Word-Sequence Kernels

Nicola Cancedda NICOLA.CANCEDDA@XRCE.XEROX.COM
Eric Gaussier ERIC.GAUSSIER@XRCE.XEROX.COM
Cyril Goutte CYRIL.GOUTTE@XRCE.XEROX.COM
Jean-Michel Renders JEAN-MICHEL.RENDERS@XRCE.XEROX.COM
Xerox Research Centre Europe
6, chemin de Maupertuis
38240 Meylan, France

Editors: Jaz Kandola, Thomas Hofmann, Tomaso Poggio and John Shawe-Taylor

Abstract

We address the problem of categorising documents using kernel-based methods such as Support Vector Machines. Since the work of Joachims (1998), there is ample experimental evidence that SVM using the standard word frequencies as features yield state-of-the-art performance on a number of benchmark problems. Recently, Lodhi et al. (2002) proposed the use of *string kernels*, a novel way of computing document similarity based of matching non-consecutive subsequences of characters. In this article, we propose the use of this technique with sequences of *words* rather than characters. This approach has several advantages, in particular it is more efficient computationally and it ties in closely with standard linguistic pre-processing techniques. We present some extensions to sequence kernels dealing with symbol-dependent and match-dependent decay factors, and present empirical evaluations of these extensions on the Reuters-21578 datasets.

Keywords: Kernel machines, text categorisation, linguistic processing, string kernels, sequence kernels

1. Introduction

The application of machine learning techniques to classification of documents is a rich and challenging research area with many related tasks, such as routing, filtering or cross-lingual information retrieval. Since Joachims (1998) and other researchers like Yang and Liu (1999) have shown that Support Vector Machines (SVM) perform favourably compared to competing techniques for document categorisation, kernel machines have been a popular choice for document processing. In most reported works, however, documents were represented using the standard vector space, aka *bag-of-word* model (Salton and McGill, 1983)—that is, more or less, word frequencies with various added normalisations—in conjunction with general purpose kernels (linear, polynomial, RBF, etc.).

Recently, Watkins (1999) and Lodhi et al. (2001, 2002) proposed the use of *string kernels*, one of the first significant departures from the vector space model. In string kernels, the features are not word frequencies or an implicit expansion thereof, but the extent to which all possible ordered subsequences of characters are represented in the document. In addition, Lodhi et al. (2001) proposed a recursive dynamic programming formulation allowing the practical computation of the similarity between two sequences of arbitrary symbols as the dot-product in the implicit feature space of all ordered, non-consecutive subsequences of symbols. Although it allows to perform the kernel calculation without performing an explicit feature space expansion, this formulation is extremely computationally demanding and is not applicable (with current processing power) to large document collections without approximation (Lodhi et al., 2002).

In this document, we propose to extend the idea of sequence kernels to process documents as sequences of words. This greatly expands the number of symbols to consider, as symbols are words rather than characters, but it reduces the average number of symbols per document. As the dynamic programming formulation used for computing sequence matching depends only on sequence length, this yields a significant improvement in computing efficiency. Training a SVM on a dataset of around 10000 documents like the Reuters-21578 corpus becomes feasible without approximation. In addition, matching sequences of words allows to work with symbols that are expected to be more linguistically meaningful. This leads to extensions of the word sequence kernels that implement a kind of inverse document frequency (IDF) weighting by allowing symbol-varying decay factors. Words may also be equivalent in some context, and we show how to implement soft word matching in conjunction with the word-sequence kernel. This allows the use of this kernel in a multi-lingual context, an application that was not possible with the string kernel formulation.

In Section 2 we offer a brief self-contained introduction to kernel methods, after which we will proceed with the presentation of sequence kernels (Section 2.2). Section 3 formulates and verifies a hypothesis that helps explain why string kernels perform well, even though they are based on processing entities (characters) which are essentially meaningless from a linguistic point of view. Section 4 presents our extension of sequence kernels to word-sequences, which we test empirically in Section 4.2. Finally, we present in Section 5 some extensions of word-sequence kernels to soft-matching and cross-lingual document similarity.

2. Kernel Methods

Kernel methods are a research field in rapid expansion. Based on theoretical work on statistical learning theory (Vapnik, 1995) and reproducing kernel Hilbert spaces (Wahba, 1990), kernel machines were first popular as SVM (Boser et al., 1992, Cortes and Vapnik, 1995). In this section we will briefly introduce *kernel-based classifiers* and *sequence kernels*. For additional information, the reader may refer to general introductions such as Cristianini and Shawe-Taylor (2000), Schölkopf and Smola (2002) or Herbrich (2002).

2.1 Kernel Classifiers

A binary classifier is a function from some input space $\mathcal{X}$ into the set of binary labels $\{-1,+1\}$. A supervised learning algorithm is a function that assigns, to each labeled training set $S \in (\mathcal{X} \times \{-1,+1\})^l$, a binary classifier $h : \mathcal{X} \to \{-1,+1\}$. For computational reasons and in order to limit overfitting, learning algorithms typically consider only a subset of all classifiers, $\mathcal{H} \subset \{h | h : \mathcal{X} \to \{-1,+1\}\}$, the *hypothesis space*. In the common case in which $\mathcal{X}$ is a vector space, one of the simplest binary classifiers is given by the linear discriminant:

$$h(\mathbf{x}) = \operatorname{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

where $\langle \cdot, \cdot \rangle$ denotes the standard dot-product. Learning the linear classifier amounts to finding values of $\mathbf{w}$ and b which maximise some measure of performance. Learning algorithms for linear classifiers include e.g. Fisher's linear discriminant, the Perceptron and SVM. All linear classifiers will naturally fail when the boundary between the two classes is not linear. However, it is possible to leverage their conceptual simplicity by projecting the data from $\mathcal{X}$ into a different *feature space* $\mathcal{F}$ in which we hope to achieve linear separation between the two classes. Denoting the projection into feature space by $\phi : \mathcal{X} \to \mathcal{F}$, the generalised linear classifier becomes:

$$h(\mathbf{x}) = \operatorname{sign}(\langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b) \tag{1}$$

The separating hyperplane defined by $\mathbf{w}$ and b is now in the feature space $\mathcal{F}$, and the corresponding classifier in the input space $\mathcal{X}$ will generally be non-linear. Note also that in this setting, $\mathcal{X}$ need not even be a vector space as long as $\mathcal{F}$ is. This is a crucial observation for the rest of this article, as we will be dealing with the case in which $\mathcal{X}$ is the associative monoid of the sequences over a fixed alphabet, which is not a vector space.

