Detecting near-duplicate documents using sentence-level features and supervised learning

Yung-Shen Lin, Ting-Yi Liao, Shie-Jue Lee *

Department of Electrical Engineering, National Sun Yat-Sen University, Kaohsiung 804, Taiwan

A R T I C L E   I N F O
Keywords:
Near-duplicate
Feature selection
Similarity function
Training data
Support vector machine
Discriminant function

A B S T R A C T
We present a novel method for detecting near-duplicates from a large collection of documents. Three major parts are involved in our method, feature selection, similarity measure, and discriminant derivation. To find near-duplicates to an input document, each sentence of the input document is fetched and preprocessed, the weight of each term is calculated, and the heavily weighted terms are selected to be the feature of the sentence. As a result, the input document is turned into a set of such features. A similarity measure is then applied and the similarity degree between the input document and each document in the given collection is computed. A support vector machine (SVM) is adopted to learn a discriminant function from a training pattern set, which is then employed to determine whether a document is a near-duplicate to the input document based on the similarity degree between them. The sentence-level features we adopt can better reveal the characteristics of a document. Besides, learning the discriminant function by SVM can avoid trial-and-error efforts required in conventional methods. Experimental results show that our method is effective in near-duplicate document detection.

© 2012 Elsevier Ltd. All rights reserved.

1. Introduction

As the World Wide Web is increasingly popular, digital documents are easily generated and put on the internet. By using a search engine, one can collect a large set of documents in a very short time (Chowdhury, Frieder, Grossman, & McCabe, 2002; Henzinger, 2006). Through the delete, copy, and paste commands provided by an editor or other tools (de Carvalho, Laender, Goncalves, & da Silva, 2012; Valls & Rosso, 2011), similar documents are likely to appear in various web communities (Conrad, Guo, & Schriber, 2003; Fetterly, Manasse, & Najork, 2003; Manku, Jain, & Sarma, 2007; Narayana, Premchand, & Govardhan, 2009; Pereira, Baeza-Yates, & Ziviani, 2006; Yang & Callan, 2005), e.g., blogs and forums. Such similar documents not only increase the volume of information one may have to go through but also require more storage and bandwidth for communication. To reduce the data volume and increase the search efficiency, detecting similar documents has become an important issue in the field of information retrieval (Pereira et al., 2006).

Similar documents can be divided into two categories, duplicates and near-duplicates. Two documents are duplicates if they are totally identical (Broder, 2000). Two documents are near-duplicates if one document is a modification of the other document. The modification can be insertion, deletion, or replacement of parts of the text. Due to the provision of editing facilities, near-duplicate documents are prevailing on almost all kinds of social media (Enron email dataset, 2012; Yang & Callan, 2006). Duplicate documents can be easily detected. However, detecting near-duplicates is much harder (Sood & Loguinov, 2011; Jiang & Sun, 2011). In this paper, we focus on how to detect near-duplicate documents efficiently and effectively.

To detect near-duplicate documents, one can adopt the bag-of-words model (Bag of words, 2012) for document representation. Let \( D = \{ d_1, d_2, \ldots, d_n \} \) be a set of \( n \) documents, in which \( d_1, d_2, \ldots, d_n \) are individual documents. Each document \( d_i \), \( 1 \leq i \leq n \), is represented by a feature set \( f_i = \{ f_{i1}, f_{i2}, \ldots, f_{im} \} \) where \( m \) is the number of features selected for \( D \). A feature can be anything associated with documents (Arasu, Ganti, & Kaushik, 2006; Fagin, Kumar, & Sivakumar, 2003; Gong, Huang, Cheng, & Bai, 2008; Huffman et al., 2007; Li, Wang, & Yang, 2007; Qiu & Zeng, 2010; Theobald, Siddharth, & Paepcke, 2008; Wang & Chang, 2009; Xiao, Wang, Lin, & Shang, 2009, 2011; Yang & Callan, 2006). Usually, terms appearing in documents are taken to be the basis of features (Goyal, Behera, & McGinnity, 2012; Han, Finin, McNamee, Joshi, & Yesha, 2012; Kim & Lee, 2012; Luo, Lin, Wang, & Zhou, 2007). To investigate how one document is similar to another document, one can calculate the similarity degree between the two sets of features corresponding to these two documents (Bayardo, Ma, & Srikant, 2007; Huang, Wang, & Li, 2008; Zhao, Wang, Liu, & Ye, 2011). The higher the degree is, the more the documents are similar to each other. Conventionally, a manually designated

* Corresponding author.
E-mail address: leesj@mail.ee.nsysu.edu.tw (S.-J. Lee).
threshold is provided by the user in advance. If the similarity degree is equal to or higher than the threshold, the two documents are near-duplicates. Otherwise, they are not. However, different choices on features may require different settings of the threshold. Besides, the determination of a satisfactory threshold is also a problem. Usually, trial-and-error cannot be avoided. Setting a good threshold manually is neither an easy task nor an effective way for near-duplicate document detection.

