Computing and Informatics, Vol. 32, 2013, 877–896

WORD COMBINATION KERNEL FOR TEXT CLASSIFICATION WITH SUPPORT VECTOR MACHINES

Lujiang Zhang, Xiaohui Hu

School of Automation Science and Electrical Engineering
Beijing University of Aeronautics & Astronautics
Beijing 100191, China
e-mail: zhanglujiang.dr@gmail.com

Communicated by Jacek Kitowski

Abstract. In this paper we propose a novel kernel for text categorization. This kernel is an inner product defined in the feature space generated by all word combinations of specified length. A word combination is a collection of unique words co-occurring in the same sentence. The word combination of length k is weighted by the k^{th} root of the product of the inverse document frequencies (IDF) of its words. By discarding word order, the word combination features are more compatible with the flexibility of natural language and the feature dimensions of documents can be reduced significantly to improve the sparseness of feature representations. By restricting the words to the same sentence and considering multi-word combinations, the word combination features can capture similarity at a more specific level than single words. A computationally simple and efficient algorithm was proposed to calculate this kernel. We conducted a series of experiments on the Reuters-21578 and 20 Newsgroups datasets. This kernel achieves better performance than the *word kernel* and *word-sequence kernel*. We also evaluated the computing efficiency of this kernel and observed the impact of the word combination length on performance.

Keywords: Machine learning, kernel methods, support vector machines, text classification, word-combination kernel

Mathematics Subject Classification 2010: 62H30, 46E22, 68T05, 68T50

1 INTRODUCTION

The Support Vector Machine (SVM) is a state of the art machine learning technique that has achieved great success in many domains. It is also very promising for text categorization [1, 2, 3] and web page categorization [4]. The effects of Support Vector Machines largely depend on the choice of kernels. The first and most commonly used kernel for text classification is the *word kernel* [1], which is a conventional kernel (linear, polynomial, RBF) combined with the *bag-of-words* model [5]. The *bag-of-words* model assumes that the words in a document are independent of each other, and their relative positions have no effect on text classification. So only the word frequencies with additional weighting and normalization are used to represent documents in word space, while the information regarding word positions is totally discarded.

Lodhi et al. [6] proposed the *string kernel*, the first significant departure from the *bag-of-words* model. In *string kernel*, features are all possible ordered subsequences of characters occurring in documents. The similarity between documents is assessed by the number of matching subsequences shared by two documents. Cancedda et al. [7] proposed the *word-sequence kernel* that extends the *string kernel* to process documents as word sequences. This approach greatly reduces the average length of symbols per document, which yields a significant improvement in computing efficiency. Moreover, matching word sequences allows working with more linguistically meaningful symbols.

There are still some issues with the *word-sequence kernel*. This kernel has very high dimensional and sparse feature space that hinders the effective training of kernel machines. Besides, natural language is flexible and considering word order is not helpful to conventional text classification tasks. Cancedda et al. [7] have proved by experiments that taking word order into account has very little effect on performance. Moschitti and Basili [8] found that phrases (including n-grams, sequences of words, noun phrases such as named entities and other complex nominals) are not adequate to improve classification accuracy, while elementary textual representations based on words are very effective.

In this paper we propose a novel kernel, called *word-combination kernel*. In this kernel, we use word combinations, rather than single words or word sequences, as features. A word combination is a collection of unique words without order relations co-occurring in the same sentence. The feature space of this kernel is generated by all word combinations of specified length and this kernel is an inner product defined in this space. By discarding word order, the word combination features are more compatible with the flexibility of natural language and the feature dimensions of documents can be reduced significantly to improve the sparseness of feature representations. By restricting the words of a word combination to the same sentence and considering multi-word combinations, the word combination features can capture similarity at a more specific level than single words and carry some information regarding the relative positions of words.

The rest of this paper is organized as follows. In Section 2 we briefly introduce the related work. In Section 3 we give a detailed description of the *word-combination kernel*. Section 4 presents the experimental results and evaluation. Finally, we conclude this paper in Section 5. A preliminary version of this work has been presented in [9].

2 RELATED WORK

2.1 Kernel Methods for Text Classification

A number of machine learning techniques have been applied to text categorization. A comprehensive survey about the machine learning techniques for text categorization can be found in [10]. In this paper, we focused on kernel methods for text classification. The effects of Support Vector Machines depend mainly on the choice of kernels. Support Vector Machines are very universal learners that can be used in conjunction with any kernel. The general kernels such as linear, polynomial and Gaussian RBF kernels have been used for text classification [1, 11]. Cristianini et al. [12] proposed the *Latent Semantic Kernels* based on latent semantic indexing. Though having obtained good performance, these conventional kernels are based on the *bag-of-words* model and inherit its intrinsic drawbacks. Some researchers inject lexical dependencies [13] or semantic relations [14] into the vector representations of documents as an extension to the standard *bag-of-words* model.

Lodhi et al. [6] proposed the *string kernel*, the first significant departure from the *bag-of-words* model. Cancedda et al. [7] proposed the *word-sequence kernel* which extends the *string kernel* to process documents as word sequences. Then the *factored sequence kernel* [15] was proposed to the case where the symbols that define the sequences have multiple representations. The *word-sequence kernel* proves to be more effective than *string kernel*, especially in computing efficiency and when using the standard linguistic preprocessing techniques. To resolve the poor computational efficiency problem, the *suffix-tree-based* and *suffix-array-based* string kernels [16, 17] are proposed to make the *string kernel* computationally feasible. Suzuki and Isozaki [18] embedded a statistical feature selection method into the *word-sequence kernel* to select significant features automatically.

Some researchers proposed syntactic and semantic kernels for text classification [19, 20, 21]; but only when the text categorization tasks are linguistically complex, such as classification in Question Answering (QA), syntax and semantics may play a relevant role [20, 22, 23]. Apparently promising syntactic and semantic structures have been shown inadequate for conventional text categorization tasks [8, 24, 25].