In the case of SVM, the optimal weights $\widehat{\mathbf{w}}$ can be expressed as a linear combination of the data:

$$\widehat{\mathbf{w}} = \sum_{i=1}^{l} y_i \alpha_i \phi(\mathbf{x}_i) \tag{2}$$

where $\alpha_i \in \mathbb{R}$ is the *weight* of the i-th training example with input $\mathbf{x}_i \in \mathcal{X}$ and label $y_i \in \{-1,+1\}$. Accordingly, by combining (1) and (2), the SVM classifier can be written in its *dual form*:

$$h(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{l} y_i \alpha_i \langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle + b \right) \tag{3}$$

Notice that the projection ϕ only appears in the context of a dot product in (3). In addition, the optimal weights (α_i) are the solution of a high dimensional quadratic programming problem which can be expressed in terms of dot products between projected training data $\langle\phi(\mathbf{x}_i),\phi(\mathbf{x}_j)\rangle$. Rather than use the explicit mapping ϕ, we can thus use a *kernel function* $K : X \times X \to \mathbb{R}$ such that $K(\mathbf{x},\mathbf{y}) = \langle\phi(\mathbf{x}),\phi(\mathbf{y})\rangle$. As long as $K(\mathbf{x},\mathbf{y})$ can be calculated efficiently, there is actually no need to explicitly map the data into feature space using ϕ —this is the *kernel trick*. It is especially useful when the computational cost of calculating $\phi(\mathbf{x})$ is overwhelming (e.g., when $\mathcal{F}$ is high dimensional) while there is a simple expression for $K(\mathbf{x},\mathbf{y})$. In fact, *Mercer's theorem* (see, e.g. Cristianini and Shawe-Taylor, 2000, Section 3.3.1) states that any positive semi-definite symmetric function of two arguments corresponds to some mapping ϕ in some space $\mathcal{F}$ and is thus a valid kernel. This means that in principle a kernel can be used even if the nature of the corresponding mapping ϕ is not known.

This article addresses the application of kernel methods to the well known problem of document categorisation. So far, most of the kernel-based work in this area used a standard *bag-of-words* representation, with word frequencies as features, ie $X = \mathbb{R}^n$. Although these vector representations have proved largely successful, we will here investigate the use of representations which are intuitively closer to the intrinsic nature of documents, namely sequences of symbols. This representation does not form a vector space, but the use of appropriate kernels will allow us to leverage the discriminative power of kernel-based algorithms. As we will see, the kernels we are going to consider will also allow us to test basic factors (e.g., order or locality of multi-word terms) in the context of document categorisation.

2.2 Sequence Kernels

The assumption behind the popular *bag-of-words* representation is that the relative position of tokens has little importance in most Information Retrieval (IR) tasks. Documents are just represented by word frequencies (and possibly additional weighting and normalisation), and the loss of the information regarding the word positions is more than compensated for by the use of powerful algorithms working in vector space. Although some methods inject positional information using phrases or multi-word units (Perez-Carballo and Strzalkowski, 2000) or local co-occurrence statistics (Wong et al., 1985), the underlying techniques are based on a vector space representation.

String kernels (or *sequence kernels*) are one of the first significant departures from the vector-space representation in this domain. String kernels (Haussler, 1999, Watkins, 1999, Lodhi et al., 2001, 2002) are similarity measures between documents seen as *sequences* of symbols (e.g., possible characters) over an alphabet. In general, similarity is assessed by the number of (possibly non-contiguous) matching subsequences shared by two sequences. Non contiguous occurrences are penalized according to the number of gaps they contain.

Formaly, let Σ be a finite alphabet, and $s = s_1 s_2 \ldots s_{|s|}$ a sequence over Σ (i.e. $s_i \in \Sigma, 1 \leq i \leq |s|$). Let $\mathbf{i} = [i_1, i_2, \ldots, i_n]$, with $1 \leq i_1 < i_2 < \ldots < i_n \leq |s|$, be a subset of the indices in s: we will indicate as $s[\mathbf{i}] \in \Sigma^n$ the subsequence $s_{i_1} s_{i_2} \ldots s_{i_n}$. Note that $s[\mathbf{i}]$ does not necessarily form a contiguous subsequence of s. For example, if s is the sequence *CART* and $\mathbf{i} = [2,4]$, then $s[\mathbf{i}]$ is AT. Let us write $l(\mathbf{i})$ the length spanned by $s[\mathbf{i}]$ in s, that is: $l(\mathbf{i}) = i_n - i_1 + 1$. The sequence kernel of two strings s and t over Σ is defined as:

$$K_n(s,t) = \sum_{u \in \Sigma^n} \sum_{\mathbf{i}:s[\mathbf{i}]=u} \sum_{\mathbf{j}:t[\mathbf{j}]=u} \lambda^{l(\mathbf{i})+l(\mathbf{j})} \tag{4}$$

where $\lambda \in]0;1]$ is a decay factor used to penalise non-contiguous subsequences and the first sum refers to all possible subsequences of length n. Equation 4 defines a valid kernel as it amounts to performing an inner product in a feature space with one feature per ordered subsequence $u \in \Sigma^n$ with value:

$$\phi_u(s) = \sum_{\mathbf{i}:s[\mathbf{i}]=u} \lambda^{l(\mathbf{i})} \tag{5}$$

Intuitively, this means that we match all possible subsequences of n symbols, even when these subsequences are not consecutive, and with each occurrence "discounted" according to its overall length.

A direct computation of all the terms under the nested sum in (4) becomes impractical even for small values of n. In addition, in the spirit of the kernel trick discussed in Section 2, we wish to calculate $K_n(s,t)$ directly rather than perform an explicit expansion in feature space. This can be done using a recursive formulation proposed by Lodhi et al. (2001), which leads to a more efficient dynamic-programming-based implementation. The derivation of this efficient recursive formulation is given in appendix A, together with an illustrative example. It results in the following equations:

$$\begin{aligned}
K'_0(s,t) &= 1, \text{ for all } s,t, \\
&\quad \text{for all } i = 1,\ldots,n-1: \\
K''_i(s,t) &= 0, \text{ if } \min(|s|,|t|) < i && (6)\\
K'_i(s,t) &= 0, \text{ if } \min(|s|,|t|) < i && (7)\\
K''_i(sx,ty) &= \lambda K''_i(sx,t) \quad \text{iff } y \neq x && (8)\\
K''_i(sx,tx) &= \lambda K''_i(sx,t) + \lambda^2 K'_{i-1}(s,t) \quad \text{otherwise} && (9)\\
K'_i(sx,t) &= \lambda K'_i(s,t) + K''_i(sx,t) && (10)\\
&\quad \text{and finally:} \\
K_n(s,t) &= 0, \text{ if } \min(|s|,|t|) < n && (11)\\
K_n(sx,t) &= K_n(s,t) + \sum_{j:t_j=x} \lambda^2 K'_{n-1}(s,t[1:j-1]) && (12)
\end{aligned}$$

The time required to compute the kernel according to this formulation is $O(n|s||t|)$. In some situations, it may be convenient to perform a linear combination of sequence kernels with different values of n. Using the recursive formulation, it turns out that computing all kernel values for subsequences of lengths up to n is not significantly more costly than computing the kernel for n only. In order to keep the kernel values comparable for different values of n and to be independent from the length of the strings , it may be advisable to consider the normalised version:

$$\hat{K}(s,t) = \frac{K(s,t)}{\sqrt{K(s,s)K(t,t)}} \qquad (13)$$

This is a generalisation of the “cosine normalisation” which is widespread in IR and corresponds to the mapping $\hat{\phi}(s) = \frac{\phi(s)}{\|\phi(s)\|_2}$ (where $\|.\|_2$ is the 2-norm).

3. The Noisy-Stemming Hypothesis

Text categorisation using string kernels operating at the character level has been shown to yield performance comparable to kernels based on the traditional bag-of-words representation (Lodhi et al., 2001, 2002). This result is somewhat surprising, considering string kernels use only low-level information. A possible explanation of the effectiveness of the string kernel, supported by the observation that performance improves with n, is that subsequences most relevant to the categorisation may correspond to noisy versions of word stems, thus implicitly implementing some form of morphological normalisation. Furthermore, as gaps within the sequence are allowed —although penalized— string kernels could also pick up stems of consecutive words, as illustrated in Figure 1. This is known to help in IR applications (Salton and McGill, 1983).