We develop a novel method for detecting near-duplicates from a large collection of documents. Our method consists of three major components, feature selection, similarity measure, and discriminant derivation. To find near-duplicates to an input document, we first do preprocessing, e.g., removing stop words and punctuation marks, on the input document. Then for each sentence, the weight of each term is calculated, and the heavily weighted terms are selected to be the feature of the sentence. As a result, the input document is turned into a set of features. Then the similarity degree between the input document and each document in the given collection is computed. Finally, we use a support vector machine to learn a classifier from a training pattern set (Arnosti & Kalita, 2011; Brin, Davis, & Garcia-Molina, 1995; Hajishirzi, Yih, & Kolcz, 2010; Martins, 2011). A discriminant function is derived, which is then used to determine whether a document is a near-duplicate to the input document based on the similarity degree between them. Our method has several advantages. The sentence-level features we adopt can better reveal the characteristics of a document, and learning the discriminant function by SVM can avoid trial-and-error efforts required in conventional methods. Experimental results show that our method is effective in near-duplicate document detection.

The rest of this paper is organized as follows. Section 2 presents a brief description about related work. Section 3 details our proposed near-duplicate document detection method. The way to create a feature set for a given document and the adoption of SVM in learning a discriminant function are described. Experimental results are presented in Section 4. Comments on a frequency-based representation are discussed in Section 5. Finally, a conclusion is given in Section 6.

### 2. Related work

For a document, Shingles (Manning, Raghavan, & Schutze, 2008) divided it into a series of strings. Each string is k words long, called a k-gram. The list of such k-grams is taken to be the feature set of this document. This method may result in a large feature set. For example, if a document consists of L words, the feature set of the document contains $L - k + 1$ elements. Some improvements to Shingles have been proposed. Li et al. (2007) took discontinuous k-grams by skipping the words in between. The strings between two pause symbols are treated as features. The SpotSigs proposed by Theobald et al. (2008) used stop words (Common Stopword set, 2012) instead. A feature is taken to be a string starting with a stop word. For example, [the super computer] and [a good movie] are elements of the feature set. However, the stop word list adopted is a key factor to the feature set obtained. Different stop word lists lead to different feature sets for a given document. A popular stop word list used in many applications is shown in Table 1. SpotSigs (Theobald et al., 2008) adopts some rules to cut down the size of a feature set, e.g., preferring more frequently used stop words. Other methods based on sentences were proposed (Wang & Chang, 2009). With these methods, each individual sentence of a document is divided into a series of k-grams. The union of the k-grams of all the sentences is taken as the feature set of the document. However, these methods result in large feature sets for document representation.

A similarity function is used to calculate the similarity degree of any two documents. Let $d_1 = \{f_1,1, f_1,2, \ldots , f_{1,m}\}$ and $d_2 = \{f_2,1, f_2,2, \ldots , f_{2,m}\}$ be the feature sets of documents $d_1$ and $d_2$, respectively. Some popular similarity functions are listed below.

- **Jaccard function:**
  \[
  \text{sim}(d_1, d_2) = \frac{f_1 \cap f_2}{f_1 \cup f_2}
  \]  
  where $\cap$ stands for the AND operation and $\cup$ for the OR operation in the set theory.

- **Cosine function:**
  \[
  \text{sim}(d_1, d_2) = \frac{f_1 \cdot f_2}{\|f_1\| \cdot \|f_2\|}
  \]  
  where $f_1 \cdot f_2$ is defined to be $f_1,1 \cdot f_2,1 + f_1,2 \cdot f_2,2 + \ldots + f_{1,m} \cdot f_{2,m}$ and $\|f\| = \sqrt{f_i^2}$ for $i = 1, 2$.

- **Euclidean distance:**
  \[
  \text{sim}(d_1, d_2) = \frac{f_1 - f_2}{\|f_1 - f_2\|}
  \]  
  where $f_1 - f_2 = \{f_{1,1} - f_{2,1}, f_{1,2} - f_{2,2}, \ldots , f_{1,m} - f_{2,m}\}$.

- **Extended Jaccard function:**
  \[
  \text{sim}(d_1, d_2) = \frac{f_1 \cdot f_2}{f_1 + f_2 - f_1 \cdot f_2}
  \]  

- **Dice function:**
  \[
  \text{sim}(d_1, d_2) = \frac{2f_1 \cdot f_2}{f_1 + f_2}
  \]  

Note that $\text{Eff}(d_1, d_2)$ is an extended version of $\text{J}(d_1, d_2)$ and $\text{D}(d_1, d_2)$ is a simplified version of $\text{Eff}(d_1, d_2)$. Conventionally, a manually pre-designated threshold provided by the user is required to determine if two documents are near-duplicates. Supervised learning techniques, in particular support vector machines (SVM) (Martins, 2011), can be applied to determine optimally whether two documents are near-duplicates automatically. Given a training data set with instances belonging to one of two classes, near-duplicate and non-near-duplicate, SVM learns how to separate the instances of one class from the instances of the other class. As matter of fact, an optimal hyperplane can be derived which not only separates the instances on the right side of the hyperplane but also maximizes the margin from the hyperplane to the instances closest to it on either side. If the problem is not linearly separable, one can map the original space to a new space by using nonlinear basis functions. It is generally the case that this new space has many more dimensions than the original space, and, in the new space, the optimal hyperplane can be found.
3. Proposed approach

In general, replaced terms, inserted terms, and missed terms are cases frequently occurring in near-duplicate documents. For the near-duplicate detection methods based on terms, e.g., Shingles, different document representations may be adopted and diverse results can be obtained. For example, suppose we have a sentence A of the following.

A: People rally on the sidewalk as legal arguments over the Patient Protection and Affordable Care Act take place at the Supreme Court.