2.2 Word Kernel and Word-Sequence Kernel

In this section, we briefly introduce the *word kernel* and *word-sequence kernel*. The *word kernel* is a conventional kernel (linear, polynomial, or Gaussian RBF kernel) combined with the *bag-of-words* model. For this kernel, each document is represented as a vector in word space using the TF×IDF weighting system [26] or its variant [11], where TF represents the term frequency and IDF represents the inverse document frequency. Specifically, given a document collection containing n distinct words, each document d is represented as a n-dimensional vector $\mathbf{f} = (tf_1 * idf_1, tf_2 * idf_2, \ldots, tf_n * idf_n)$. Here tf_i is the term frequency defined as the number of occurrences of word w_i in document d, and idf_i is the inverse document frequency defined as $\log \frac{N}{df_i}$, where df_i is the number of documents containing word w_i and N is the total number of documents in the collection. Then in this vector space we use a linear, polynomial, or Gaussian RBF kernel to compute the kernel values between documents.

We now introduce the *word-sequence kernel*. Let Ω be a finite vocabulary. $s = s_1 s_2 \ldots s_{|s|}$ is a word sequence over Ω ($s_i \in \Omega$). Let $\mathbf{i} = [i_1, i_2, \ldots, i_n]$ be a subset of the indices in s ($1 \leq i_1 \leq i_2 \leq \ldots i_n \leq |s|$). We indicate the subsequence $s_{i_1} s_{i_2} \ldots s_{i_n}$ as $s[\mathbf{i}] \in \Omega^n$ (Ω^n is the set of all subsequences of length n). Note that $s[\mathbf{i}]$ does not necessarily form a contiguous subsequence of s. We denote by $l(\mathbf{i})$ the length spanned by $s[\mathbf{i}]$ in s, that is $l(\mathbf{i}) = i_n - i_1 + 1$. The kernel for two sequences s and t is defined as:

$$K_n(s,t) = \sum_{u \in \Omega^n} \sum_{\mathbf{i}:s[\mathbf{i}]=u} \sum_{\mathbf{j}:t[\mathbf{j}]=u} \lambda^{l(\mathbf{i})+l(\mathbf{j})}, \tag{1}$$

where $\lambda \in [0,1]$ is a decay factor used to penalize non-contiguous subsequences. $K_n(s,t)$ is a valid kernel as it amounts to computing an inner product in the feature space $F = R^{\Omega^n}$, with the coordinate $\phi_u(s)$ for each $u \in \Omega^n$:

$$\phi_u(s) = \sum_{\mathbf{i}:s[\mathbf{i}]=u} \lambda^{l(\mathbf{i})}. \tag{2}$$

The basic idea is that we match all possible subsequences of n words, and non contiguous occurrences are penalized according to the number of gaps they contain. A direct calculation of this kernel becomes impractical even for small values of n. However, we can compute this kernel using a recursive formulation proposed by Lodhi et al. [6], which leads to an efficient dynamic programming technique.

3 WORD-COMBINATION KERNEL

In this section we describe the details of the *word-combination kernel*. The key of this kernel is the use of word combination features. A word combination is a collection of different words co-occurring in the same sentence. The feature space of this kernel is generated by all word combinations of specified length. Note that there are no order relations between the words of a word combination. The basic idea is to measure the similarity between two documents by the number of word combinations they share in common. The more common word combinations they share, the more similar they are. A word combination of length k is weighted by the k^{th} root of the product of the inverse document frequencies (IDF) [26] of its words. The feature value corresponding to a specific word combination is the sum of weights over all occurrences of the word combination.

The flexibility of natural language makes it possible to express the same or similar information by means of various sentence structures or word orders (e.g., "give Mary a pie" and "give a pie to Mary", "book seller" and "seller of books", "He wrote the letter." and "The letter was written by him."), so word combinations are more compatible with the flexibility of natural language than word sequences. Besides, for the conventional text categorization tasks that explore primarily topic and category information, taking complex linguistic features, including syntactic and semantic structures or phrases (n-grams, sequences of words, noun phrases) into account does not necessarily improve classification accuracy [8, 24, 25]. Documents usually have high dimensional and sparse representations in feature space. By discarding word order, the feature dimensions of documents can be reduced significantly to improve the sparseness of feature representations. By restricting its words to the same sentence, a multi-word combination can carry some information regarding the relative position of its words, and the multi-word combinations can capture similarity at a more specific level than single words.

3.1 Definition

Let Ω be a finite vocabulary, which is a set of words. A document d is composed of n successive sentences: $d = \{s_1, s_2, \ldots, s_n\}$. A sentence $s_i = \{s_{i_1}, s_{i_2}, \ldots, s_{i_{|s_i|}}\}$ ($s_{i_j} \in \Omega$) is regarded as a collection of words without order relations. A word combination $u = \{u_1, u_2, \ldots, u_{|u|}\}$ ($u_i \neq u_j$ for any $i \neq j$) is a collection of unique words co-occurring in the same sentence. We say u is a word combination of document d if and only if there exists at least a sentence s_i in document d satisfying $u \subseteq s_i$, and we use the shorthand notation $u \subseteq d[s_i]$ to denote it. We denote by Ω^n the set of all word combinations of length n.

We now define the feature space $F_n = R^{\Omega^n}$. The mapping ϕ from a document $d = \{s_1, s_2, \ldots, s_n\}$ to the feature space F_n is defined as $\phi : d \to (\phi_u(d))_{u \in \Omega^n}$, where $\phi_u(d)$ is the coordinate of the word combination u in the feature space F_n. The definition of $\phi_u(d)$ is given as follows:

$$\phi_u(d) = \sum_{s_i : u \subseteq d[s_i]} \lambda_u^{(d)}, \quad (u \in \Omega^n) \tag{3}$$

$$\lambda_u^{(d)} = \left(\prod_{i=1}^{|u|} \lambda_{u_i}^{(d)} \right)^{\frac{1}{|u|}}, \quad (u_j \in u). \tag{4}$$

We denote by $\lambda_u^{(d)}$ the weight of the word combination u in document d, and by $\lambda_{u_j}^{(d)}$ the inverse document frequency of the word u_j in document d.