In order to test this *noisy stemming* hypothesis, the following experiment was conducted. An SVM using a string kernel was trained on a small subset of the Reuters-21578 corpus,[1] using the labels for the `acq` topic. A sample of features—subsequences of a fixed length n—was randomly chosen. In order to avoid a vast majority of irrelevant low-weight features, we restricted ourselves to possibly non-contiguous subsequences appearing in the Support Vectors. For each sampled feature u, its impact on the categorisation decision was compared to a measure of *stemness* σ_u. The impact on the categorisation decision is measured by the absolute

1. Available at http://www.daviddlewis.com/resources/testcollections/reuters21578/. The subset is built from the first 114 documents with the `acq` label and the first 266 documents not in `acq`.

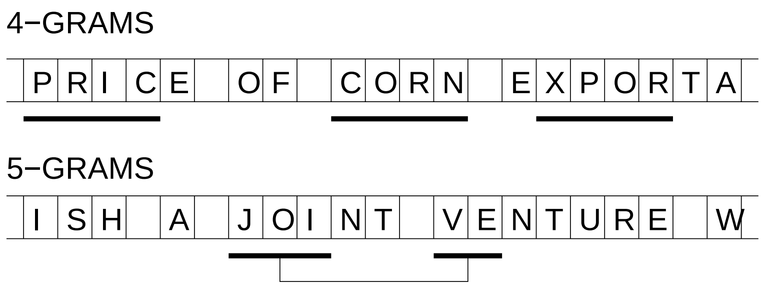
4−GRAMS
P R I C E O F C O R N E X P O R T A
5−GRAMS
I S H A J O I N T V E N T U R E W

Figure 1: The noisy-stemming hypothesis. N-grams may be able to pick parts of words/stems (top, with $n=4$) or even pick parts of stems of consecutive words (bottom, $n=5$).

weight w_u in the linear decision:

$$w_u = \left| \sum_j \alpha_j y_j \phi_u(x_j) \right|$$

In order to assess the *stemness* of a feature, ie the extent to which a feature approximates a word stem, a simplifying assumption was made by considering that stems are the initial characters in a word. Moreover, in order to verify that a feature may span two consecutive stems, a measure of stemness was devised so as to consider pairs of consecutive tokens. Let $\mathbf{t} = \{t_1, \ldots, t_m\}$, with $1 = t_1 < t_2 < \ldots < t_m < t_{m+1} = |s|+1$, be a *tokenisation* of the string s (the tokenisation is the segmentation process that provides the indices of the first symbol of each token). Let $m(k,u)$ be the set of all matches for the subsequence u in the pair of consecutive tokens starting in t_{k-1} and t_k respectively, with $2 \leq k \leq m$:

$$m(k,u) = \left\{ \left(\mathbf{i}', \mathbf{i}''\right) | u = s[\mathbf{i}']s[\mathbf{i}''], t_{k-1} \leq \mathbf{i}'_1 \leq \mathbf{i}'_{|\mathbf{i}'|} < t_k \leq \mathbf{i}''_1 \leq \mathbf{i}''_{|\mathbf{i}''|} < t_{k+1} \right\}$$

In a matching pair where $\mathbf{i}' = \emptyset$, the constraint $t_{k-1} \leq \mathbf{i}'_{|\mathbf{i}'|} < t_k$ is considered automatically satisfied. In addition, $\mathbf{i}''$ is forced to be different from $\emptyset$, in order to avoid double counting of single word matches, and an empty symbol is added at the beginning of the string to allow one-word matching on the first token. The stemness of a subsequence u with respect to s is then defined as:

$$\sigma_u(s) = \text{avg}_{k:m(k,u)\neq\emptyset} \left(\lambda^{\min_{m(k,u)} \left(\mathbf{i}'_{|\mathbf{i}'|} - t_{k-1} - |\mathbf{i}'| + \mathbf{i}''_{|\mathbf{i}''|} - t_k - |\mathbf{i}''| \right)} \right)$$

In other words, for each pair of consecutive tokens containing a match for u, a pair of indices $(\mathbf{i}', \mathbf{i}'')$ is selected such that $s[\mathbf{i}']$ matches into the first token, $s[\mathbf{i}'']$ matches in the second and each component is found as compact and close to the start of the corresponding token as possible. The stemness of u with respect to the whole training corpus is then defined as the (micro-)average of the stemness in all documents in the corpus. This experiment was performed with a sample of 15000 features of length $n=3$, and a sample of 9000 features of $n=5$, with $\lambda = 0.5$ in both cases. Under the noisy-stemming hypothesis, we expect relatively few features with high weight and low stemness. The outcome of our experiment, presented in Table 1, seems to confirm this. For n=3, among features with high weight (> 0.15) about 92% have high stemness, whereas among features with low weight the fraction of features with high stemness is of 73%. Similar results were found for $n=5$ (90% and 46% respectively).

For each non-empty set of matches $m(k,u) \neq \emptyset$, stemness is calculated by first finding the most compact match $(\mathbf{i}'_\mathbf{c}, \mathbf{i}''_\mathbf{c}) = \text{argmin}_{(\mathbf{i}',\mathbf{i}'')\in m(k,u)} \left(\mathbf{i}'_{|\mathbf{i}'|} - t_{k-1} - |\mathbf{i}'| + \mathbf{i}''_{|\mathbf{i}''|} - t_k - |\mathbf{i}''| \right)$. It is interesting to check how often this "minimal" subsequence matches on two tokens, as opposed to single-token match. For that purpose, we define $g(k,u) = 1$ if the most compact match is on one token (i.e. $\mathbf{i}'_\mathbf{c} = \emptyset$) and $g(k,u) = 2$ otherwise. The multigramness of a subsequence u on sequence s is then:

$$\gamma_u(s) = \text{avg}_{k:m(k,u)\neq\emptyset} \left(g(k,u) \right)$$

$n=3$	w_u	
σ_u	low	high
low	3622	125
high	9877	1376

$n=5$	w_u	
σ_u	low	high
low	4413	86
high	3686	815

Table 1: Contingency tables for weight against stemness. Left ($n=3$, ie trigrams): Weights are split at 0.15 to separate the lowest 90% and highest 10% values, whereas stemness is split at 0.064 to separate the 25% lowest and 75% highest values. Right ($n=5$): Weights are split at 0.0066 to separate the lowest 90% and highest 10% values, whereas stemness is split at 0.029 to separate the 50% lowest and 50% highest values. The χ^2 values for the tables are 247 (left) and 655 (right), which suggests a highly significant departure from independence in both cases (1 d.o.f.).