By replacing sidewalk with pavement, we get another sentence B.

B: People rally on the pavement as legal arguments over the Patient Protection and Affordable Care Act take place at the Supreme Court.

Sentence A and B have the same meaning, and it is no doubt that they are near-duplicates. However, existing methods may obtain different representations for these two sentences. For example, the k-gram feature selection method (Manning et al., 2008) produces several k-grams containing the word sidewalk for sentence A, while it produces several k-grams containing the word pavement for sentence B. Therefore, different methods define different representations. Also, for the k-gram method, different values of k could affect both the feature set size and the required computation power. To solve this problem, we propose a method for extracting features from individual sentences in a way to better reveal the characteristics of a document. The method turns out to be more invariant against insertion, deletion, or replacement of terms. As a result, the feature sets obtained are more suitable for near-duplicate document detection.

3.1. Feature sets based on sentences

We adopt individual sentences as basic units for feature selection since sentences are more representative than terms. In daily life, we have usually found that a certain sentence can be the representative of a well-known book or article. A representative sentence may easily remind people of the book containing this sentence. Besides, a sentence has a certain structure. A modified version of a sentence is highly likely to contain the same terms that appear in the original version. From these observations, we propose using weighted keywords to represent individual sentences. The weight of a keyword is determined by the tf-idf of the keyword. The tf-idf of a word is the product of the term frequency (tf) and the inverse document frequency (idf) of the word (Manning et al., 2008). For sentence I, let \( I_0 \) be the modified sentence after preprocessing is done. Then we compute the weight for each remaining word, and sort the words in descending order in terms of their weights. For example, the following is sentence A after stop words and punctuation marks are deleted:

\[ A_0: \text{rally sidewalk legal arguments patient protection affordable care supreme court.} \]

The ordering of the terms sorted by tf-idf is

\[
\text{supreme} \geq \text{affordable} \geq \text{protection} \geq \text{patient} \geq \text{legal} \\
\geq \text{sidewalk} \geq \text{argument}. 
\]

For sentence B, the ordering of the terms sorted by tf-idf is

\[
\text{supreme} \geq \text{affordable} \geq \text{protection} \geq \text{patient} \geq \text{pavement} \\
\geq \text{legal} \geq \text{argument}. 
\]

The set of the top k words are selected as the feature of the underlying sentence. For example, let k be 4. The feature \( f_A \) obtained for sentence A contains four words, supreme, affordable, protection, and patient, i.e.,

\[ f_A = (\text{supreme}, \text{affordable}, \text{protection}, \text{patient}). \]

Note that the ordering of the words in \( f_A \) matters. The feature \( f_B \) obtained for sentence B also contains these four words:

\[ f_B = (\text{supreme}, \text{affordable}, \text{protection}, \text{patient}). \]

Therefore, sentences A and B are identical from the viewpoint of features. This property is good for near-duplicate document detection. For a document, we represent it by a feature set which is the union of the features of all the sentences contained in this document. The procedure of finding the feature set for a document can be summarized below.

---

**Input:** a document \( d \)
**Output:** the feature set \( f_d \) for \( d \)

**Procedure** find feature set \( (d) \)

1. For each sentence \( s \) in \( d \)
   - Perform preprocessing on \( s \)
   - Calculate the weight for each remaining word
   - Sort the words in descending order
   - Let \( f \) contain the top \( k \) words
2. Add \( f \) into \( f_d \)

**End Procedure**

With this method, near-duplicate documents tend to have identical or very similar representations. Note that the number of words selected from a sentence may affect the performance of near-duplicate document detection. With fewer words, the representation of a document can be shorter in size and the computation time for later processing can be smaller. However, detection accuracy can be worse.

3.2. Preparing training patterns

Conventionally a predesignated threshold is required to determine whether two documents are near-duplicate to each other. Setting such a threshold manually is quite difficult and needs a lot of trial-and-error efforts. To alleviate this difficulty, we adopt a support vector machine (SVM) (Martins, 2011) to learn a classifier based on a set of training patterns. Before going on, we define the similarity vector between two documents \( d_1 \) and \( d_2 \), denoted \( S(d_1, d_2) \), as follows.

**Definition 1.** The similarity vector between two documents \( d_1 \) and \( d_2 \) is defined as \( S(d_1, d_2) = (x_1, x_2, \ldots, x_r) \) where \( x_i \), \( 1 \leq i \leq r \), is the similarity degree computed for \( d_1 \) and \( d_2 \) by a certain similarity function.

The value of \( r \) and the similarity functions involved can be selected by the users. For example, we can let

\[ x_1 = f(d_1, d_2) = \text{Ec}(d_1, d_2) \text{ for } r = 2. \]

A training pattern set is derived from a collection of training documents. Each training pattern in the training data set involves a pair of training documents \( d_1 \) and \( d_2 \) which are known in advance to be near-duplicate or non-near-duplicate to each other, and is expressed as \((x, t)\) where:

- \( x = S(d_1, d_2) \) is the similarity vector between \( d_1 \) and \( d_2 \);
- \( t \) is the desired output, and \( t = +1 \) if \( d_1 \) and \( d_2 \) are near-duplicate and \( t = -1 \) if \( d_1 \) and \( d_2 \) are non-near-duplicate.
The process of preparing a training pattern set is summarized below.