In the feature space $F_n = R^{\Omega^n}$, the *word-combination kernel* is defined as follows:

$$\begin{aligned} K_n(d_1, d_2) &= \langle (\phi_u(d_1))_{u\in\Omega^n}, (\phi_u(d_2))_{u\in\Omega^n} \rangle \\ &= \sum_{u\in\Omega^n} \phi_u(d_1)\phi_u(d_2) \\ &= \sum_{u\in\Omega^n} \left(\sum_{s_i:u\subseteq d_1[s_i]} \lambda_u^{(d_1)} \right) \left(\sum_{s_j:u\subseteq d_2[s_j]} \lambda_u^{(d_2)} \right) \\ &= \sum_{u\in\Omega^n} \sum_{s_i:u\subseteq d_1[s_i]} \sum_{s_j:u\subseteq d_2[s_j]} \left(\prod_{i=1}^{|u|} \lambda_{u_i}^{(d_1)} \right)^{\frac{1}{|u|}} \left(\prod_{i=1}^{|u|} \lambda_{u_i}^{(d_2)} \right)^{\frac{1}{|u|}}. \end{aligned} \tag{5}$$

$K_n(d_1, d_2)$ satisfies the definition of positive definite kernel [27] because it is an inner product defined in the feature space $F_n = R^{\Omega^n}$. After the kernel has been computed we need to normalize it to remove any bias introduced by the document length. We use the following l_2 normalization to normalize the kernel:

$$\begin{aligned} \hat{K}_n(d_1, d_2) &= \langle \hat{\phi}(d_1), \hat{\phi}(d_2) \rangle \\ &= \left\langle \frac{\phi(d_1)}{\|\phi(d_1)\|_2}, \frac{\phi(d_2)}{\|\phi(d_2)\|_2} \right\rangle \\ &= \frac{K_n(d_1, d_2)}{\sqrt{(K_n(d_1, d_1)K_n(d_2, d_2)}}. \end{aligned} \tag{6}$$

In Equation (4), we use the k^{th} root operator to make the weights of word combinations of different lengths have the same order of magnitude. This is helpful to combine the *word-combination kernels* with different feature lengths. For the word weighting, we do not consider the term frequency (TF) because the summation operation in Equation (3) has taken into account the word combination frequencies and the word frequencies can be embodied in the word combination frequencies. If we use the TF $\times$ IDF weighting instead of the inverse document frequency (IDF) to weight a word, the weighting system will contribute some redundant information about word frequencies that can negatively bias the computed similarity.

Documents usually have sparse representations in feature space. Reducing the dimensionality of feature space is helpful to alleviate the sparseness of feature representations. For a vocabulary Ω and a specified feature length n, the dimensionality of the *word-combination kernel* is $\binom{|\Omega|}{n}$, while that of the *word-sequence kernel* is $|\Omega|^n$. The former is much lower than the latter because $\frac{\binom{|\Omega|}{n}}{|\Omega|^n} = \frac{1}{n!}\frac{|\Omega|\ldots(|\Omega|-n+1)}{|\Omega|^n} \leq \frac{1}{n!}$.

3.2 Combining Kernels of Different Lengths

In general, word combinations of any length can make a contribution to similarity between documents. So it is necessary to combine the kernels with different feature lengths. We use a linear combination formula to combine the *word-combination kernels* with feature lengths from 1 to a fixed n:

$$K'_n(d_1, d_2) = \sum_{i=1}^{n} w_i K_i(d_1, d_2). \tag{7}$$

Kernels of different lengths should be normalized independently before being combined. We can obtain the optimized weighting parameters w_i ($i = 1, 2, \dots, n$) by means of *multiple kernel learning* [28] or *cross-validation*. However, in practice we can simply set $w_i = i$. That is, the importance of a *word combination kernel* is proportional to its feature length. In Equation (4), the k^{th} root operator makes the weights of word combinations of different lengths have the same order of magnitude, which is helpful for linear combination of *word-combination kernels* with different feature lengths.

3.3 The Sentence-Intersections Between Documents

A *sentence-intersection* is the intersection, that is, the collection of common words between two sentences which belong to two documents, respectively. Formally speaking, the *sentence-intersection* between the sentence s_1 in document d_1 and the sentence s_2 in document d_2 is given by $s_1 \cap s_2$. The *sentence-intersections* are used to generate the common word combinations of specified length between documents using the combination generation algorithm. For example, from the *sentence-intersection*: $\{\text{newspaper}, \text{report}, \text{football}\}$, we can generate the two-word combinations: $\{\text{newspaper}, \text{report}\}$, $\{\text{report}, \text{football}\}$, $\{\text{newspaper}, \text{football}\}$.

We statistically analyzed the distribution of the *sentence-intersections* of different lengths between documents in the Reuters-21578 and 20 Newsgroups datasets (The descriptions of the two datasets are presented in Section 4.1). Table 1 displays the distribution ratio of *sentence-intersections* of different lengths. From Table 1, we can see that the distribution of the long *sentence-intersections* whose lengths are greater than 3 is very sparse.

	$n = 1$	$n = 2$	$n = 3$	$n > 3$
Reuters-21578	82.09 %	13.74 %	2.62 %	1.55 %
20 Newsgroups	93.41 %	6.08 %	0.38 %	0.13 %

Table 1. The distribution ratio of the *sentence-intersections* of different lengths between documents in the Reuters-21578 and 20 Newsgroups datasets. n represents the length of *sentence-intersection*.

3.4 Algorithm

In this section we give the details about the algorithm of *word-combination kernel*. Before the calculation of this kernel, we need to preprocess the documents (see

Section 4.2). After preprocessing each document is converted into a list of sentences. The algorithm computes the *word-combination kernels* with feature lengths from 1 to a specified length m. Then these kernels are normalized using Equation (6) and combined using Equation (7).