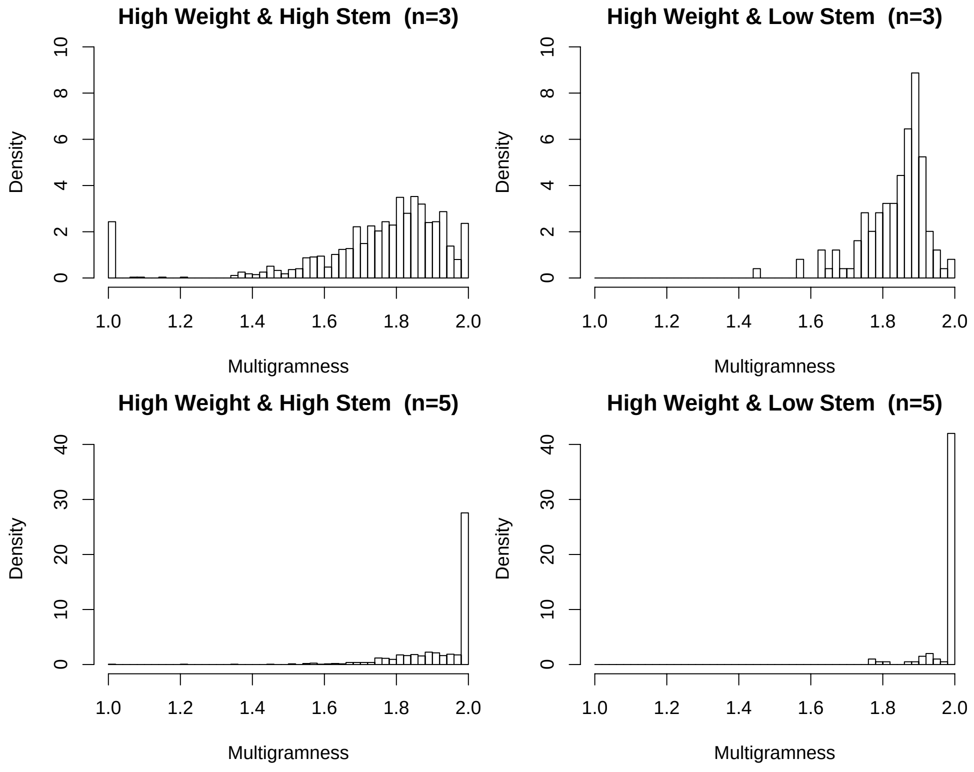

Figure 2: Density of “Multigramness” for high weight features, $n=3$ and $n=5$.

Using the same feature samples as above, we estimate the distribution of $\gamma_u(s)$ for features with high weights (the most interesting features). Figure 2 shows that the mass of each distribution is close to 2, indicating that matching occurs more often on more than one word. This effect is even clearer for larger n: high weight features of length $n=5$ are concentrated near $\gamma_u=2$. This is reasonable considering longer subsequences are harder to match on a single word. On the other hand, a significant amount of high weight, high stemness features of length $n=3$ are clearly single stem features. In addition, features with high weight but low stemness have a tendency to spread on two stems (right column vs. left column in Figure 2).

This suggests that multiple word matching really does occur and is beneficial in forming discriminant, high weight features. This is also in agreement with results on SVM using traditional kernels (Joachims, 1998) showing that polynomial kernels consistently outperform the linear kernel.

4. Word-Sequence Kernels

The seeming validity of the noisy-stemming hypothesis suggests that sequence kernels operating at the word (possibly word-stem) level might prove more effective than those operating at the character level. In this section, we describe the impact of applying sequence kernels at the word level, and extend the original kernel formulation so as to take into account additional information.

4.1 Theory

The direct application of the sequence kernel defined above on an alphabet of individual words raises some significant issues. First, the feature space on which documents are implicitly mapped has one dimension for each of the $|\Sigma|^n$ ordered n-tuples of symbols in the alphabet: going from characters to words, the order of magnitude of $|\Sigma|$ increases from the hundreds to the tens of thousands, and the number of dimensions of the feature space increases accordingly. However, the average length in symbols of documents decreases by an order of magnitude. As the algorithm used for computing sequence kernels depends on sequence length and not on alphabet size, this yields a significant improvement in computing efficiency: word-sequence kernels can be computed on datasets for which string kernels have to be approximated.

Nevertheless, the combination of the two effects (increase in alphabet size, decrease in document average length) causes documents to have extremely sparse implicit representations in the feature space. This in turn means that the kernel-based similarity between any pair of distinct documents will tend to be small with respect to the “self-similarity” of documents, especially for larger values of n. In other words, the Gram matrix tends to be nearly diagonal, meaning that all examples are nearly orthogonal. In order to overcome this problem, it is convenient to replace the fixed-length sequence kernel with a combination of sequence kernels with subsequence lengths up to a fixed n. A straightforward formulation is the following 1-parameter linear combination:

$$\bar{K}_n(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{n} \mu^{1-i} \hat{K}_i(\mathbf{x},\mathbf{y}) \tag{14}$$

Considering the dynamic programming technique used for the implementation, computing the sequence kernel for subsequences of length up to n requires only a marginal increase in time compared to the computation for subsequences of length n only. Notice that kernel values for different subsequence lengths are normalised independently before being combined. In this way it is possible to control the relative weight given to different subsequence lengths directly by means of the parameter μ. The μ parameter, or possibly the parameters of a general linear combination of sequence kernels of different orders, can be optimized by cross-validation, or alternatively by *kernel alignment* (Cristianini et al., 2001).

In its original form, the sequence kernel relies on mere occurrence counts, or term frequencies, and lacks feature weighting schemes which have been deemed important by the IR community. We want to present now two extensions to the standard sequence kernel. The first uses different values of λ to assign different weights to gaps, and allows the use of well-established weighting schemes, such as the inverse document frequency (IDF). It can also be used to discriminate symbols according to, say, their part-of-speech. The second extension consists in adopting different decay factors for gaps and for symbol matches, and generalises over the previous extension in the sense that informative symbols are treated differently depending on whether they are used in matches or gaps.

4.1.1 SYMBOL-DEPENDENT DECAY FACTORS

The original formulation of the sequence kernel uses a unique decay factor λ for all symbols. It can well be the case, however, that sequences containing some symbols have a significantly better discriminating power than sequences containing other symbols. For example, a sequence made of three nouns is more likely to be more informative than a sequence made of a preposition, a determiner and a noun. A way to leverage on this non-uniform discriminative power consists in assigning different decay factors to distinct symbols. We thus introduce a distinct λ_x for each $x \in \Sigma$. This induces the following new embedding:

$$\phi_u(s) = \sum_{\mathbf{i}:s[\mathbf{i}]=u} \prod_{i_1<j<i_n} \lambda_{s_j}$$

The value of the feature $u =$ *gas injection* in the sequence $s =$ *gas assist plastic injection* is then:

$$\phi_u(s) = \lambda_{gas}\lambda_{assist}\lambda_{plastic}\lambda_{injection}$$

The corresponding weighted sequence kernel K_w is thus defined as follows:

$$K_{\mathrm{w}n}(s,t) = \sum_{u\in\Sigma^n} \phi_u(s)\phi_u(t) = \sum_{u\in\Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \prod_{i_1\le k\le i_n} \prod_{j_1\le l\le j_n} \lambda_{s_k}\lambda_{t_l}.$$

A recursive formulation analogous to the one for the original sequence kernel can be derived. We can define functions K_w' and K_w'' to store intermediate results (see Appendix A):

$$\begin{aligned} K_{\mathrm{w}}{}_i'(s,t) &= \sum_{u\in\Sigma^i} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \prod_{i_1\le k\le |s|} \prod_{j_1\le j\le |t|} \lambda_{s_k}\lambda_{t_j}, \quad \text{for } i=1,\ldots,n-1 \\ K_{\mathrm{w}}{}_i''(sx,t) &= \sum_{j:t_j=x} K_{\mathrm{w}}{}_{i-1}'(s,t[1:j-1])\lambda_x \prod_{j\le l\le |t|} \lambda_{t_l}, \end{aligned}$$

and use the general recursion equations given in Section 2.2, with the following modifications: replace λ by λ_y in Equation 8, and by λ_x in Equations 9, 10 and 12.