<table>
<thead>
<tr>
<th>Input: a set D of training documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: the derived training pattern set X</td>
</tr>
</tbody>
</table>

Procedure derive_training_pattern_set (D)

For each pair of documents d1 and d2 in D

- Calculate the similarity vector \( x = S_i(d_1, d_2) \)
- If \( d_1 \) and \( d_2 \) are near-duplicate
  - Set \( t = +1 \)
- Else
  - Set \( t = -1 \)
- Add \( (x, t) \) into \( X \)

End Procedure

The output training pattern set is then used by SVM as described below.

3.3. Discriminant derivation

When the training pattern set is ready, SVM works on the set and finds a hyperplane \( g(x) = 0 \) which optimally separates the training patterns with \( t = +1 \) from those with \( t = -1 \). Let the resulting training pattern set contain \( N \) training patterns, denoted as \( X = \{(x_i, t_i) | 1 \leq j \leq N \} \). We’d like to minimize

\[
L_p = \frac{1}{2} ||w||^2 + C \sum_j \zeta_j,
\]

where \( w \) is the coefficient vector of the hyperplane to be optimized, \( C \) is the penalty factor, and \( \zeta_j \geq 0, 1 \leq j \leq N \), are slack variables, subject to the constraints

\[
t_iw^T\Phi(x_i) \geq 1 - \zeta_j, 1 \leq j \leq N,
\]

where \( \Phi(x) = (\phi_1(x), \phi_2(x), \ldots, \phi_h(x)) \) is a mapping from the \( r \)-dimensional \( x \) space to the \( h \)-dimensional \( z \) space

\[
z = \Phi(x),
\]

with \( z_i = \phi_i(x), 1 \leq i \leq h \). The hyperplane in the \( h \)-dimensional \( z \) space is \( g(z) = 0 \), and the desired discriminant function in the \( x \) space is

\[
g(x) = w^T\Phi(x) = \sum_{i=1}^{h} w_i \phi_i(x).
\]

Eq. (6) can be converted to the dual

\[
L_d = \sum_j a_j - \frac{1}{2} \sum_j \sum_k a_j a_k t_j t_k K(x_j, x_k),
\]

where \( K(x_j, x_k) = \Phi(x_j)^T \Phi(x_k) \) is a kernel function, and \( a_j \) and \( a_k \) are Lagrange multipliers. Eq. (10) should be maximized with respect to \( a_j \), subject to

\[
\sum_j a_j t_j = 0 \land 0 < a_j < C, 1 \leq j \leq N.
\]

Eq. (10) can be solved using quadratic optimization methods (Martins, 2011). Then the discriminant \( g(x) \) is obtained by

\[
g(x) = \sum_j a_j K(x_j, x),
\]

which will be used in the testing phase.

3.4. Testing phase

Once the discriminant \( g(x) \) is obtained, we can use it to determine if a document \( d_c \) is a near-duplicate to an input document \( d_I \) as follows. Firstly, the feature sets \( f_c \) and \( f_I \) of \( d_c \) and \( d_I \), respectively, are obtained. Then we calculate the similarity vector \( S_i(d_c, d_I) \). Let \( x = S_i(d_c, d_I) \). Then we calculate \( g(x) \). If \( g(x) \geq 0 \), \( d_c \) is determined to be a near-duplicate to \( d_I \).

3.5. System operation

Fig. 1 shows the flowchart of our proposed algorithm for near-duplicate document detection. Fig. 1(a) shows the training phase of the algorithm. Given a set of training documents, we calculate the feature set for each document by procedure find_feature_set. Then we derive a training pattern set by procedure derive_training_pattern_set. Then we employ SVM to build a classifier based on the training pattern set. A discriminant function for separating the documents of class \( +1 \) and class \( -1 \) apart is derived. Fig. 1(b) shows the testing phase of the algorithm. Given an input document \( d_I \), we want to retrieve all documents in \( D_c \) that are near-duplicates to \( d_I \). We calculate the feature sets for \( d_I \) and each document \( d_c \) in \( D_c \). We calculate the similarity vector between \( d_I \) and \( d_c \). Then we feed the similarity vector to the discriminant function obtained in the training phase. If the returned value is equal to or greater than 0, \( d_c \) is determined to be a near-duplicate to \( d_I \). Otherwise, \( d_c \) is not a near-duplicate to \( d_I \).

3.6. Example

We give an example here to illustrate how our method works. Suppose we have a document \( d_I \):

Markets in Britain, Germany and France were all up about 2 percent in early trading Monday, while shares in Spain – led by financial stocks – surged nearly 4 percent. Analysts said the bailout deal reached Saturday is an important step in shoring up Spain’s banks, but cautioned it is not a definitive solution for the country’s economy as a whole. Investor relief could be short-lived, as voters in Greece prepare for parliamentary elections Sunday, and the possibility of a new government prepared to leave the euro currency rather than face austerity measures demanded by other nations.
which contains three sentences. After preprocessing, we have the following three sentences:

- \( s_{1,1} \): market britain germany france percent trading Monday share spain1 financial stock surg;
- \( s_{1,2} \): analyst bailout deal reach Saturday important step shoring spain bank caution definitive solution country economy;
- \( s_{1,3} \): investor relief shortliv voter greece parliamentary election Sunday possibility government leave euro currency austerity measure demand nation.

