Loose Phrase String Kernels

Janez Brank
Department of Knowledge Technologies
Jozef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
e-mail: janez.brank@ijs.si

Abstract

When representing textual documents by feature vectors for the purposes of further processing (e.g. for categorization, clustering, or visualization), one possible representation is based on "loose phrases" (also known as "proximity features"). This is a generalization of n-grams: a loose phrase is considered to appear in a document if all the words from the phrase occur sufficiently close to each other. We describe a kernel that corresponds to the dot product of documents under a loose phrase representation. This kernel can be plugged into any kernel method to deal with documents in the loose phrase representation instead of the bag of words representation.

1 Introduction

When processing textual documents using data mining techniques (such as categorization, clustering, or visualization), each document must typically be represented by a feature vector. In this vector, each component indicates the number of occurrences of the feature in the document. (Various weighting and normalization schemes can be applied on top of that, such as the well-known TF-IDF weighting.) The features can be defined in various ways; the most common approach is to define one feature for each term that occurs anywhere in our corpus of documents. The corresponding component of the vector is simply the number of occurrences of the term in the document. This representation is known as the "bag of words" or the "vector space model".

However, several other kinds of features have also been proposed, for example n-grams, which are sequences of n adjacent words. An n-gram is said to have occurred in the document if all of its words occur one immediately after another in the correct order. n-grams have been shown to improve performance vis-a-vis the bag of words model for some text categorization tasks [5]. The motivation for using n-grams is that an n-gram, since it consists of several adjacent words, may correspond to a phrase, which can be more informative than if each of its individual words is considered in isolation of the others. As an alternative to n-grams, phrases defined on linguistic principles (e.g.ñoun phrases, adjective phrases, etc.; or subjects, objects, predicates, etc.) can be used if a suitable natural-language parser is available.

One possible criticism of n-grams is that they can be too rigid since they insist that the words must appear in a particular order and with no words intervening in between. This rigidity can be said to be at odds with linguistic reality. The occurrence conditions can be relaxed in various ways, for example by removing the requirement that the words of the phrase must occur in a particular order (thus the phrase is treated as a bag of words rather than a sequence of words), and by allowing the words of the phrase to be interspersed by other words that are not part of the phrase. These relaxed requirements lead to "loose phrases", sometimes also known as "proximity features" since the only remaining condition is that the words of the phrase must occur sufficiently close to one another.

A downside of both n-gram representation is the large number of possible features; the resulting vectors can be huge and are only tractable because they are also very sparse. Care must be taken

to generate these vectors in an efficient manner [5]. The problem of such a representation is exacerbated if it leads to *non-sparse* vectors in the same (and very high-dimensional) vector space, which can easily happen during classification or clustering. If loose phrases are used instead of n-grams, the problem becomes even worse because the relaxed occurrence requirements mean that a lot more phrases now occur in a document than was the case under the stricter occurrence requirements of the traditional n-grams.

Thus, n-gram and loose phrase representations are good candidates to benefit from kernelization. Let X be the space of our documents and F be the vector space of our document representations (e.g. based on n-grams or loose phrases), in which a document $x \in X$ is represented by the vector $\phi(x) \in F$. A *kernel* is a function $K(x, \hat{x}) = \langle \phi(x), \phi(\hat{x}) \rangle_F$, where $\langle \cdot, \cdot \rangle_F$ is an inner product on F (e.g. the usual dot product of vectors). Many data mining methods, such as support vector machines [4], do not require us to be able to do anything more with the data except compute a kernel over it. They are collectively known as "kernel methods" [1]. This enables us to avoid the unwieldy explicit representations of very high-dimensional vectors $\phi(x)$ as long as we can compute a suitable kernel $K(x, \hat{x})$ without constructing $\phi(x)$ and $\phi(\hat{x})$ explicitly.

For the n-gram representation, a suitable kernel (the "string kernel") has been described by Lodhi et al. in [3]. Kernels for various other discrete structures besides strings have also been considered [2]. In this paper we present a kernel that corresponds to the loose phrase representation.

2 Definitions

Consider a *document* $d = d_1 \dots d_n$, where d_i is the i'th word. A number of *windows* are defined over the document, $W(d) = \{w_1, \dots, w_r\}$. For each window $w_k \in W(d)$ we have a set of indices, $I_k \subseteq 1..n$, and a set of *distinguished indices* $J_k \subseteq I_k$ such that the set $\{J_k : w_k \in W(d)\}$ is a partition of $1..n$. We will also impose the requirement that J_k must be disjoint with $\cup_{\kappa>k} I_\kappa$; this requirement will be helpful later when dealing with occurrences of phrases on a window-by-window basis while making sure that no occurrence is counted more than once. We will discuss some concrete examples of window families, and the corresponding distinguished index sets, at the end of the next section.