Using different values of λ is one way of incorporating prior knowledge into the sequence kernel. In the case of word-sequence kernels, for instance, symbols (i.e. words) could be grouped according to their part-of-speech. By doing so it is possible, for instance, to penalize more heavily occurrences with extraneous nouns, such as *gas assist plastic injection*, than occurrences with extraneous adverbs, as *gas only injection*. The rationale behind this is that the latter sequence is semantically closer to *gas injection* than the former (indeed, the *plastic*, and not the *gas*, is the main substance being injected in the former sequence).

This same extension can be useful—in the case of word-sequence kernels—in order to integrate information on the inverse document frequency of terms. A broadly used formula for taking IDF into account in term weighting schemes is:

$$\mathrm{idf}_c = \log\left(\frac{N}{N_c}\right) \tag{15}$$

where N is the number of documents in the collection and N_c is the number of documents containing term c. This formula yields a value between 0 and $log(N)$ (only terms occurring in the collection are given non-zero weight), and need be normalised to be used as a decay factor:

$$\lambda_c = \frac{log(\frac{N}{N_c})}{log(N)} \tag{16}$$

However, one can notice that there is a tension between the different types of information which can be integrated. On the one hand, words pertaining to certain parts-of-speech tend to be non discriminative when inserted in sequences (as *only* in the example above), and should thus be assigned high λ's. But at the same time, the very same words tend to appear in many different documents, thus leading to low λ's when an IDF based weighting scheme is used. The second extension we propose solves this problem.

4.1.2 INDEPENDENT DECAY FACTORS FOR GAPS AND MATCHES

The previous section introduced variable decay factors for different symbols, and mentioned the inclusion of part-of-speech information and of inverse document frequency as background knowledge that could be added using this extension. However, one would like gaps containing highly relevant symbols (e.g.: nouns) to be strongly penalized (i.e.: λ_c small), but highly relevant *matching* symbols to be rewarded (i.e.: λ_c large). This requirement cannot be met with the previous formulation.

One way to obtain the desired behaviour is to introduce two separate sets of decay factors for gaps and for matches. The u coordinate of the feature vector for the sequence s is now defined by

$$\hat{\phi}_u(s) = \sum_{\mathbf{i}:u=s[\mathbf{i}]} \prod_{1\le j\le |u|} \lambda_{m,u_j} \prod_{i_1<k<i_{|u|},k\notin \mathbf{i}} \lambda_{g,s_k} \tag{17}$$

where $\lambda_{g,x}$ is the discount factor for the symbol x when it occurs as a gap, and $\lambda_{m,x}$ is the discount factor for the same symbol when it occurs as a matching symbol. With this definition one could, for instance, discount gaps according to part-of-speech and weight matching symbols according to inverse document frequency information. The value of the feature $u =$ *gas injection* in the sequence $s =$ *gas only injection* would thus be:

$$\phi_u(s) = \lambda_{m,gas}\lambda_{g,only}\lambda_{m,injection}$$

and we now have a way to consider *only* as unimportant when it appears in a gap ($\lambda_{g,only}$ close to 1), and uninformative when it appears in a match ($\lambda_{m,only}$ close to 0.). We now define the weighted sequence kernel K_{dw} of two sequences s and t as

$$\begin{aligned} K_{\text{dw}n}(s,t) &= \sum_{u\in\Sigma^n} \hat{\phi}_u(s)\hat{\phi}_u(t) \\ &= \sum_{u\in\Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \prod_{1\le j\le |u|} \lambda^2_{m,u_j} \prod_{i_1<k<i_{|u|},k\notin\mathbf{i}} \lambda_{g,s_k} \prod_{j_1<l<j_{|u|},l\notin\mathbf{j}} \lambda_{g,t_l} \end{aligned} \tag{18}$$

As for the evaluation of K_{dw}, we define K_{dw}' and K_{dw}'' as:

$$\begin{aligned} K_{\text{dw}}{}'_i(s,t) &= \sum_{u\in\Sigma^i} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \prod_{k=1}^{i} \lambda^2_{m,u_i} \prod_{i_1<l\le|s|,l\notin\mathbf{i}} \lambda_{g,s_l} \prod_{j_1<r\le|t|,r\notin\mathbf{j}} \lambda_{g,t_r}, \\ K_{\text{dw}}{}''_i(sx,t) &= \sum_{j:t_j=x} K_{\text{dw}}{}'_{i-1}(s,t[1:j-1])\lambda^2_{m,x} \prod_{l=j+1}^{|t|} \lambda_{g,t_l}, \quad \text{for } i=1,\ldots,n-1. \end{aligned}$$

and use the general recursion equations given in Section 2.2, with the following modifications: replace λ by $\lambda_{g,y}$ in Equation 8, replace Equation 9 with

$$K_{\text{dw}}{}''_i(sx,tx) = \lambda_{g,x} K_{\text{dw}}{}''_i(sx,t) + \lambda^2_{m,x} K_{\text{dw}}{}'_{i-1}(s,t)$$

replace λ by $\lambda_{g,x}$ in Equation 10 and replace λ by $\lambda_{m,x}$ in Equation 12.

This formulation shows that considering separate sets of decay factors for matching symbols and for gaps does not impact the computational complexity of the kernel. In the remainder of the paper, we will refer to the extension corresponding to Equation 17, as the word-sequence kernels with **symbol-dependent matching scores and decay factors**.

4.2 Experiments

Word-sequence kernels generalise over previously introduced methods in several respects:

- For $n = 1$, individual terms are matched. This amounts to performing the usual cosine measure between document vectors (once the kernel is normalised). If symbol-dependent matching scores and decay factors are adopted, IDF based weighting schemes can be enforced;
- For $\lambda = 1$, gaps between symbols in a subsequence are not penalised, and the sequence kernel becomes similar to a polynomial kernel with $d = n$ in which unordered tuples of terms are replaced with ordered tuples; this suggests a way to test the importance of order in multi-word terms;

- For $\lambda_g = 0$, only contiguous sequences are matched, and the sequence kernel amounts to computing a similarity between documents based on the number of *n*-grams (more precisely on the number of 1-gram, 2-grams, ... up to *n*-grams) they share;
- With varying λ_g, the constraint on the contiguity of sequences can be relaxed, which suggests a way to test whether locality of multi-word terms (i.e. the fact that words of a sequence appear relatively close to each other) is an important factor or not.