The crux of calculating this kernel is to find out all of the *sentence-intersections* between documents. The *sentence-intersections* are used to generate the common word combinations between documents. The major cost of calculating this kernel is consumed in searching the *sentence-intersections*. To accelerate the searching process, we designed a hash table structure (see Figure 1) for documents. The key of this hash table is a word and the corresponding element is the list of sentences that contain this word. The *BKDR Hash Function* [29] is used to compute the hash code of a word. We use the *double hashing* to deal with the address collision, and set the *load factor* (the ratio of the actual number of keys in the hash table to the size of the hash table) to 0.75. By help of this structure, the sentences in a document containing a specific word can be found in $O(1)$ time. The algorithm includes two steps. The first step creates a hash table for each document and the second step computes the kernel values between documents of feature lengths from 1 to a fixed m. The algorithm is as follows:

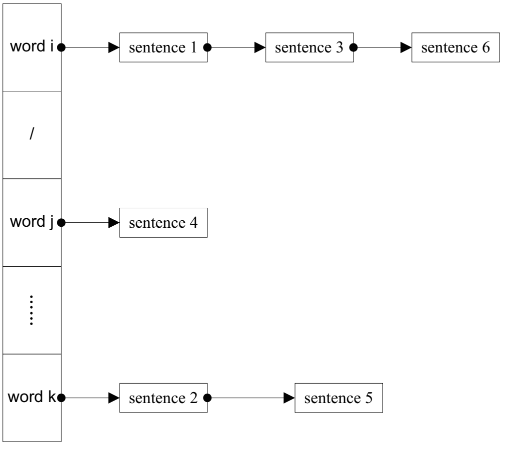

Fig. 1. The hash table structure for documents

Step 1: Creating the hash table for a document.
Input:
d: The input document;
Output:
$HT^{(d)}$: The hash table for the input document d;
Procedure:
Initialize the hash table $HT^{(d)}$ and the hashing function $h(x)$;

for each word w_k in document d
    if w_k exists in $HT^{(d)}$
        List $L \leftarrow HT^{(d)}[h(w_k)]$;
        Add the sentence index of w_k into L;
        $HT^{(d)}[h(w_k)] \leftarrow L$;
    else
        List $L \leftarrow \emptyset$;
        Add the sentence index of w_k into L;
        $HT^{(d)}[h(w_k)] \leftarrow L$;
    end if
end for

Step 2: Computing the kernel values between documents of feature lengths from 1 to m.
Input:
$HT^{(d_1)}$: The hash table for the input document d_1;
$HT^{(d_2)}$: The hash table for the input document d_2;
m: The maximal word combination length;
Output:
$K[1 \ldots m]$: The array of kernel values of feature lengths from 1 to m;
Procedure:
$K[1 \ldots m] \leftarrow \{0, \ldots, 0\}$;
$n_1 \leftarrow$ The number of sentences in document d_1;
$n_2 \leftarrow$ The number of sentences in document d_2;
Sentence-intersection array $S[1 \ldots n_1, 1 \ldots n_2] \leftarrow \emptyset$;
for each key w_k in $HT^{(d_1)}$
    if w_k exists in $HT^{(d_2)}$
        List $L1 \leftarrow HT^{(d_1)}[h(w_k)]$;
        List $L2 \leftarrow HT^{(d_2)}[h(w_k)]$;
        for each sentence index i in $L1$
            for each sentence index j in $L2$
                Add w_k to $S[i, j]$;
            end for
        end for
    end if
end for
for each sentence-intersection s_{ij} in $S[1 \ldots n_1, 1 \ldots n_2]$
    if $s_{ij} \neq \emptyset$
        for $l \leftarrow 1$ to m
            $U^{(l)} \leftarrow$\{The word combinations of length l generated from the sentence-intersection s_{ij} using
                    the combination generation algorithm\};
            for each word combination $u_k^{(l)}$ in $U^{(l)}$
                $K[l] \leftarrow K[l] + \lambda^{(d_1)}_{u_k^{(l)}} * \lambda^{(d_2)}_{u_k^{(l)}}$;

end for
end for
end if
end for

In this algorithm, we denote by $u_k^{(l)}$ a word combination of length l generated from a *sentence-intersection* s_{ij} and denote by $\lambda_{u_k^{(l)}}^{(d)}$ the weight of $u_k^{(l)}$ in document d. The $\lambda_{u_k^{(l)}}^{(d)}$ is computed using Equation (4).

Computational complexity of this kernel is $O(2|d_1| + 2|d_2| + n_1 n_2 \sum_{i=1}^{m} i\binom{M}{i})$, where $|d_1|$ and $|d_2|$ are the lengths of documents d_1 and d_2, respectively, n_1 and n_2 are the number of sentences in d_1 and d_2, respectively, m is the specified maximal word combination length, and M is the maximal length of *sentence-intersections* between d_1 and d_2. This computational complexity consists of two parts, the first part $2|d_1| + 2|d_2|$ corresponds to the cost of creating two hash tables for documents d_1 and d_2 and searching the *sentence-intersections* between d_1 and d_2, while the second part $n_1 n_2 \sum_{i=1}^{m} i\binom{M}{i}$ corresponds to the cost of generating the common word combinations between d_1 and d_2 from the *sentence-intersections* and computing the kernel values of feature lengths from 1 to m. The $O(2|d_1| + 2|d_2| + n_1 n_2 \sum_{i=1}^{m} i\binom{M}{i})$ is actually an upper bound of the real computational complexity. Given that more than 95% of the *sentence-intersections* between documents have a length less than or equal to 3 (see Table 1), we can let $M = 3$. Thus the computational complexity is $O(2|d_1| + 2|d_2| + n_1 n_2 \sum_{i=1}^{m} i\binom{3}{i}) \leq O(2|d_1| + 2|d_2| + n_1 n_2 \sum_{i=1}^{3} i\binom{3}{i}) = O(2|d_1| + 2|d_2| + 12 n_1 n_2) = O(|d_1| + |d_2| + 6 n_1 n_2)$. Because n_1 and n_2 is much smaller than $|d_1|$ and $|d_2|$, this complexity is close to the linear complexity with respect to the document length.

4 EXPERIMENTS

In this section we describe the experiments. The objectives of our experiments include:

- observe the impact of the word combination length on performance of the *word-combination kernel*;
- compare the classification performance of this kernel to those of the *word kernel* and *word-sequence kernel*;
- compare the computing efficiency of this kernel to those of the *word kernel* and *word-sequence kernel*.

We use Equation (7) with the parameters $w_i = i$ $(i = 1, 2, \ldots, n)$ to compute the *word-combination kernel*. For the SVM classifier, we select the libSVM [30] of C-SVC type to conduct our experiments. The 3-fold *cross-validation* is used to optimize the value of C. For the *word kernel*, we use the linear version rather than the Gaussian version. As Yang and Liu [3] have pointed out, we also found that the linear kernel can obtain a slightly better result than the Gaussian kernel for text classification with the *bag-of-words* model.

