail.

### *4.1.1 Experimental setting*

The dataset we used in this task is the *gold set* of near-duplicate news articles collected by Theobald *et al.* [20], which contains 2,160 news articles crawled in 2006. These articles are manually clustered into 68 directories, where documents in the same directory have the same content news and are considered near-duplicate.

| | Unigram ($k = 1$) | | Trigram ($k = 3$) | |
|---|---|---|---|---|
| | Cosine | Jaccard | Cosine | Jaccard |
| ANDD-Raw | **0.956** | **0.952** | **0.936** | **0.910** |
| TFIDF | 0.884 | 0.874 | 0.875 | 0.873 |
| Binary | 0.861 | 0.852 | 0.869 | 0.867 |
| SpotSigs | 0.953 | 0.952 | - | - |

**Table 2: MAX $F_1$ of ANDD-Raw $k$-gram vectors with $k \in \{1, 3\}$ using the cosine and Jaccard similarity functions, compared to other $k$-gram weighting functions (binary and TFIDF) in the News domain.**

We use this dataset for a five-run experimental setting. In each run of the experiment, we divide these clusters randomly into two disjoint sets of equal size; one set of clusters of documents is used for training and the other is used for testing. In other words, no cluster has documents in both the training and testing sets.

We construct our training instances as pairs of documents drawn from the training set. Each instance is associated with a binary label. If two documents are from the same cluster and thus near-duplicates, then the label is positive; otherwise, it's negative. Notice that the number of negative instances in the training set is much more than the number of positive instances. We balance the class distribution by randomly selecting the same number of negative instances as positive instances from the current set. 80,000 random pairs of these instances are used to train the model.

Likewise, testing instances are all the pairs of documents created from the testing set. To mimic the true distribution, we do not balance the number of positive and negative instances as done previously and evaluate various methods using all the testing instances. In particular, we evaluate the accuracy by the number of these instances with their labels correctly predicted. Following [20], we report the averaged Max $F_1$ scores of our experimental results as the major evaluation metric.

### 4.1.2 Raw similarity measure

We first show the quality of the similarity measures computed on the $k$-gram vectors that include all the $k$-grams, when used for detecting near-duplicate documents. As discussed previously, existing NDD methods essentially approximate the raw similarity measure efficiently by either using the hashing trick or inverted index. In other words, the quality of the raw similarity score dictates the final NDD prediction accuracy.

We compare several configurations of the raw similarity measures based on three variables: $k \in \{1, 3\}$ (i.e., unigram or trigram), similarity function $f_{sim} \in \{cosine, jaccard\}$ and the $k$-gram weighting function, which can be binary, TFIDF or learned (denoted as ANDD-Raw). Table 2 presents the averaged Max $F_1$ scores by varying the decision threshold when using the similarity measures of all the configurations. As can be observed from the table, regardless of the choice of $k$ and the similarity function, ANDD-Raw yields a better similarity measure for near-duplicate detection. Their Max $F_1$ scores are statistically significantly better than their TFIDF counterparts. Not surprisingly, without pruning $k$-grams using heuristics such as *idf* values, the raw similarity measure based on binary vector representation leads to the lowest Max $F_1$ scores in our experiments. Again, the differences are statistically significant[1]. We also found that on this dataset, the cosine function performs slightly better than the Jaccard coefficient, although the differences are not statistically significant. This phenomenon is shown on both unigram ($k = 1$) and trigram ($k = 3$) representations, where using unigram derives better NDD results. It is also worth noticing that with a small set of simple features, our approach can perform comparable to SpotSigs, which is based on a heuristic designed specifically for this problem domain.

[1]We run a student's paired-t test on individual scores from the five rounds of the compared two configurations. The results are considered statistically significant when the p-value is lower than 0.05.