As we see, word-sequence kernels define a paradigm from which it is possible to test the importance of different, basic factors for the document categorisation task. Assessing the validity of these factors, as well as the performance of word-sequence kernels, is the goal of the experiments we designed. More precisely, we want to answer the following questions:

1. Does the knowledge of the order in which the matched terms of a sequence occur improve the quality of the similarity measure? In other words, does a word-sequence kernel with $\lambda = 1$ perform better than the corresponding polynomial kernel?
2. Does the knowledge of the distance at which the matched terms of a sequence occur improve the quality of the similarity measure? That is, does a word-sequence kernel with $\lambda < 1$ perform better than the same one with $\lambda = 1$? Similarly, how does the version with $\lambda_g = 0$ (i.e. the *n*-gram version) compare to a more flexible word-sequence kernel in which the contiguity constraint is partially removed?
3. What relative importance should be given to individual words and to combination of words as indexing features? In other words, how does the parameter μ affect performance?
4. Do results improve if combination of more and more words are considered? How is performance affected by the value of *n*?
5. Is it actually useful to use symbol-dependent matching scores and decay factors? Related to this point is the questioning: is the word-sequence kernel sensitive to IDF based weighting schemes?
6. Finally, how does the word-sequence kernel compare to other kernels for text categorisation?

In order to answer these questions we performed a series of experiments using a standard benchmark for text categorisation systems: the Reuters-21578 corpus. We adopted the so-called "ModApte split", which leads to a training set of 9603 documents and a test set of 3299. Although computationally far more tractable than sequence kernels operating at the character level (note that the standard string kernels cannot be directly computed on this collection, and need be approximated, as mentioned in Lodhi et al. (2002)), word-sequence kernels are still somewhat resource-demanding.[2] We thus decided to limit our attention to the ten most frequent categories. Some characteristics of the composition of the training and test set relative to these categories are summarised in Table 2.

Documents were preprocessed by performing lemmatisation and resolving ambiguities by means of a part-of-speech tagger. Xerox tools were used to this effect. Lemmatisation maps different morphological variants of a same word (e.g. singular/plural forms of nouns, present/past tenses of verbs) onto the same feature (the *lexeme*). The underlying assumption is that the "inflected" form does not bring extra useful information compared to the "normalised" form. Notice however that many words are ambiguous (e.g. "saw") and determining the correct lemma (in our example, "saw" as noun, "saw" as verb or "see" as verb) for each is non-trivial. This is the reason why a part-of-speech (POS) tagger is used: the tagger model is able to choose (i.e. disambiguate) one POS category according to the word's context. After preprocessing, the number of features is about 53,000 and the average document length is 141 (when stopwords are removed, the average length drops to 77).

2. In the order of 10^3 kernel computations per second for sequences of this length and $n = 2$ on a Ultra SPARC II processor.

Category	# in training set	Neg/Pos ratio	# in test set
earn	2877	3	1087
acq	1650	6	719
money	538	18	179
grain	433	22	149
crude	389	25	189
trade	369	26	117
interest	347	28	131
ship	197	49	89
wheat	212	45	71
corn	181	53	56

Table 2: Number of positive examples in the ten most frequent categories of the Reuters-21578 corpus (ModApte split).

Experiments were conducted using the SVM^{light} (Joachims, 1999) package,[3] version 3.50, appropriately adapted for using sequence kernels.[4] Training sets for all categories are all more or less unbalanced in favour of negative examples. This is a well known problem with the SVM algorithms, which implicitly tends to maximise accuracy, thus leading to learning overly conservative classifiers. SVM^{light} provides a parameter, however, to weight the relative importance of positive and negative examples in the training set. As a heuristic rule, all experiments were run with this parameter set at the integer value closest to the ratio between negative and positive examples in the training set. While there is no evidence that this setting optimises any of the measures usually adopted in IR, this seemed a reasonable choice to at least reduce the impact of the lack of balance in the training set.

PERFORMANCE EVALUATION

In the following experimental results, we use standard IR performance measures. From the test categorisation results, we calculate the true positives TP (number of documents the model correctly identifies as positives), the false positives FP (number of documents the model falsely identifies as positives) and the false negatives FN (number of documents the model fails to identify as positives). From these counts, we calculate the following performance measures:

Precision: the ratio of true positives among all retrieved documents, $p = \frac{TP}{TP+FP}$.

Recall: the ratio of true positives over all positives, $r = \frac{TP}{TP+FN}$.

F-score: the harmonic mean of precision and recall, $F_1 = \frac{2pr}{p+r}$.

Note that these measures depend on the threshold that is applied to the decision function in order to decide whether a document is relevant. In order to provide a threshold-independent measure, we also compute the break-even point, which is the performance obtained at the threshold for which $p = r$. By definition, at the break-even point, precision, recall and F-score are equal.

In the remainder of this section, most experimental results are centered around a "reference" setting of $\mu = 0.5$, in which the weight of terms of size n is double that of terms of order $n-1$. We will see that this setting does not in general yield optimal performance. Note, however, that we study the impact of factors, such as order, locality or multi-term length, which control the way in which single-word terms interact with multi-word terms. In that context, using such a small value of μ gives more importance to longer terms, making this impact more visible.

3. Available at http://svmlight.joachims.org/
4. The adaptation mainly consisted in extending the processing of data structures different from the traditional one (vector), in modifying some I/O functions and in enhancing computing time performance by precomputing the Gram matrix only once for all categories.

	micro-average				macro-average			
	p	r	F_1	BP	p	r	F_1	BP
Stopwords not removed	91.98	87.62	89.75	90.21	85.20	74.30	78.92	80.86
Stopwords removed	**93.01**	**88.09**	**90.52**	**91.46**	**85.37**	**76.71**	**80.64**	**83.09**

Table 3: The effect of removing stopwords on Precision (p), Recall (r), F_1 score and Break-even Point (BP). These results are obtained with $n = 2$, $\lambda = 0.5$ and $\mu = 0.5$.

The performance measures presented above are calculated for each category. In order to present a synthetic measure of performance over all categories, we present the micro-averaged and macro-averaged performance. Macro-averaging consists in simply averaging the results obtained on each category, while micro-averaging averages over individual decisions made on each document for each category. In effect, micro-averaging is dominated by the performance of "large" categories, while macro-averaging gives equal influence to all categories. In all our experiments, both methods give consistent results. However, as micro-averaging is dominated by the large categories which are relatively easy to learn, the effect of different parameter settings is usually more visible on the macro-averaged performance.

STOPWORD REMOVAL

Stopword removal is a common practice in document categorisation and filtering. In the context of word-sequence kernels, it has the favourable side effect of reducing the average sequence length by about 50%, and correspondingly reducing kernel computation time by 75%.

Preliminary to any further experiment, we verified whether stopword removal would be beneficial in the case of word-sequence kernels as well. The results in Table 3 show that performance improves significantly for all performance measures after stopword removal. Accordingly, all subsequent experiments were performed on sequences from which stopwords had been removed.

ORDER

Word-sequence kernels of length 2 use implicit features which are the ordered combinations of two words. The quadratic kernel on the bag-of-word representation uses the same kind of features, but does not take word order into account. Using word order allows to differentiate between documents containing the same words in different orders, potentially yielding a higher precision. On the other hand, when the order information is not considered, recall may increase because such documents are considered similar.