4.1 Dataset

We apply the Reuters-21578 and 20 Newsgroups datasets to our experiments. The Reuters-21578 dataset was compiled by David Lewis in 1987, and is available at `http://www.daviddlewis.com/resources/testcollections/reuters21578/`. We use the "ModeApte" split of the Reuters-21578 dataset. It comprises 9 603 training and 3 299 test documents that had been classified into 118 categories. We select the eight most frequent categories: "earn", "acq", "money", "grain", "crude", "trade", "interest" and "ship" for experiments. There are some overlapped documents across categories in the Reuters-21578 dataset. We removed the overlapped test documents and retained the overlapped training documents. The eight categories are summarized in Table 2. We use all of the training and test documents of each category to evaluate the performance of a kernel.

The 20 Newsgroups dataset was originally collected by Ken Lang in the mid-90's, and is available at `http://people.csail.mit.edu/jrennie/20Newsgroups/`. It is a collection of approximately 20 000 newsgroup documents and is partitioned evenly across twenty different newsgroups that correspond to twenty categories, respectively (see Table 3). For the 20 Newsgroups dataset, we evaluate the performance of a kernel by averaging the results over the 10 runs of the algorithm, and for each run we randomly selected 300 training documents and 150 test documents from each category to form a target dataset.

Category	# training samples	# test samples
earn	2 877	1 083
acq	1 650	709
money	538	130
grain	433	133
crude	389	142
trade	369	103
interest	347	87
ship	197	43

Table 2. Summarization of the eight categories of the Reuters-21578 dataset after removing the overlapped test documents

4.2 Data Preprocessing

The data preprocessing includes sentence boundary detection, stop word removal, inflectional stemming and computing the weighting of words. The sentence boundary detection is only used for the *word-combination kernel* and after this processing each document is converted into a list of sentences. For the 20 Newsgroups dataset,

Category	
alt.atheism	rec.sport.hockey
comp.graphics	sci.crypt
comp.os.ms-windows.misc	sci.electronics
comp.sys.ibm.pc.hardware	sci.med
comp.sys.mac.hardware	sci.space
comp.windows.x	soc.religion.christian
misc.forsale	talk.politics.guns
rec.autos	talk.politics.mideast
rec.motorcycles	talk.politics.misc
rec.sport.baseball	talk.religion.misc

Table 3. The twenty categories of the 20 Newsgroups dataset

we need to remove the headers of each document before preprocessing. After preprocessing, the Reuters-21578 dataset contains 26384 unique words, with the average of 66 words and 6.1 sentences per document, while the 20 Newsgroups dataset contains the average of 72 342 unique words, with the average of 118 words and 16.3 sentences per document.

To find the common word combinations between documents, we need to split a document into sentences. This processing has been well implemented in the java.text package of the Java^{TM} Platform Standard Edition 6. Stop word removal filters out the words that are generally regarded as 'functional words' and do not carry meaning. We removed the words occurring in a stop word list built for the SMART information retrieval system (`ftp://ftp.cs.cornell.edu/pub/smart/english.stop`). Inflectional stemming is the process of transforming a word into its base, non-inflected form. It is not an easy linguistic processing and may introduce additional errors, so we only perform the singular/plural regularization using regular expression based approach. For the *word-combination kernel*, we use the inverse document frequency (IDF) to weight the words in a document, and use the l_2 normalization to normalize the inverse document frequencies of a document. For the *word kernel* and *word-sequence kernel*, we strictly follow the weighting system proposed by their researchers [1, 7].

4.3 Performance Evaluation

We use the F_1 score to measure the classification performance. It is given by $F_1 = 2pr/(p+r)$, where p is precision and r is recall. We calculate the F_1 score for each category and also provide the micro-averaged and macro-averaged F_1 scores over all categories. The macro-averaging averages the results obtained on each category, while the micro-averaging averages over individual decisions on each document for each category.

We further compare the classification performance of different kernels using the significance tests: macro sign test (S-test) and macro t-test (T-test) [3]. The S-test and T-test are both designed for comparing two systems A and B using the paired F_1 scores for individual categories. The S-test has the following notations:

- N is the number of unique categories;
- $a_i \in [0,1]$ is the F_1 score of system A on the i^{th} category $(i = 1, 2, \ldots, N)$;
- $b_i \in [0,1]$ is the F_1 score of system B on the i^{th} category $(i = 1, 2, \ldots, N)$;
- n is the number of times that a_i and b_i differ;
- k is the number of that a_i is larger than b_i.

The null hypothesis is that k has a binomial distribution of $\mathrm{Bin}(n, p)$ where $p = 0.5$. The alternative hypothesis is that k has a binomial distribution of $\mathrm{Bin}(n, p)$ where $p > 0.5$, meaning that system A is better than system B. If $k \geq 0.5n$, the P-value (1-side) is computed using the binomial distribution under the null hypothesis:

$$P(Z \geq k) = \sum_{i=k}^{n} \binom{n}{i} \times 0.5^n. \tag{8}$$

Symmetrically, if $k < 0.5n$, the P-value for the other extreme is computed using the formula

$$P(Z \leq k) = \sum_{i=0}^{k} \binom{n}{i} \times 0.5^n. \tag{9}$$

The P-value indicates the significance level of the observed evidence against the null hypothesis. To define the T-test, we use the same notations as defined for S-test, and the following additional items:

- $\delta_i = a_i - b_i$ is the difference of a_i from b_i;
- $\bar{\delta}$ is the simple average of the δ_i values for $i = 1, 2, \ldots, n$.

The null hypothesis is $\bar{\delta} = 0$. The alternative hypothesis is $\bar{\delta} > 0$. We denote by $s.e.(\bar{\delta})$ the standard error of the $\bar{\delta}$. The P-value is computed using the t-distribution with the degree of freedom $n - 1$:

$$T \geq \frac{\bar{\delta}}{s.e.(\bar{\delta})}. \tag{10}$$

S-test may be more robust for reducing the influence of outliers, but is not sufficiently sensitive in performance comparison because it ignores the absolute differences between F_1 values. The T-test is sensitive to the absolute values, but could be overly sensitive when F_1 scores are unstable. So using the two tests jointly instead of using one test alone would be a good compromise.