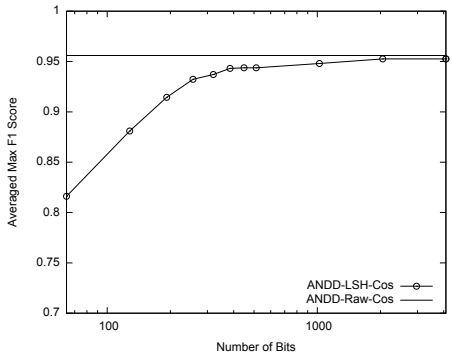


**Figure 1: Averaged Max $F_1$ scores of ANDD-LSH-Cosine for the News domain when encoded using different numbers of bits.**

### 4.1.3 Signature generation

While the raw similarity measures derived from ANDD-Raw $k$-gram vectors can achieve highly accurate results, reducing the size of such document representation is essential to efficient computation. In this section, we first apply locality sensitive hashing schemes described earlier to map unigram vectors to signatures with different lengths and examine the degree of performance degradation in terms of NDD accuracy. We then compare our approach with other signature-based NDD algorithms.

Taking the unigram ANDD-Raw vectors where the weights are learned to optimize for the cosine function, we map each document to a signature of $m$ bits via the LSH-Cosine scheme. The similarity score of two documents is then determined by the number of identical bits their corresponding signatures share, divided by $m$. We vary $m$ from 64 to 4096 and report the averaged Max $F_1$ scores of the 5 rounds in Figure 1. As can be observed from the figure, the accuracy of the NDD prediction based on the bit-string signatures improves quickly as the number of bits (i.e., $m$) increases. The performance degradation compared to the raw vector representation is limited when $m$ is not too small. In fact, when $m \geq 512$, the difference in Max $F_1$ when compared to either the raw vector or the SpotSigs method is not statistically significant. In practice, the choice of $m$ is determined by the trade-off between the efficiency constraint and the desired prediction accuracy, as longer signatures take more storage space and processing time.

We conduct similar experiments applying the LSH-Jaccard scheme to the unigram ANDD-Raw vectors where the weights are learned to optimize for the Jaccard function. Each vector is mapped to a set of $m$ min-hash values. Because each min-hash value takes roughly 20 bits to store, to get roughly the same size of signatures, we vary $m$ from 8 to 256 and report the averaged Max $F_1$ scores of the 5 rounds in Figure 2. Compared with the LSH-Cosine scheme, LSH-Jaccard is less stable and also needs a larger size of signature to approximate the raw similarity measure. In this set of experiments, we found that the signature needs to have at least 80 min-hash values to achieve a Max $F_1$ score that is not statistically significantly different from the one derived from the original Jaccard score.

We next compare our approach with other signature-based NDD methods, including Shingles, Charikar's random projection algorithm and I-Match. Following the setting chosen by Henzinger [10],

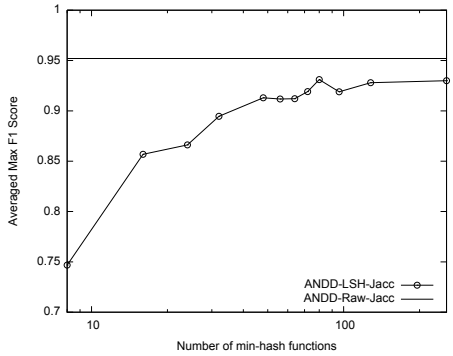


**Figure 2: The averaged Max $F_1$ scores of ANDD-LSH-Jaccard for the News domain when encoded using different numbers of min-hash functions.**

| Method | IDF | Bytes | Prec | Rec | $F_1$ |
|---|---|---|---|---|---|
| ANDD-LSH-Cos | [0,1] | 48 | 0.965 | 0.923 | 0.943 |
| ANDD-LSH-Jacc | [0,1] | 200 | 0.957 | 0.907 | 0.931 |
| Shingles | [0.2,0.85] | 672 | 0.886 | 0.867 | 0.876 |
| Charikar | [0.2,0.85] | 48 | 0.868 | 0.777 | 0.820 |
| 1-Shingles | [0.2,0.85] | 48 | 0.994 | 0.013 | 0.026 |
| I-Match | [0.4,0.75] | 24 | 0.983 | 0.001 | 0.002 |