We go ahead to derive the feature for each sentence. The tf-idf values for each term in \( s_{1,1} \) is shown in Table 2. Suppose we have \( r = 4 \). The top four terms in Table 2, as indicated in boldface, are spain, respectively, as feature set for \( s_{1,1} \). We consider two cases: to find the similarity degree between \( s_1 \) and \( s_2 \), as shown in Table 3. Suppose we have \( f_1 = \{f_{1,1}, f_{1,2}, f_{1,3}\} \). Suppose we have another document \( d_2 \), consisting of four sentences, and the feature set for \( d_2 \) is \( f_2 = \{f_{2,1}, f_{2,2}, f_{2,3}, f_{2,4}\} \) where

\[
\begin{align*}
  f_{1,1} &= \text{spain, spanish, stabilizing, rescue}, \\
  f_{1,2} &= \text{spanish, mariano, rajoy, eurozone}, \\
  f_{1,3} &= \text{spanish eurozone, chiefly}, \\
  f_{1,4} &= \text{spanish, robert, halver, baader}.
\end{align*}
\]

Let’s have one more document \( d_3 \) whose feature set is \( f_3 = \{f_{3,1}, f_{3,2}, f_{3,3}, f_{3,4}\} \) where

\[
\begin{align*}
  f_{1,1} &= \text{eurozone, ailing, investor}, \\
  f_{1,2} &= \text{surg, britain, germany}, \\
  f_{1,3} &= \text{nikei, japan, hang, seng}, \\
  f_{1,4} &= \text{eurozone, chiefly}, \\
  f_{1,5} &= \text{shortliv, austerity, euro, greece}.
\end{align*}
\]

To find the similarity degree between \( d_1 \) and \( d_2 \) using Jaccard function, we have

\[
\begin{align*}
  J(d_1, d_2) &= \frac{|f_1 \cap f_2|}{|f_1 \cup f_2|} = \frac{0 + 0 + 0 + 0 + 0 + 0 + 0 + 0}{1 + 1 + 1 + 1 + 1 + 1 + 1 + 1} = 0
\end{align*}
\]

Similarly,

\[
\begin{align*}
  J(d_1, d_3) &= \frac{1 + 1 + 1 + 0 + 0}{1 + 1 + 1 + 1 + 1} = 0.6 \\
  J(d_2, d_3) &= \frac{0 + 0 + 0 + 0 + 0 + 0 + 0 + 0}{1 + 1 + 1 + 1 + 1 + 1 + 1 + 1} = 0
\end{align*}
\]

Let the similarity vector contain the Jaccard similarity as its only component. We consider two cases:

- Training: Suppose we know that \( d_1 \) and \( d_2 \) are non-near-duplicate, \( d_1 \) and \( d_2 \) are near-duplicate, and \( d_2 \) and \( d_3 \) are non-near-duplicate. Then we have three training patterns, \( (0,0,0), (0.6,1) \), and \( (0,0,0) \), respectively. Using such training patterns and others to train a SVM, a discriminant \( g(x) \) can be obtained.
- Testing: Suppose we have already got a discriminant \( g(x) \) learned from a set of training patterns. We want to know whether \( d_1 \) and \( d_3 \) are near-duplicate or not. We compute \( g((0.6)) \). If \( g((0.6)) \geq 0 \), we conclude that \( d_1 \) and \( d_3 \) are near-duplicate. If \( g((0.6)) < 0 \), we conclude that \( d_1 \) and \( d_3 \) are non-near-duplicate.

### 4. Experiments

In this section, results of several experiments are presented. Comparisons with other methods are also given to demonstrate the effectiveness of our proposed approach. Two sets of documents, Enron and RCV1, are involved in the experiments. The Enron document set [Enron email dataset, 2012] includes the mailbox e-mails of 150 users. Most of the users are executives of Enron. The Enron set contains 128,173 e-mails and takes 310 MB in size, as shown in Table 3. The RCV1 document set (RCV1, 2012) was edited and collected by Reuters. It includes 223,496 full English text news from 20 August 1996 to 30 November 1996, as also shown in Table 3. Each news in RCV1 contains 119 words and 13 sentences on average.

For convenience, our proposed feature selection method is abbreviated as KSBS (Keyword Set Based on Sentences). For the top \( k \) words selected to form the feature of a sentence, our method is indicated as KSBS-\( k \). For example, KSBS-3 is for top 3 words selected and KSBS-4 is for top 4 words selected. The methods we compare with are 3-gram (Manning et al., 2008) and SpotSigs (Theobald et al., 2008). Two versions of SpotSigs are considered, SpotSigs-Co and SpotSigs-Ch. SpotSigs-Co stands for SpotSigs (Common) which adopts the common stop word list, while SpotSigs-Ch stands for SpotSigs (Choices) which uses the selected stop word list. We define three versions of accuracy, \( A_p \), \( A_n \), and \( A_v \), as follows:

\[
\begin{align*}
  A_p &= \frac{C_p}{N_p} \\
  A_n &= \frac{C_n}{N_n} \\
  A_v &= \frac{C_p + C_n}{N_p + N_n}
\end{align*}
\]

where \( C_p \) stands for the number of near-duplicate testing patterns classified correctly, \( N_p \) for the total number of near-duplicate testing patterns, \( C_n \) for the number of non-near-duplicate testing patterns classified correctly, and \( N_n \) for the total number of non-near-duplicate testing patterns. All the programs that follow were implemented as a compact Java prototype and run on an Intel Core i5 quad-core CPU 2.80 Ghz with 8.00 GB RAM.