4.4 Experimental Results

In this subsection, we present the experimental results. We evaluate the performance of the *word-combination kernel* on the Reuters-21578 and 20 Newsgroups datasets.

In Section 4.4.1 we observe the impact of the word combination length on performance of this kernel. In Section 4.4.2 we compare the performance of this kernel to those of the *word kernel* and *word-sequence kernel*. In Section 4.4.3 we compare the computing efficiency of this kernel to those of the *word kernel* and *word-sequence kernel*.

4.4.1 Impact of the Word Combination Length on Performance

To assess the impact of the word combination length on performance, we provided the micro-averaged and macro-averaged F_1 scores varying with the word combination length from 1 to 4 in Table 4. It can be seen from Table 4 that when length $n = 2$ the *word-combination kernel* obtains the best performance. Besides, when length $n \geq 2$, the micro-averaged and macro-averaged F_1 scores consistently decrease with the increase of word combination length. We think that this is because of the sparse distribution of the long common word combinations between documents whose lengths are greater than 2, which can be deduced from Table 1. When the long word combinations are taken into account the precision increases, but the loss in recall more than offsets the increase in precision, so the F_1 scores decrease. It is notable that the *word-combination kernel* with length $n = 2$ achieves better performance than the *word-combination kernel* with length $n = 1$, which indicates the effectiveness of multi-word combinations compared to single words.

		$n = 1$	$n = 2$	$n = 3$	$n = 4$
Reuters-21578	micro-averaged F_1	0.9452	0.9506	0.9416	0.9214
	macro-averaged F_1	0.9017	0.9071	0.8895	0.8484
20 Newsgroups	micro-averaged F_1	0.7955 ± 0.010	0.8241 ± 0.006	0.8213 ± 0.006	0.8076 ± 0.004
	macro-averaged F_1	0.8005 ± 0.010	0.8280 ± 0.006	0.8277 ± 0.006	0.8211 ± 0.004

Table 4. The impact of the word combination length on performance of the *word-combination kernels* with feature lengths from 1 to 4. The results for the 20 Newsgroups dataset are averaged over 10 runs of the algorithm.

4.4.2 Comparison of Performance

We compare the performance of the *word-combination kernel* to that of the *word kernel* and *word-sequence kernel* on the Reuters-21578 and 20 Newsgroups datasets. We present the F_1 score for each category and the micro-averaged and macro-averaged F_1 scores over all categories. Tables 5 and 6 display the experimental results for the Reuters-21578 and 20 Newsgroups datasets, respectively. We use the boldface to mark the best result for each category. The results show that the *word-combination kernel* can achieve good performance on the two datasets. For the Reuters-21578 dataset this kernel obtains seven best results among the eight categories, while for the 20 Newsgroups dataset this kernel obtains eighteen best results among the twenty categories. The micro-averaged and macro-averaged F_1 scores of this kernel are also better than those of the *word kernel* and *word-sequence kernel*.

To further verify the performance of *word-combination kernel*, we apply two significance tests: macro sign test (S-test) and macro t-test (T-test) to the experimental results. Table 7 shows the test results. "$\gg$" means P-value ≤ 0.01, indicating a strong evidence that the left-hand kernel is better than the right-hand one; "$>$" means $0.01 <$ P-value ≤ 0.05, indicating a weak evidence that the left-hand kernel is better than the right-hand one; "$\sim$" means P-value > 0.05, indicating that it has no significant difference between the two side. The results in Table 7 provide clear and convincing evidence that this kernel performs better than the *word kernel* and *word-sequence kernel* on the Reuters-21578 and 20 Newsgroups datasets. Combining the results in Tables 5, 6 and 7, we think that the *word-combination kernel* can be an effective approach for text categorization tasks.

	WCK	WK	WSK
earn	**0.9823**	0.9693	0.9672
acq	**0.9521**	0.9394	0.9278
money	**0.8595**	0.8561	0.8167
grain	**0.9585**	0.9470	0.9549
crude	**0.9024**	0.8904	0.8966
trade	**0.9108**	0.8919	0.8981
interest	**0.8539**	0.7662	0.6904
ship	0.8205	0.8101	**0.8312**
micro-average	**0.9504**	0.9353	0.9276
macro-average	**0.9050**	0.8838	0.8729

Table 5. The F_1 scores of the *word-combination kernel* (WCK), *word kernel* (WK) and *word-sequence kernel* (WSK) on the Reuters-21578 dataset. The *word-combination kernel* and *word-sequence kernel* are both with feature length $n = 2$.

4.4.3 Comparison of Computing Efficiency

We compare the computing efficiency of the *word-combination kernel* to that of the *word kernel* and *word-sequence kernel* on a notebook computer with 2.40 GHz Intel Core$^{\text{TM}}$ Duo CPU. Table 8 shows the preprocessing, training and test time of each kernel on the Reuters-21578 and 20 Newsgroups datasets. The time is measured in seconds. We can see from Table 8 that the total running time of the *word-combination kernel* is more than but roughly comparable to that of the *word kernel*, yet the *word-sequence kernel* takes an excessively far more running time even for the eight categories of the Reuters-21578 dataset. Besides, the preprocessing time only accounts for a small fraction of the total running time. The computational complexities of the three kernels are displayed in Table 9. Among the three kernels, the *word kernel* (we use the linear version) has the lowest computational complexity, while the *word-sequence kernel* has the highest computational complexity. Combin-