**Table 3: Results of signature-based NDD methods for the News domain. ANDD-LSH-Cos achieves the best Max $F_1$ despite its short signature length.**

we use 384 bits (48 bytes) to encode the signatures in ANDD-LSH-Cos and Charikar's random projection method, and represent each document using 84 shingles (672 bytes) and 6 super shingles (48 bytes) for the Shingles and 1-Shingles methods. Because the unigram model performs better than trigram model, we set the window size in Shingles to 1. For ANDD-LSH-Jacc, each signature consists of 80 min-hash values (200 bytes) since it gives the best trade-off between accuracy and storage based on Figure 2. I-Match uses only one hash-value for each document and the length is 24 bytes.

Table 3 summarizes the performance of these algorithms in terms of the averaged Max $F_1$ scores of the five rounds, as well as the corresponding precision and recall numbers. Among them, ANDD-LSH-Cos achieves the best Max $F_1$ score despite the fact that its signature length is relatively short. The Jaccard-based counterpart achieves pretty competitive result, but with longer signatures. A regular Shingles method with 84 hash values performs better than Charikar's random projection approach with 384 bits on this domain. Both have Max $F_1$ scores higher than 0.8. Surprisingly, despite having impeccable precision, both 1-Shingles (at least 1 of the 6 super shingles matches) and I-Match achieve extremely low recall. Consequently, the $F_1$ scores are also low. This might be due to the fact that we judge the accuracy using all document pairs, rather than whether each document is put in the same cluster.

Notice that the additional features other than TF and DF significantly boost the performance. Even though the Max $F_1$ scores of ANDD-LSH are lower than ANDD-Raw, they are still much higher than those derived from raw TFIDF vectors (see Table 2).

## 4.2 Detecting email campaigns

Another application of near-duplicate document detection is capturing email campaigns, which has been shown an effective technique for collaborative spam filtering and identifying abused email accounts [9, 18, 15, 13]. An email campaign is a large volume of semantically identical mail sent to a large number of recipients within a short period of time. Although some of these campaigns may be legitimate marketing newsletters, a big portion of them are in fact spam. To a large email service provider, such as Hotmail or GMail, as long as there are a few users reporting junkmail messages, all other messages from the same campaign can be quickly classified as spam and removed from the inboxes of their recipients.

Unfortunately, reliably detecting spam campaigns is not trivial. In order to escape from such spam filtering scheme, spammers typically randomize their messages by obfuscating words or phrases in the email subject or body, or by appending random, unrelated paragraphs at the end of the mail. Figure 3 shows an example of two messages from the same campaign. As we can see here, the notion of "near-duplicate" in this domain becomes quite different from the Web domain – two dissimilar messages may in fact come from the same campaign because they convey the same payload content.

### *4.2.1 Experimental setting*

To judge the effectiveness of different NDD algorithms when applied to a practical problem such as spam campaign detection, it is best to evaluate them in a realistic setting. We experiment with our approach using a set of 11,108,298 *outbound* messages served by Hotmail, randomly sampled during the period between Dec 10th, 2008 and Jan 24th, 2009. Detecting outbound email campaigns serves two purposes: reducing outbound spam and capturing abused accounts. Because legitimate marketing campaigns are usually sent from the same sender, duplicate messages sent by multiple unrelated users can therefore be reliably classified as spam and intercepted by an outbound spam filter. In addition, accounts that send these messages are very likely to be abused or hijacked by adversaries. Detecting email campaigns in outbound messages can help find and suspend these compromised accounts.