### 4.1. Experiment I

The Enron document set [Enron email dataset, 2012] contains many spam e-mails which are just slight modifications of the original e-mails. So the set contains many e-mails which are similar to each other. For any two documents \( X \) and \( Y \) in the Enron set, we derive the feature sets \( f_X \) and \( f_Y \) for \( X \) and \( Y \), respectively, using the 3-gram method (Manning et al., 2008). Then we compute the

### Table 2

<table>
<thead>
<tr>
<th>Words</th>
<th>tf-idf</th>
<th>Words</th>
<th>tf-idf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market</td>
<td>0.2121</td>
<td>Britain</td>
<td>0.2533</td>
</tr>
<tr>
<td>Germany</td>
<td>0.2512</td>
<td>France</td>
<td>0.2510</td>
</tr>
<tr>
<td>Percent</td>
<td>0.2057</td>
<td>Trading</td>
<td>0.2309</td>
</tr>
<tr>
<td>Monday</td>
<td>0.2178</td>
<td>Share</td>
<td>0.2213</td>
</tr>
<tr>
<td>Spain</td>
<td>0.5571</td>
<td>1</td>
<td>0.2489</td>
</tr>
<tr>
<td>Financial</td>
<td>0.2213</td>
<td>Stock</td>
<td>0.2195</td>
</tr>
<tr>
<td>Surg</td>
<td>0.3125</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 3

Two sets of documents used in experiments.

<table>
<thead>
<tr>
<th></th>
<th># of doc</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enron</td>
<td>128,173</td>
<td>310</td>
</tr>
<tr>
<td>RCV1</td>
<td>223,496</td>
<td>652</td>
</tr>
</tbody>
</table>
similarity degree \( \text{sim}(X, Y) \). If \( \text{sim}(X, Y) \geq 0.75 \), \( X \) and \( Y \) are labeled to be near-duplicates. Otherwise, \( X \) and \( Y \) are labeled as non-near-duplicates. We randomly choose 2,500 pairs of near-duplicate documents and 2,500 pairs of non-near-duplicate documents, from which 500 pairs of near-duplicate documents and 500 pairs of non-near-duplicate documents are chosen for training documents and the rest are used for testing. The fivefold cross validation is adopted and the results are the average of the five runs. Note that near-duplicate document pairs are converted to positive patterns and non-near-duplicate document pairs are converted to negative patterns.

**Table 4** shows comparisons on \( A_4 \), with \( \text{sim}(X, Y) \) being computed by Jaccard for labeling near-duplicate and non-near-duplicate documents. In this table, the first column in each row indicates the similarity function applied in the derivation of training and testing patterns. For example, the first row indicates Jaccard is applied and the second row means Dice is applied. All the similarity vectors involved are one-dimensional in this case. Note that SpotSigs-Co and SpotSigs-Ch get the same results since they generate identical feature sets for the training and testing documents. We can see KSBS-4 offers the most accurate results for all the cases in **Table 4**. For example, KSBS-4 with Euclidean gets 98.06% in accuracy, while SpotSigs-Co and SpotSigs-Ch with Euclidean only get 94.91%.

**Table 5** shows comparisons on \( A_4 \), with \( \text{sim}(X, Y) \) being computed differently for labeling near-duplicate and non-near-duplicate documents. In this table, the first column in each row indicates the similarity function applied in labeling near-duplicate and non-near-duplicate documents using 3-gram. For example, the first row indicates Jaccard is applied and the second row means Dice is applied. However, Jaccard is adopted in computing the similarity vector for training and testing patterns. Note that KSBS-4 performs best for all the cases in **Table 5**. For example, with Dice for labeling, KSBS-4 gets 98.05% in accuracy, while SpotSigs-Co and SpotSigs-Ch get 97.37% and 94.39%, respectively.

**Table 6** shows comparisons on \( A_4 \), based on 2-dimensional similarity vectors, with \( \text{sim}(X, Y) \) being computed differently for labeling near-duplicate and non-near-duplicate documents. In this table, the first column in each row indicates the similarity function applied in labeling near-duplicate and non-near-duplicate documents using 3-gram. For example, the first row indicates Jaccard is applied and the second row means Dice is applied. However, Jaccard and Dice are adopted for the similarity vector for training and testing patterns. That is, the similarity vector is two-dimensional, with the first component being computed by Jaccard and the second component being computed by Dice. Note that KSBS-4 performs best for all the cases in **Table 6**.

Finally, we compare for the case of adopting four similarity functions, Jaccard, Dice, Cosine, and Euclidean, in the derivation of training and testing patterns. That is, the similarity vector is four-dimensional. The result is shown in **Table 7**. Note that KSBS-4 performs best for all the cases in **Table 7**. Note also that having more dimensions in the similarity vector does not necessarily mean a better performance. For example, KSBS-4 with Cosine has 97.81% in accuracy in **Table 7** which is better than 97.67% of its counterpart in **Table 6**. However, KSBS-4 with Dice has 97.94% in accuracy in **Table 7** which is less than 98.05% of its counterpart in **Table 6**. Similar situations happen for SpotSigs-Co and SpotSigs-Ch.