Given a *phrase* $t = bag(t_1, \dots, t_m)$, we define its *occurrences* thus:

$$occurs_d(t, I) \Leftrightarrow t = bag(d_i : i \in I) \\ \wedge \exists w \in W(d) : I \subseteq I_w.$$

Then the *frequency* of t in d is

$$\phi_d(t) = |\{I \subseteq 1..n : occurs_d(t, I)\}|.$$

A document can now be represented by a vector $\phi(d) := (\phi_d(t))_{t \in T}$, where T is the set of all phrases of length at most M. We would like to compute the kernel $K(d, \tilde{d}) = \langle \phi(d), \phi(\tilde{d}) \rangle$ as efficiently as possible.

3 The Loose Phrase String Kernel

Each occurrence I of t in d must be wholly located in some window; but it may actually be wholly located in several different windows. In any case, we can find the first window such that not only is I contained in that window but it but also contains one of the distinguished indices of that window. This can be used to partition the occurrences of t in d.

$$Occ_d(t, w_k) := \{I : occurs_d(t, I) \wedge \\ k = \arg\min\{\kappa \in 1..r : I \subseteq I_\kappa \wedge I \cap J_\kappa \neq \emptyset\}\}.$$

Each group of occurrences can be counted separately,

$$\phi_d(t, w_k) = |Occ_d(t, w_k)|.$$

Clearly, $\phi_d(t) = \sum_{k=1}^{r} \phi_d(t, w_k)$. Thus we get

$$\begin{aligned} K(d, \tilde{d}) &= \textstyle\sum_{t \in T} \phi_d(t) \phi_{\tilde{d}}(t) \\ &= \textstyle\sum_{t \in T} \sum_{k=1}^{r} \sum_{\tilde{k}=1}^{\tilde{r}} \phi_d(t, w_k) \phi_{\tilde{d}}(t, \tilde{w}_{\tilde{k}}) \\ &= \textstyle\sum_{k=1}^{r} \sum_{\tilde{k}=1}^{\tilde{r}} K(d, \tilde{d}, k, \tilde{k}) \end{aligned}$$

if we define

$$K(d, \tilde{d}, k, \tilde{k}) = \textstyle\sum_{t \in T} \phi_d(t, w_k) \phi_{\tilde{d}}(t, \tilde{w}_{\tilde{k}}).$$

It turns out that

$$I \in Occ_d(t, w_k) \Leftrightarrow t = bag(d_i : i \in I) \\ \wedge I \subseteq I_k \wedge I \cap J_k \neq \emptyset.$$

Proof: ($\Rightarrow$) The fact that $I \in Occ_d(t, w_k)$ implies that k is the smallest κ such that $I \subseteq I_\kappa$ and $I \cap J_\kappa \neq \emptyset$, which means that $I \subseteq I_k$ and $I \cap J_k \neq \emptyset$. Additionally, $I \in Occ_d(t, w_k)$ implies $occurs_d(t, I)$, which implies $t = bag(d_i : i \in I)$. ($\Leftarrow$) From $t = bag(d_i : i \in I)$ and $I \subseteq I_k$ it follows that $occurs_d(t, I)$. To show that $I \in Occ_d(t, w_k)$, it remains to prove that k is the smallest κ such that $I \subseteq I_\kappa$ and $I \cap J_\kappa \neq \emptyset$. Clearly these two statements are true for $\kappa = k$. Suppose that they were true for some $\kappa < k$; but for such a κ, we know that $J_\kappa \cap I_k = \emptyset$; so I, being non-disjoint with J_κ, contains some element from J_κ, which (because J_κ is disjoint with I_k) cannot belong to I_k; but this contradicts our assumption that $I \subseteq I_k$. $\square$

Thus, we see that to compute $K(d, \tilde{d}, k, \tilde{k})$, we need look only at those phrases that occur entirely in the window w_k of d and contain some distinguished index from w_k. A similar condition can be stated regarding the occurrence of these phrases in $\tilde{d}$.

We will define the following bags:

$$A_k = bag(d_i : i \in I_k), \quad R_k = bag(d_i : i \in J_k),$$
$$\tilde{A}_{\tilde{k}} = bag(\tilde{d}_i : i \in \tilde{I}_{\tilde{k}}), \quad \tilde{R}_{\tilde{k}} = bag(\tilde{d}_i : i \in \tilde{J}_{\tilde{k}}).$$

For a phrase t to contribute towards $\phi_d(t, w_k)$, it must be a sub-bag of A_k and non-disjoint with R_k (and analogously regarding the other document).