Given the large number of messages, one difficulty we encountered is how to manually find and label campaign messages in this collection. In order to efficiently construct the "gold" set of email campaigns, we first ran both I-Match and Shingling on this dataset. Two messages were treated near-duplicate and put in the same cluster (i.e., campaign) if either I-Match or Shingling predicted so. As a result, we derived 31,512 initial clusters in total. Not surprisingly, email campaigns detected in this way were not totally correct and contain quite a few false-positive and false-negative cases. To further *clean* the campaign labels, we randomly selected 875 clusters with at least two messages and corrected their labels manually. Since in reality, a spam detection system can only be trained using historical messages, we mimicked this scenario by splitting the clusters based on the time their messages were sent. 400 clusters (2,256 email messages) sent between Dec 10th, 2008 and Jan 5th, 2009 were used for training, and the remaining 475 clusters (658 email messages) sent between Jan 5th, 2009 and Jan 24th, 2009 were used for testing.

### *4.2.2 Raw similarity measure*

Table 4 shows the results of the similarity measures based on the unigram vectors. Contrary to the experiments on the news data, here we found that the Jaccard coefficient performs better than the cosine function. As for the comparisons against other weighting schemes, when the weights are learned using labeled data, the quality of the raw similarity measure is still better than TFIDF, which is again better than using the binary vector representation.

Interestingly, we also found that the NDD accuracy of SpotSigs drops sharply on this dataset. After carefully examining the unigrams it selects, we realized that although its heuristics of selecting the first non-stop words after some anchor stopwords works great

```
Subject: Outstaindng prcies and huge disuconts

sowftare you've ever watned
www.freesoftwarepark.com
_____
sowftare you've ever watned
www.freesoftwarepark.com
Invite your mail contacts to join your friends list with
Windows Live Spaces. It's easy! Try it!
```

```
Subject: Dolwnoadable software, 80% discounts

sofwtare you've ever watned
www.computercodepark.com
_____
sofwtare you've ever watned
www.computercodepark.com
Get news, entertainment and everything
you care about at Live.com. Check it out!
```

**Figure 3: Two email messages from the same campaign.**

| | Cosine | Jaccard |
|---|---|---|
| ANDD-Raw | **0.674** | **0.700** |
| TFIDF | 0.625 | 0.626 |
| Binary | 0.622 | 0.622 |
| SpotSigs | 0.257 | 0.258 |

**Table 4: MAX $F_1$ of ANDD-Raw $k$-gram vectors with $k = 1$ using the cosine and Jaccard similarity functions, compared to other $k$-gram weighting functions (binary and TFIDF) in the Email domain.**

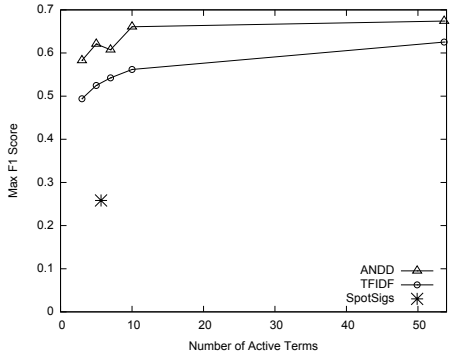


**Figure 4: Max $F_1$ vs. the number of active terms in the Email domain for ANDD, TFIDF, and SpotSigs.**

on the Web news crawls, it is much less effective on email for two reasons. Email messages are typically shorter than news articles and are written somewhat informally. As a result, stopwords occur less frequently, and each email consists only 5.7 Spot signature in average. In addition, we hypothesize that the SpotSigs heuristic effectively removes the terms in the template blocks such as ads or navigation bar, which typically don't exist in email. It is therefore not clear whether SpotSigs can capture the main content any more.

To validate these conjectures, we conduct an experiment to test the performance degradation when removing some unigrams of the vectors based on their importance, judged by the weighting score. Figure 4 shows the Max $F_1$ scores given by the cosine similarity based on ANDD-Raw and TFIDF vectors with different numbers of active unigrams in them. Although the averaged number of active unigrams is 53.7, the performance of our method does not suffer much even when the number of active unigrams is restricted to 10. When the number of active unigrams is 5, the Max $F_1$ scores of ANDD and TFIDF are 0.621 and 0.525, respectively. Both outperform SpotSigs substantially.