### 4.2. Experiment II

In this experiment, we examine the performance of KSBS-k with different \( k \) settings for the Enron document set. **Table 8** shows comparisons on \( A_4 \), with \( \text{sim}(X, Y) \) being computed by Jaccard for labeling near-duplicate and non-near-duplicate documents. In this table, the first column in each row indicates the similarity function applied in labeling near-duplicate and non-near-duplicate documents using 3-gram. For example, the first row indicates Jaccard is applied and the second row means Dice is applied. Jaccard is adopted in computing the similarity vector for training and testing patterns. Note that KSBS-3 performs best and KSBS-5 performs worst in all the cases in **Table 8**.

**Table 9** shows comparisons on \( A_4 \), with \( \text{sim}(X, Y) \) being computed by Jaccard for labeling near-duplicate and non-near-duplicate documents. In this table, the first column in each row indicates the similarity function applied in labeling near-duplicate and non-near-duplicate documents using 3-gram. For example, the first row indicates Jaccard is applied and the second row means Dice is applied. However, Jaccard and Dice are adopted for the similarity vector for training and testing patterns. For convenience, J stands for Jaccard, D for Dice, C for Cosine, and E for Euclidean. Different combinations are applied in different rows. For example, the J–D in the first row indicates Jaccard and Dice are applied, the D–C in the second row indicates that Dice and Cosine are applied, and the J–D–C–E in the last row indicates that all the four similarity functions are applied. Note
that KSBS-3 performs best and KSBS-5 performs worst in all the cases in Table 9.

4.3. Experiment III

The RCV1 document set (RCV1, 2012) contains English text news edited and collected by Reuters. From this set, we randomly choose 2,000 documents. Each of these documents is evenly divided into six parts, and one to three parts are randomly chosen and modified by insertion, deletion, or replacements of terms. The resulting document is taken as a near-duplicate to the original document. We also randomly choose another 2,000 documents. Four to six parts of each of these documents are randomly chosen and modified by insertion, deletion, or replacements of terms, and the resulting document is taken as a non-near-duplicate to the original document. Among the 4,000 pairs of documents, 200 pairs of near-duplicate documents and 200 pairs of non-near-duplicate documents are chosen for training documents and the rest are used for testing. The ten-fold cross validation is adopted and the results are the average of the ten runs.

Table 10 shows comparisons on accuracy of different methods, using Jaccard for similarity vectors. Note that SpotSigs-Ch performs very well in $A_p$, but not well in $A_n$. On the contrary, SpotSigs-Co performs very well in $A_n$, but not well in $A_p$. KSBS-4 works well in both $A_p$ and $A_n$. However, 3-gram performs worse in all cases.

Table 11 shows comparisons on accuracy of different methods, using Dice and Cosine for similarity vectors. Table 12 shows comparisons on accuracy of different methods, using Jaccard, Dice and Cosine for similarity vectors. Finally, Table 13 shows comparisons on accuracy of different methods, using all four functions for similarity vectors. Clearly, KSBS-4 can work well in both $A_p$ and $A_n$. SpotSigs-Ch can only perform well in $A_p$, while SpotSigs-Co can only perform well in $A_n$. Again, 3-gram performs worst in all cases.

4.4. Experiment IV

In this experiment, we examine the performance of KSBS-\(k\) with different \(k\) settings for the RCV1 document set. Table 14 shows comparisons on $A_k$ of KSBS-\(k\) with different \(k\) settings, based on one-dimensional similarity vectors. Note that KSBS-4 offers the most accurate results, while KSBS-5 provides least accurate results, for all the cases in Table 14. Table 15 shows comparisons on $A_k$ of KSBS-\(k\) with different \(k\) settings, based on multi-dimensional similarity vectors. Again, KSBS-4 offers the most accurate results, while KSBS-5 provides least accurate results, for all the cases in Table 15.

4.5. Experiment V

In this experiment, we show the amount of resources demanded by different methods for the RCV1 document set. A comparison on the size of the preprocessed document sets is shown in Table 16. Note that for KSBS and 3-gram, the stop words are eliminated after preprocessing. Therefore, the size of the preprocessed document set for KSBS and 3-gram is much smaller than that for SpotSigs, i.e., 189.1 MB <269.3 MB. A comparison on the size of the storage involved in indexing is shown in Table 17. In this table, the column ‘Indexed table’ includes dictionary and posting lists required for indexing (Manning et al., 2008). For KSBS-\(k\), additional storage is required for storing the appearing frequency of each term, as shown in the ‘Indexed terms’ column. For SpotSigs-Co, the table for storing the stop words takes 592 bytes. However, SpotSigs-Ch only considers 4 stop words which take 18 bytes. Note that the amount of storage required by KSBS-\(k\) is much less than that of the other methods. For example, KSBS-4 requires 70.19 MB, while SpotSigs-Co 236.6 MB and 3-gram 646.6 MB, respectively.