Let $U = \{d_1, \ldots, d_n, \tilde{d}_1, \ldots, \tilde{d}_{\tilde{n}}\}$ be the set of all words appearing in the two documents. Let us order the set U in an arbitrary manner, writing it as $U = \{u_1, \ldots, u_S\}$. Now any bag B constructed over U can be thought of as a function $B : 1..S \to \mathbb{Z}_0^+$, where $B(s)$ is the number of times that the word u_s is an element of B.

This notation can be further generalized to count the number of times that a bag $\tilde{B}$ is a sub-bag of B: $B(\tilde{B}) := \prod_{s=1}^{S} \binom{B(s)}{\tilde{B}(s)}$.

For a phrase t, we will be interested in $\phi_d(t, w_k)$, which is the number of times that t is a sub-bag of A_k but excluding occurrences located wholly in $A_k - R_k$. Thus we see that $\phi_d(t, w_k) = A_k(t) - F_k(t)$, where $F_k = A_k - R_k$. This means that

$$K(d, \tilde{d}, k, \tilde{k}) = \sum_{t \in T} (A_k(t) - F_k(t))(\tilde{A}_{\tilde{k}}(t) - \tilde{F}_{\tilde{k}}(t)),$$

which can clearly be split into four sums $\sum_{t \in T} B(t)\tilde{B}(t)$, where B is either A_k or F_k and $\tilde{B}$ is either $\tilde{A}_{\tilde{k}}$ or $\tilde{F}_{\tilde{k}}$.

The remaining problem is how to compute, given two bags B and $\tilde{B}$, the sum

$$K(B, \tilde{B}) := \sum_{t \in T} B(t)\tilde{B}(t) = \sum_{t \in T} \prod_{s=1}^{S} \binom{B(s)}{t(s)} \binom{\tilde{B}(s)}{t(s)}.$$

Remember that T contains all phrases of at most M words. (Even if we set $M = \infty$ to avoid any explicit constraint on the word length, this will be equivalent to using $M = \min\{|B|, |\tilde{B}|\}$ since no phrase longer than that could possibly occur in both B and $\tilde{B}$.) T can be partitioned into several sets based on the number of occurrences of the first word: $T = \cup_{c=0}^{M} T_c$, where $T_c = \{t \in T : t(1) = c\}$.

$$\begin{aligned} K(B, \tilde{B}) &= \textstyle\sum_{c=0}^{M} \sum_{t \in T_c} \prod_{s=1}^{S} \binom{B(s)}{t(s)} \binom{\tilde{B}(s)}{t(s)} \\ &= \textstyle\sum_{c=0}^{M} \sum_{t \in T_c} \binom{B(1)}{c} \binom{\tilde{B}(1)}{c} \prod_{s=2}^{S} \binom{B(s)}{t(s)} \binom{\tilde{B}(s)}{t(s)} \\ &= \textstyle\sum_{c=0}^{M} \binom{B(1)}{c} \binom{\tilde{B}(1)}{c} \sum_{t \in T_c} \prod_{s=2}^{S} \binom{B(s)}{t(s)} \binom{\tilde{B}(s)}{t(s)}. \end{aligned}$$

In the last expression, the inner sum

$$\textstyle\sum_{t \in T_c} \prod_{s=2}^{S} \binom{B(s)}{t(s)} \binom{\tilde{B}(s)}{t(s)} = (\dagger)$$

does not make any use of the word u_1 whatsoever. If we remove all occurrences of u_1 from all the phrases of T_c, we obtain the set of all phrases containing at most $M - c$ words and using only $\{u_2, \ldots, u_S\}$ as the base set, rather than the entire $U = \{u_1, u_2, \ldots, u_S\}$. We will denote this set of phrases by $T^{M-c,2}$. The original T is actually $T^{M,1}$ in this new notation. In general, we will use $T^{m,s}$ to refer to the set of all bags of at most m elements constructed over the set $\{u_s, \ldots, u_S\}$. We thus see that ($\dagger$) remains unchanged if we sum over $t \in T^{M-c,2}$ rather than over $t \in T_c$. This leaves us with a problem very similar to the original one. Defining

$$K^{m,s}(B, \tilde{B}) := \textstyle\sum_{t \in T^{m,s}} B(t)\tilde{B}(t),$$

we see that

$$K^{m,s}(B, \tilde{B}) = \textstyle\sum_{c=0}^{m} \binom{B(s)}{c} \binom{\tilde{B}(s)}{c} K^{m-c,s+1}(B, \tilde{B}).$$