### *4.2.3 Signature generation*

Table 5 reports the prediction accuracy of various NDD methods in terms of Max $F_1$, along with the corresponding precision and recall. Although the Jaccard similarity on the ANDD-Raw unigram vectors performs better than the cosine version, such advantage is not fully carried after dimensionality reduction using the LSH scheme. As a result, ANDD-LSH-Cos achieves slightly

| Method | IDF | Bytes | Prec | Rec | $F_1$ |
|---|---|---|---|---|---|
| ANDD-LSH-Cos | [0,1] | 48 | 0.723 | 0.599 | 0.656 |
| ANDD-LSH-Jacc | [0,1] | 160 | 0.821 | 0.532 | 0.646 |
| Charikar | [0,1] | 48 | 0.972 | 0.425 | 0.591 |
| Shingles | [0,1] | 672 | 0.752 | 0.502 | 0.602 |
| 1-Shingles | [0,1] | 48 | 0.827 | 0.410 | 0.548 |
| I-Match | [0.4,0.75] | 24 | 0.874 | 0.382 | 0.532 |

**Table 5: Results of signature-based NDD methods for the Email domain. ANDD-LSH-Cos achieves the best Max $F_1$ despite its short signature length.**

higher Max $F_1$ score than ANDD-LSH-Jacc, and both outperform Charikar's random projection method. Due to the selection bias introduced when sampling email messages for annotation, it is not entirely fair to compare our approach with Shingles or I-Match. We list the results of these algorithms here for reference only.

## 4.3 Discussion

As shown previously, our ANDD approach not only outperforms existing methods, but can also adapt to a different domain better. Its advantages are mainly due to learning from the labeled training documents and from extracting more information from documents. In what follows we discuss how these two extra needs may impact the deployment of our algorithm in practice.

One natural question for a learning approach such as ours is how many training documents are needed to achieve satisfactory accuracy. To answer this question, we repeat our experiments using the news data set by reducing the number of training documents and record the corresponding Max $F_1$ scores. Although in our regular setting we use 1,005 labeled documents in average for training, surprisingly we found that the algorithm can in fact train a model that achieves the asymptotic result even with a small number of labeled documents. Figure 5 shows the learning curve of the model on unigrams. The initial model of random parameter values gives a 0.856 Max $F_1$ score, which is worse than the cosine similarity based on binary unigram vector (see Table 2). With as few as 60 documents, it already improves the Max $F_1$ score to 0.953, which is very close to 0.956, the Max $F_1$ score of our regular model. Such a sharp learning curve may be due to the fact that we are learning only a few model parameters (23 in these experiments). However, we want to point out that manually tuning these model parameters is still difficult and may not be much better than the initial random model weights (from preliminary experiments not reported here). Also, adding more training documents still improves the performance although the gain is relatively small.

Another practical concern of deploying our approach is the runtime efficiency. Thanks to the effective LSH scheme, our approach can encode documents in short bit-string signatures that preserve the good accuracy result of the raw vectors. The storage requirement of our method remains the same and standard indexing techniques applied to other NDD approaches can be adapted here as well (notice that training the $k$-gram weighting model is done of-

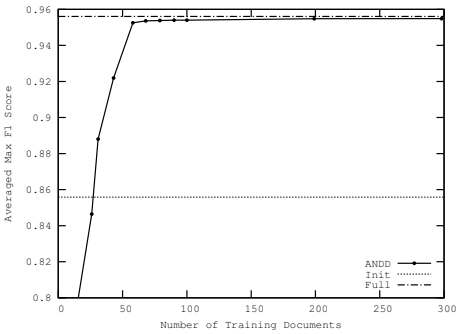


**Figure 5: Max $F_1$ vs. number of training samples compared to the random initialization (init) and fully learned model (full). Accuracy is only lowered slightly as the number of training examples decreases.**

fline). The only place that our algorithm may need slightly more computational time is feature extraction. Fortunately, we can already achieve good performance based on straightforward features listed in Table 1. Among them, the document frequency and query log frequency can be found using a simple table lookup operation. Other features such as term frequency and term position can be determined when the parser or tokenizer finishes scanning the document. Therefore, such additional computation cost should be negligible. Of course, adding more domain-dependent features is usually the most effective way to improve a learning system in practice. Whether more such features should be added will depend on the trade-off between the computational cost of extracting such features and its potential benefits to the final NDD prediction accuracy.