Next, we compare the time efficiency of different methods. Since KSBS-4 runs almost as fast as KSBS-3 and KSBS-5, we only consider KSBS-4 here. A comparison on testing time of different methods, using Jaccard for similarity vectors, is shown in Fig. 2. The horizontal line indicates how many input documents are tested. The vertical line shows the total number of seconds for testing a certain number of input documents for near-duplicates. Some
values taken from Fig. 2 are shown in Table 18. We can see that KSBS-4 runs faster than the other methods. For example, KSBS-4 takes 6.78 s for testing 30,000 input documents, while SpotSigs-Ch takes 9.80 s, SpotSigs-Co takes 19.38 s, and 3-gram takes 37.52 s. Another comparison, using Jaccard and Dice for similarity vectors, is shown in Fig. 3. Some values taken from Fig. 3 are shown in Table 19. Again, we can see that KSBS-4 runs faster than the other methods. However, the times in Table 19 are nearly identical to those listed in Table 18. This shows that processing two-dimensional similarity vectors takes about the same time as processing one-dimensional similarity vectors.
5. Frequency-based representation

The feature sets defined in Section 3.1 are binary. That is, a feature is considered to appear or not appear in a document. As long as a feature appears, the number of its occurrences does not matter. For example, the feature set \( \{ x, y, x, x \} \) is identical to the feature set \( \{ x, y, x \} \) where \( x \) and \( y \) are features derived from individual sentences. For both sets, \( x \) is associated with attribute value 1 although \( x \) appears three times in \( \{ x, y, x, x \} \). Here we discuss the case of taking the frequency of a feature into consideration. The number of occurrences counts. So \( x \) in \( \{ x, y, x, x \} \) is associated with attribute value 3 while \( x \) in \( \{ x, y, x, x \} \) is associated with attribute value 1. We show the effects resulted from this consideration for the RCV1 document set.

Table 20 shows comparisons on accuracy of different methods, using Extended Jaccard for similarity vectors. Note that we use Extended Jaccard (EJaccard) instead of Jaccard, since Jaccard is only applied in the binary-based case. By comparing to Table 10, we can see that the accuracies vary but not much. For example, SpotSigs-Co gets 97.68% in \( A_1 \) in the binary-based case, while it gets 98.03% in the frequency-based case. Table 21 shows comparisons on accuracy of different methods, using Extended Dice and Cosine for similarity vectors. Again, the accuracies offered in the frequency-based case do not differ much from those offered in the binary-based case as shown in Table 11.

Table 22 shows comparisons on \( A_1 \) of KSBS-\( k \) with different \( k \) settings, based on one-dimensional similarity vectors. By comparing to Table 14, we can see that KSBS-3 and KSBS-4 perform a bit better than before. For example, KSBS-3 with Cosine gets 98.18% in the frequency-based case while it gets 97.92% in the binary-based case. KSBS-4 with Euclidean gets 98.25% in the frequency-based case while it gets 98.10% in the binary-based case. However, KSBS-5 performs a bit worse than before. For example, KSBS-5 with Cosine gets 96.60% in the frequency-based case while it gets 96.65% in the binary-based case. Table 23 shows comparisons on \( A_1 \) of KSBS-\( k \) with different \( k \) settings, based on multi-dimensional similarity vectors. Again, by comparing with Table 15, we can see that KSBS-3 and KSBS-4 perform a bit better while KSBS-5 performs slightly worse than before.

Next, we compare the time efficiency of different methods. Since KSBS-4 runs almost as fast as KSBS-3 and KSBS-5, we only consider KSBS-4 here. A comparison on testing time of different methods, using Extended Jaccard and Dice for similarity vectors, is shown in Fig. 4. Some values taken from Fig. 4 are shown in Table 24. We can see that KSBS-4 runs faster than the other methods. However, by comparing to Fig. 3 and Table 19, we can see that the frequency-based version runs almost as fast as the binary-based version.

6. Conclusion

Digital documents are easily generated and put on the internet. Through the delete, copy, and paste commands provided by an editor or other tools, near-duplicate documents are likely to appear in various web communities, e.g., blogs and forums, increasing the volume of information one may have to go through and requiring more storage and bandwidth for communication. Detecting near-duplicates is an important issue in the field of information retrieval. However, it is not an easy task. We have presented a novel method for detecting near-duplicates from a large collection of documents. Our method consists of three major components, feature selection, similarity measure, and discriminant derivation. For an input document, some preprocessing work is done on it. Then for each sentence, the heavily weighted terms are selected to be the feature of the sentence. As a result, the input document is turned into a set of features. Then the similarity degree between the input document and each document in the given collection is computed. Finally, a discriminant function is derived from a SVM-based classifier, which is then used to determine whether a document is a near-duplicate to the input document based on the similarity degree between them. Our method has several advantages. The sentence-level features we adopt can better reveal the characteristics of a document, and learning a discriminant by SVM can avoid trial-and-error efforts required in conventional methods. A variety of experiments have shown that our method is effective in near-duplicate document detection.
Several difficulties are encountered in the study of near-duplicate detection. The real data sets for training/testing are hard to get. So far there have hardly been such benchmark data sets that can be openly accessed for research. For the study being practical, a method should be able to find near-duplicates from a huge reservoir, ideally the set of documents all over the world. We are working on the detection against the documents spread in the Internet, with the help of the cloud computing technology (Dean & Ghemawat, 2008). Our current work is focused on English texts. Expanding it to cover other languages is a necessity for it being useful.

Acknowledgment

This work was supported by the National Science Council under the Grants NSC-99-2221-E-110-064-MY3 and NSC-100-2622-E-110-009-CC3, and by “ Aim for the Top University Plan” of the National Sun Yat-Sen University and Ministry of Education.

References


Han, L., Finin, T., McNamee, P., Joshi, A., & Yesha, Y. (2012). Improving word similarity by augmenting PMI with estimates of word polysemy. IEEE Transactions on Knowledge and Data Engineering. 10.1109/TKDE.2012.3.