The aim of the first experiment is therefore to evaluate the impact of the order of word combinations on the accuracy of the similarity measure. To that effect, we compared the performance of the word-sequence kernel defined in Equation 14 with $n = 2$, $\lambda = 1$ and $\mu = 0.5$ with that of the polynomial kernel of degree $d = 2$. In both cases, the SVM margin parameter C was set to the default $1/\text{avg}(K(x,x))$. In order to ensure that only the effect of order is being measured, we used the same preprocessing for both kernels, ie lemmatisation and stopword removal. In addition, in order to ensure that single terms and multiple terms were given the same relative influence in the similarity, we computed a normalised version of the quadratic kernel, using the same normalisation as Equations 13 and 14:

$$\bar{K}_{p2}(x,y) = \frac{\langle x,y\rangle}{\sqrt{\langle x,x\rangle\langle y,y\rangle}} + \frac{1}{\mu}\frac{\langle x,y\rangle^2}{\langle x,x\rangle\langle y,y\rangle} \tag{19}$$

The first three rows of Table 4 display the results obtained without IDF in the polynomial kernels, and using a fixed $\lambda = 1$ in the word-sequence kernel. The word-sequence kernel performs slightly worse than the normalised polynomial, by a small but consistent margin, indicating that taking into account the word order does not help in the categorisation task. The results obtained by the standard quadratic kernel $K(x,y) = (\langle x,y\rangle + 1)^2$ are largely inferior to those of the other two kernels. This is because when no IDF is applied, the un-normalised quadratic kernel is dominated by the products of frequencies of very common words, which tend to be poor discriminators.

	micro-average				macro-average			
	p	r	F_1	BP	p	r	F_1	BP
WSK, $n=2$, $\lambda=1$, $\mu=\frac{1}{2}$	93.49	88.09	90.71	91.25	86.26	**76.86**	81.11	82.35
$\bar{K}_{p2}$,	**93.73**	**88.55**	**91.06**	**91.49**	**87.36**	76.78	**81.44**	**83.44**
Polynomial, $d=2$	81.45	84.20	82.80	87.00	78.50	70.50	73.65	76.72
WSK, $n=2$, varλ_m, $\lambda_g=1$	**95.37**	84.97	89.87	91.64	**90.71**	72.00	79.99	84.39
$\bar{K}_{p2}$, IDF	94.97	85.46	89.96	91.56	90.23	72.90	80.36	84.34
Polynomial, $d=2$, IDF	93.91	**88.08**	**90.90**	**91.74**	89.03	**77.29**	**82.52**	**84.82**

Table 4: The impact of taking word order into account on Precision (p), Recall (r), F_1 score and Break-even Point (BP). We compare word-sequence kernels (WSK), normalised polynomial (K_{p2}) and standard polynomial kernel, with and without IDF.

	micro-average				macro-average			
	p	r	F_1	BP	p	r	F_1	BP
var λ_m, $\lambda_g=0$	95.45	85.07	89.96	91.60	91.01	71.92	79.98	84.23
var λ_m, $\lambda_g=0.5$	95.45	84.97	89.90	91.57	91.12	71.95	80.05	84.17
var λ_m, $\lambda_g=1$	95.37	84.97	89.87	91.64	90.77	72.00	79.99	84.39
var λ_m, $\lambda_g=1-\lambda_m$	**95.66**	**85.32**	**90.20**	**91.75**	**91.71**	**72.30**	**80.45**	**84.50**

Table 5: The impact of locality on Precision (p), Recall (r), F_1 score and Break-even Point (BP). We compare the performance obtained for symbol-dependant decay factors with $\lambda_g=0$ (no gaps allowed), $\lambda_g=0.5$, $\lambda_g=1$ (any gap allowed) and $\lambda_g=1-\lambda_m$ (IDF-dependent). All results are for word-sequence kernels with $n=2$ and $\mu=0.5$.

The last three rows of Table 4 display the results obtained with IDF (eq. 15) in the polynomial kernels, and with corresponding variable λ_m (eq. 16) in the word-sequence kernel. By taking word order into account, the word-sequence kernel yields a better precision, but this does not offset the large gain in recall observed for the standard polynomial kernel, which now performs best overall. These results suggest that taking word order into account does not benefit categorisation performance.

LOCALITY

The decay factor λ allows to control the extent to which "gaps" are allowed in matching subsequences. For $\lambda=1$, gaps will have no effect on the sequence similarity, while for $\lambda\to 0$, any gap within a subsequence will drive the corresponding feature value (and hence the similarity) towards 0 (cf. eqs. 4 and 5).

The aim of our second experiment is therefore to check the importance of taking into account the distance at which matched words occur, ie whether matches should be local (small λ) or whether they may occur accross the entire document (large λ). Note that for symbol-dependent decay factors, $\lambda_g=0$ corresponds to a pure n-gram model, while $\lambda_g=1$ is similar to polynomial kernels (with an additional order constraint).

For fixed λ, Figure 3 shows that the setting of λ has little effect on the performance, suggesting that locality is not overly important for our categorisation task. Table 5 displays similar results for the symbol-dependent decay factors and different settings for λ_g.

All results suggest that locality does not seem to have a strong impact on our categorisation task. This does not mean, however, that the setting of λ is irrelevant. Indeed, using symbol-dependent decay factors for both gaps and matches (last line in Table 5) yields the overall best performance, with a small but consistent edge over all alternatives presented here.

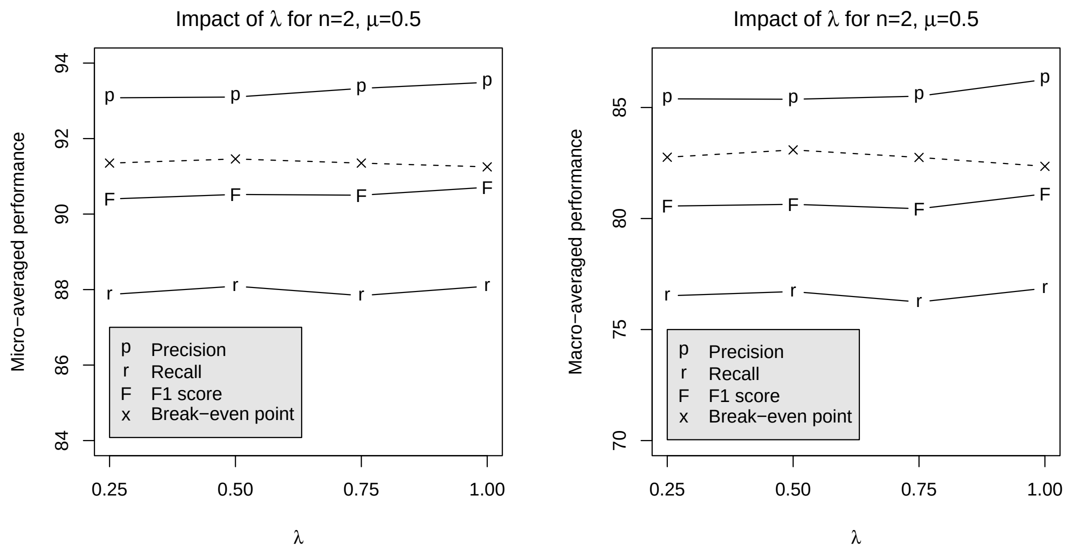

Figure 3: The effect of locality (varying decay factor λ) on the performance. All experiments are with $n = 2$ and $\mu = 0.5$. The decay factor and hence the number of gaps within a matching subsequence seems to have little impact on performance.

	micro-average				macro-average			
	p	r	F_1	BP	p	r	F_1	BP
$\mu = 0.5$, var λ_m	**95.45**	84.97	89.90	**91.57**	**91.12**	71.95	80.05	**84.17**
$\mu = 2$, var λ_m	94.28	**88.16**	**91.12**	91.50	89.71	**77.27**	**82.78**	83.87

Table 6: The impact of μ on Precision (p), Recall (r), F_1 score and Break-even Point (BP). We compare the performance obtained for symbol-dependent decay factors with $\mu = 0.5$ (more weight to multi-word matches) and $\mu = 2$ (more weight to single-word matches).