# 5. CONCLUSIONS

In this paper, we presented a novel adaptive near-duplicate detection (ANDD) method that achieves high accuracy consistently across different target domains. Observing that the raw document representation dictates the prediction accuracy of an NDD algorithm, we extend a recently proposed term-weighting framework [21] to learn general $k$-gram vectors and to optimize for a different similarity function such as the Jaccard coefficient. Each document is represented by an informative real $k$-gram vector. Similarity measures computed on these vectors can reliably predict near-duplicate documents. To the best of our knowledge, our method is the first that introduces similarity learning to the NDD problem. As demonstrated by the experiments done on two different problem domains, the Web news dataset and outbound email messages, our method can easily leverage a small number of labeled near-duplicate document clusters, and provide more accurate prediction results without tedious parameter tuning for the target domain. When this approach is applied to large-scale problems, efficiency is preserved by mapping the vector representation to short signatures with the help of effective locality sensitive hashing schemes.

In the future, we would like to explore different possibilities of feature engineering and model improvement to further enhance our approach. For instance, lexical features such as the words contained in each $k$-gram can be easily added. Information from deeper document analysis modules, if can be efficiently executed during runtime, will be used for feature extraction as well. Applying our NDD method to more applications for Web search and online advertising is also on our agenda.

# 6. ACKNOWLEDGMENTS

We thank Chris Meek and Susan Dumais for many useful discussions. We are also grateful to Martin Theobald for sharing the data and the SpotSigs package. We also would like to thank anonymous reviewers for their helpful comments.

# 7. REFERENCES

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

[2] A. Z. Broder. Identifying and filtering near-duplicate documents. In *COM '00*, pages 1–10. Springer-Verlag, 2000.

[3] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60:630–659, 2000.

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

[5] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, 2002.

[6] A. Chowdhury, O. Frieder, D. Grossman, and M. C. McCabe. Collection statistics for fast duplicate document detection. *ACM Trans. Inf. Syst.*, 20(2):171–191, 2002.

[7] D. Fetterly, M. Manasse, and M. Najork. On the evolution of clusters of near-duplicate web pages. In *LA-WEB '03*, 2003.

[8] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB '99*, 1999.

[9] R. J. Hall. A countermeasure to duplicate-detecting anti-spam techniques. Technical report, AT&T, 1999.

[10] M. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *SIGIR '06*, pages 284–291, New York, NY, USA, 2006. ACM.

[11] T. C. Hoad and J. Zobel. Methods for identifying versioned and plagiarized documents. *Journal of the American Society for Information Science and Technology*, 54(3):203–215, 2003.

[12] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of 30th STOC*, pages 604–613, 1998.

[13] A. Kolcz and A. Chowdhury. Hardening fingerprinting by context. In *CEAS '07*, 2007.

[14] A. Kolcz and A. Chowdhury. Lexicon randomization for near-duplicate detection with I-match. *Journal of Supercomputing*, 45(3):255–276, 2008.

[15] A. Kolcz, A. Chowdhury, and J. Alspector. Improved robustness of signature-based near-replica detection via lexicon randomization. In *KDD '04*, 2004.

[16] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *CEAS'05*, 2005.

[17] G. S. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In *WWW '07*, 2007.

[18] V. V. Prakash and A. O'Donnell. Fighting spam with reputation systems. *Queue*, 3(9):36–41, 2005.

[19] M. Rabin. Fingerprinting by random polynomials. Report TR-1581, Harvard University, 1981.

[20] M. Theobald, J. Siddharth, and A. Paepcke. Spotsigs: robust and efficient near duplicate detection in large web collections. In *SIGIR '08*, pages 563–570, 2008.

[21] W. Yih. Learning term-weighting functions for similarity measures. In *Proc. of EMNLP-09*, 2009.