	WCK	WK	WSK
alt.atheism	**0.7897±0.022**	0.7321 ± 0.027	0.7737 ± 0.021
comp.graphics	0.7231±0.031	0.7256 ± 0.028	**0.7280 ± 0.048**
comp.os.ms-windows.misc	**0.7504 ± 0.022**	0.7047 ± 0.028	0.7442 ± 0.032
comp.sys.ibm.pc.hardware	**0.7101 ± 0.027**	0.6584 ± 0.034	0.7016 ± 0.022
comp.sys.mac.hardware	**0.7926 ± 0.037**	0.7546 ± 0.031	0.7892 ± 0.043
comp.windows.x	**0.8293 ± 0.022**	0.7984 ± 0.022	0.8218 ± 0.025
misc.forsale	**0.7692 ± 0.025**	0.7151 ± 0.033	0.7588 ± 0.019
rec.autos	**0.8739 ± 0.018**	0.8495 ± 0.023	0.8573 ± 0.012
rec.motorcycles	**0.9323 ± 0.012**	0.9190 ± 0.015	0.9210 ± 0.019
rec.sport.baseball	**0.9400 ± 0.012**	0.9082 ± 0.014	0.9308 ± 0.013
rec.sport.hockey	**0.9508 ± 0.014**	0.9122 ± 0.028	0.9446 ± 0.012
sci.crypt	**0.8975 ± 0.020**	0.8689 ± 0.026	0.8935 ± 0.025
sci.electronics	0.7194±0.051	0.7128±0.022	**0.7232±0.034**
sci.med	**0.8941 ± 0.017**	0.8846 ± 0.017	0.8822 ± 0.018
sci.space	**0.9063 ± 0.022**	0.8950 ± 0.015	0.8954 ± 0.023
soc.religion.christian	**0.7983 ± 0.026**	0.7398 ± 0.022	0.7838 ± 0.028
talk.politics.guns	**0.8434 ± 0.022**	0.7996 ± 0.022	0.8167 ± 0.016
talk.politics.mideast	**0.9261 ± 0.015**	0.9032 ± 0.014	0.9190 ± 0.022
talk.politics.misc	**0.8141 ± 0.015**	0.7559 ± 0.027	0.7817 ± 0.016
talk.religion.misc	**0.6306 ± 0.040**	0.5270 ± 0.058	0.5625 ± 0.048
micro-average	**0.8241 ± 0.006**	0.7900 ± 0.010	0.8127 ± 0.008
macro-average	**0.8280 ± 0.006**	0.7951 ± 0.010	0.8164 ± 0.008

Table 6. The F_1 scores of the *word-combination kernel* (WCK), *word kernel* (WK) and *word-sequence kernel* (WSK) on the 20 Newsgroups dataset. The *word-combination kernel* and *word-sequence kernel* are both with feature length $n = 2$. The results are obtained by averaged over 10 runs of the algorithms.

ing the results in Tables 8 and 9, we can see that the running time of each kernel is consistent with its computational complexity. It is worth noting that the *word-sequence kernel* is extremely computationally demanding, though we use a dynamic programming formulation proposed by Lodhi et al. [6] to speed up the calculation.

5 CONCLUSIONS AND FUTURE WORK

In this paper we propose the *word-combination kernel* for text classification. We aim to provide a practical and easy-to-use text kernel. We give a detailed description of this kernel and empirically evaluate it on the Reuters-21578 and 20 Newsgroups datasets. The performance of this kernel is compared to those of the *word kernel* and *word-sequence kernel*.

We devised the word combination features for this kernel. A word combination is a collection of unique words co-occurring in the same sentence. This kernel is an inner product defined in the feature space generated by all word combinations of specified length. Compared to the word sequence features, the word combination

			S-test	T-test
Reuters-21578	WCK	WK	$\gg$	$>$
	WCK	WSK	$>$	$\sim$
	WK	WSK	$\sim$	$\sim$
20 Newsgroups	WCK	WK	$\gg$	$\gg$
	WCK	WSK	$\gg$	$\gg$
	WSK	WK	$\gg$	$\gg$

Table 7. The statistical significance test results using S-test and T-test between the *word-combination kernel* (WCK), *word kernel* (WK) and *word-sequence kernel* (WSK). "$\gg$" indicates a strong evidence that the left-hand kernel is better than the right-hand one; "$>$" indicates a weak evidence that the left-hand kernel is better than the right-hand one; "$\sim$" indicates we can't decide which side is better.

	(unit = second)	WCK	WK	WSK
Reuters-21578	Preprocessing time	29	25	25
	Training time	418	341	25 177
	Test time	133	92	11 256
	Total	551	458	36 458
20 Newsgroups	Preprocessing time	62	53	53
	Training time	558	349	36 153
	Test time	367	156	19 367
	Total	987	538	55 573

Table 8. Comparison of computing efficiency between the *word-combination kernel* (WCK), *word kernel* (WK) and *word-sequence kernel* (WSK) on the Reuters-21578 and 20 Newsgroups datasets. The running time is measured in seconds. The *word-combination kernel* and *word-sequence kernel* are both with feature length $n = 2$. The time for the 20 Newsgroups dataset is obtained by averaged over 10 runs of the algorithms.

features are more compatible with the flexibility of natural language and the feature dimensions of documents can be reduced significantly. In addition, the word combination features can capture similarity at a more specific level than single words. A computationally simple and efficient algorithm is proposed to calculate this kernel. We use a linear combination formulation to combine the *word-combination kernels* with different feature lengths, and observed the impact of the word combination length on performance. When the word combination length $n = 2$, this kernel ob-

	WCK	WK	WSK
Computational complexity	$O(\lvert d_1\rvert + \lvert d_2\rvert + 6n_1n_2)$	$O(\lvert d_1\rvert + \lvert d_2\rvert)$	$O(n\lvert d_1\rvert\lvert d_2\rvert)$

Table 9. Computational complexities of the *word-combination kernel* (WCK), *word kernel* (WK) and *word-sequence kernel* (WSK). For the *word-sequence kernel*, n is the feature length.

tains the best performance. Experimental results show that the *word-combination kernel* can achieve better performance than the *word kernel* and *word-sequence kernel* on the Reuters-21578 and 20 Newsgroups datasets.

The *word-combination kernel* can be used in conjunction with any kernel-based learning system. We will further research the use of this kernel to text clustering, ranking tasks, etc., and conduct more experiments on various kinds of text datasets. We will also research the feature selection method that can be embedded into this kernel to select the significant word combination features automatically.

REFERENCES

[1] Joachims, T.: Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In: Proceedings of the 10th European Conference on Machine Learning, Chemnitz, Germany, 1998, pp. 137–142.

[2] Dumais, S.—Platt, J.—Heckerman, D.—Sahami, M.: Inductive Learning Algorithms and Representations for Text Categorization. In: Proceedings of the 7th International Conference on Information and Knowledge Management (CIKM '98), Bethesda, Maryland, USA, 1998, pp. 148–155.

[3] Yang, Y.—Liu, X.: A Re-Examination of Text Categorization Methods. In: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Berkeley, California, USA 1999, pp. 42–49.