This recursive formula is useful for $s \leq S$, while the case for $s = S + 1$ is actually trivial: by then we are constructing bags over an empty set, therefore only an empty bag can be constructed, and it occurs exactly once in both B and $\tilde{B}$. Thus we have $K^{m,s}(B, \tilde{B}) = 0$ for $m < 0$ and

$K^{m,S+1}(B, \tilde{B}) = 1$ for $m \geq 0$. In this way we can compute $K(B, \tilde{B}) = K^{M,1}(B, \tilde{B})$ using dynamic programming in $O(M^2 S)$ time. (But keep in mind that there is no need to use $M \geq \min\{|B|, |\tilde{B}|\}$.)

Note that we could also generalize our kernel by assigning length-dependent weights to the phrases. The difference

$$\hat{K}^{m,1}(B, \tilde{B}) := K^{m,1}(B, \tilde{B}) - K^{m-1,1}(B, \tilde{B})$$

is actually the sum of $B(t)\tilde{B}(t)$ computed over all phrases t with *exactly* m words (rather than *at most* m words). If we want the kernel to act as if all the frequencies $\phi_d(t)$ (and $\phi_{\tilde{d}}(t)$) for phrases t of exactly m words have been multiplied by some constant weight ω_m, we can compute this kernel thus:

$$K_\omega(B, \tilde{B}) := \sum_{m=0}^{M} \sum_{m=0}^{M} \omega_m^2 \hat{K}^{m,1}(B, \tilde{B}).$$

(Alternatively, $\hat{K}^{m,s}$ could be obtained using the same recurrence as $K^{m,s}$, except that the trivial case $\hat{K}^{m,S+1}$ would be set to 1 only for $m = 0$, while for $m > 1$ it would be set to 0.)

Now that we know how to compute $K(B, \tilde{B})$, we can use this algorithm in the previously derived expressions for $K(d, \tilde{d}, k, \tilde{k})$ and finally $K(d, \tilde{d})$. We can compute $K(d, \tilde{d})$ in $O(M^2 S r \tilde{r})$ time, where M need only be as large as the length of the longest window in the two documents d and $\tilde{d}$. As for S, it can in fact also be defined separately for each pair of windows, so that it needs only contain as many words as the combined length of the two windows. Some examples of practical interest include:

- Classical sliding windows of length l, i.e. $r = n$, $I_k = \{k, k+1, \ldots, \min\{k+l-1, n\}\}$, $J_k = \{k\}$. In this case we can use $M \leq l$, $S = l$, yielding $O(M^2 l n^2)$. In practice one would typically not be interested in very small phrases, limiting M to e.g. 5 or less. Thus the time complexity is practically $O(l n^2)$.

- Ignoring proximity altogether. In this case we have one window of length n: $r = 1$, $l = n$, $I_1 = J_1 = 1..n$, and thus the time complexity is $O(M^2 n^3)$. For quite long documents it would actually be more reasonable to assume $S \ll n$, because not all words will be different; $S = O(\sqrt{n})$ would be a typical observation in practice.

- Nonoverlapping windows of average length l (e.g. one sentence per window), with $J_k = I_k$ for all k. In this case we have $O(M^2 l (n/l)^2) = O(M^2 n^2 / l)$.

- Sliding windows on the sentence level: each window consists of p contiguous sentences, each sentence being l words long on average. Words from the first sentence of the window are distinguished indices for that window. In this case we have $r = n/l$ windows, each being lp words long, for a time complexity of $O(M^2 p n^2 / l)$.

4 Conclusions and future work

In this paper we presented a kernel that computes the dot product of two documents under the loose-phrase representation. The kernel is a natural generalization of the string kernels that have been previously described in the literature. In future work, we intend to use this kernel in text classification tasks to compare the performance of the loose-phrase representation with that of the traditional bag-of-words model.

References

[1] J. Shawe-Taylor, N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004

[2] D. Haussler. *Convolution kernels on discrete structures*. Technical Report UCS-CRL-99-10, Dept. of Comp. Sci., Univ. of California at Santa Cruz, July 8, 1999.

[3] H. Lodhi, N. Christianini, J. Shawe-Taylor, C. Watkins. Text Classification using String Kernels. In *Advances of Neural Information Processing Systems 13*, 2000.

[4] C. Cortes, V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, September 1995.

[5] D. Mladenić, M. Grobelnik. Word sequences as features in text-learning. In *Proceedings of ERK-98, Seventh Electrotechnical and Computer Science Conference*, pp. 145–148, Ljubljana, 1998.