[4] Radovanović, M.—Ivanović, M.—Budimac, Z.: Text Categorization and Sorting of Web Search Results. Computing and Informatics, Vol. 28, 2009, No. 6, pp. 861–893.

[5] Salton, G.—Mcgill, M. J.: Introduction to Modern Information Retrieval. McGraw-Hill, New York, USA, 1983.

[6] Lodhi, H.—Saunders, C.—Shawe-Taylor, J.—Cristianini, N.—Watkins, C.: Text Classification Using String Kernels. Journal of Machine Learning Research, Vol. 2, 2002, pp. 419–444.

[7] Cancedda, N.—Gaussier, E.—Goutte, C.—Renders, J. M.: Word-Sequence Kernels. Journal of Machine Learning Research, Vol. 3, 2003, pp. 1059–1082.

[8] Moschitti, A.—Basili, R.: Complex Linguistic Features for Text Classification: A Comprehensive Study. In: Proceedings of the 26th European Conference on Information Retrieval (ECIR '04), Sunderland, United Kingdom, 2004, pp. 181–196.

[9] Zhang, L.—Hu, X.: A Novel Kernel for Text Categorization. In: Proceedings of 2012 IEEE International Conference on Computer Science and Automation Engineering, Zhangjiajie, China, 2012, Vol. 1, pp. 186–190.

[10] Sebastiani, F.: Machine Learning in Automated Text Categorization. ACM Computing Surveys, Vol. 34, 2002, No. 1, pp. 1–47.

[11] Leopold, E.—Kindermann, J.: Text Categorization with Support Vector Machines. How to Represent Texts in Input Space? Machine Learning, Vol. 46, 2002, No. 1-3, pp. 423–444.

[12] Cristianini, N.—Shawe-Taylor, J.—Lodhi, H.: Latent Semantic Kernels. Journal of Intelligent Information Systems, Vol. 18, 2002, No. 2-3, pp. 127–152.

[13] Özgür, L.—Güngör, T.: Text Classification with the Support of Pruned Dependency Patterns. Pattern Recognition Letters, Vol. 31, 2010, No. 12, pp. 1598–1607.

[14] Wittek, P.—Darányi, S.—Tan, C. L.: Improving Text Classification by a Sense Spectrum Approach to Term Expansion. In: Proceedings of the 13th Conference on Computational Natural Language Learning (CoNLL '09), Boulder, CO, USA, 2009, pp. 183–191.

[15] Cancedda, N.—Mahé, P.: Factored Sequence Kernels. Neurocomputing, Vol. 72, 2009, No. 7-9, pp. 1407–1413.

[16] Vishwanathan, S.—Smola, A. J.: Fast Kernels for String and Tree Matching. Advances in Neural Information Processing Systems, Vol. 15, 2003, pp. 569–576.

[17] Teo, C. H.—Vishwanathan, S. V. N.: Fast and Space Efficient String Kernels using Suffix Arrays. In: Proceedings of the 23th International Conference on Machine Learning, Pittsburgh, Pennsylvania, USA, 2006, pp. 929–936.

[18] Suzuki, J.—Isozaki, H.: Sequence and Tree Kernels with Statistical Feature Mining. Advances in Neural Information Processing Systems, Vol. 18, 2006, pp. 1321–1328.

[19] Siolas, G.—d'Alché-Buc, F.: Support Vector Machines based on a Semantic Kernel for Text Categorization. In: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN '00), Como, Italy, 2000, Vol. 5, pp. 205–209.

[20] Moschitti, A.: Kernel Methods, Syntax and Semantics for Relational Text Categorization. In: Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM '08), Napa Valley, California, USA, 2008, pp. 253–262.

[21] Wang, P.—Domeniconi, C.: Building Semantic Kernels for Text Classification using Wikipedia. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08), Las Vegas, Nevada, USA, 2008, pp. 713–721.

[22] Hickl, A.—Williams, J.—Bensley, J.—Roberts, K.—Shi, Y.—Rink, B.: Question Answering with LCC's CHAUCER at TREC 2006. In: Proceedings of the 15th Text Retrieval Conference (TREC'06), 2006, pp. 283–292.

[23] Voorhees, E. M.: Overview of the TREC 2004 Question Answering Track. In: Proceedings of the 13th Text Retrieval Conference (TREC '04), 2004, pp. 52–62.

[24] Fürnkranz, J.—Mitchell, T.—Riloff, E.: A Case Study in Using Linguistic Phrases for Text Categorization on the WWW. In: Working Notes of the AAAI/ICML Workshop on Learning for Text Categorization, 1998, pp. 5–12.

[25] Kehagias, A.—Petridis, V.—Kaburlasos, V. G.—Fragkou, P.: A Comparison of Word- and Sense-Based Text Categorization Using Several Classification Algorithms. Journal of Intelligent Information Systems, Vol. 21, 2003, No. 3, pp. 227–247.

[26] Salton, G.—Buckley, C.: Term-Weighting Approaches in Automatic Text Retrieval. Information Processing and Management, Vol. 24, 1988, No. 5, pp. 513–523.

[27] Schölkopf, B.—Smola, A. J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, 2002.

[28] Rakotomamonjy, A.—Bach, F. R.—Canu, S.—Grandvalet, Y.: SimpleMKL. Journal of Machine Learning Research, Vol. 9, 2008, pp. 2491–2521.

[29] Kernighan, B. W.—Ritchie, D. M.: The C Programming Language (Second Edition). Prentice Hall, 1988.

[30] Chang, C. C.—Lin, C. J.: LIBSVM: A Library for Support Vector Machines. Available at: `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Lujiang Zhang received the B. Sc. degree in computer software and theory from Chinese Academy of Sciences. Currently, he is a Ph. D. candidate in School of Automation Science and Electrical Engineering, Beijing University of Aeronautics & Astronautics, China. His research interests include machine learning and software analysis.

Xiaohui Hu received the Ph. D. degree from School of Computer Science and Engineering, Beijing University of Aeronautics & Astronautics, China. Currently, he is a Professor in School of Automation Science and Electrical Engineering, Beijing University of Aeronautics & Astronautics. His research interests include information systems integration and computer simulation technology.