INFLUENCE OF μ

Parameter μ gives the relative weight of multi-word terms compared to single words. In our experimental results, we focus on $\mu = 0.5$, which for $n = 2$ corresponds to giving double weight to matches on two words compared to single-word matches. As mentionned earlier, this allows to emphasize the effect of multi-word matches, but is not necessarily optimal. Indeed, some results in IR suggest that it usually helps to give more weight to single terms with respect to multi-word terms (Gaussier et al., 2000).

The aim of the third experiment is to check the impact of μ on the performance. Figure 4 shows that as μ increases (i.e. the influence of multi-word matches decreases), precision decreases. This is because matches on several words are a more certain indicator of similarity than matches on single words. On the other hand, forcing the similarity to consider mostly multi-word matches (for small μ), one may fail to identify as similar two related documents if they only share single words, hence a lower recall. Figure 4 shows that indeed recall increases greatly with larger μ. This more than compensates for the loss in precision and the overall F-score increases. The same effect is observed for symbol-dependent λ, as illustrated in Table 6. Despite a clear and consistent impact of μ on precision and recall for both the fixed λ and variable λ cases, the break-even point seems relatively insensitive to this parameter. Note also that larger values of μ seem to favour the F-score on this categorisation task, a result that is consistent with findings in IR (Gaussier et al., 2000).

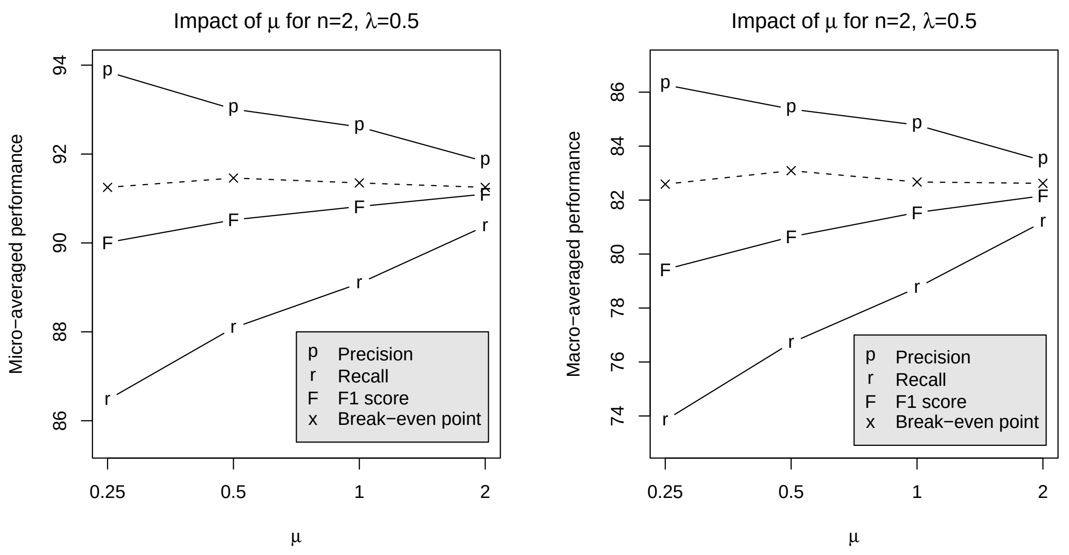

Figure 4: The effect of varying the relative weight of multi-terms μ on the performance. All experiments are with $n = 2$ and $\lambda = 0.5$.

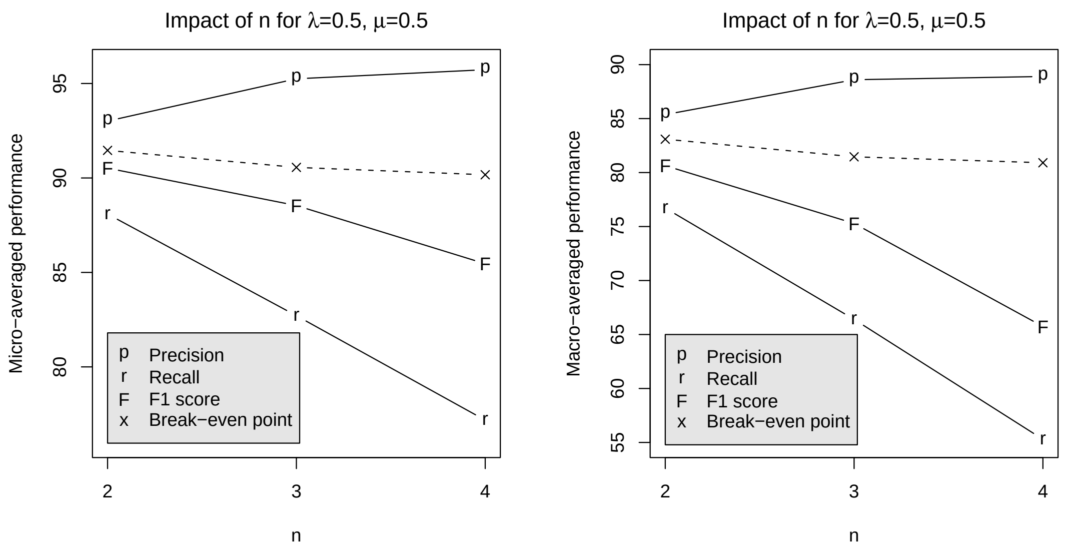

Figure 5: The effect of varying the subsequence length n on the performance. All experiments are with $\mu = 0.5$ and $\lambda_m = \lambda_g = 0.5$.

INFLUENCE OF n

All previous experiments considered only single words and word pairs ($n = 2$). We also performed experiments to assess the impact of n on performance. Results are displayed in Figure 5. When longer multi-terms are taken into account precision improves. However, the corresponding loss in recall more than compensates this increase, so that both the F1 score and the breakeven point decrease.

INFLUENCE OF SYMBOL-DEPENDENT MATCHING SCORES AND DECAY FACTORS

All experiments show that using symbol-dependent λ's for matching symbols has a positive influence on performance. As mentioned in Section 4.1, symbol-dependent λ's allow us to make use of an IDF-based weighting scheme. The fact that this weigthing scheme improves the results comes as no surprise as the importance of IDF based term weighting is well known from the IR literature.

We observe, however, that word-sequence kernels are less sensitive to term weighting than quadratic kernels. Table 4 shows that the performance of the kernel improves when an IDF based weighting scheme is used (fourth line), in particular for the macro-average, but the improvement remains marginal for the micro-average. On the contrary, using IDF yields a significant improvement for the quadratic kernel, both on the micro- and macro-average: the break-even point increases from 87% to 91.74% in micro-average, and from 76.72% to 84.82% in macro-average (Table 4, lines 3 and 6).

COMPARISON OF WORD-SEQUENCE KERNELS WITH OTHER KERNELS

The best performance obtained with word-sequence kernels of length 2 (last line of Table 5) are comparable to the best performance obtained with a polynomial kernel of degree 2 (last line of Table 4). In addition, our results are comparable with state-of-the-art results on this collection using SVM and various bag-of-word kernels.[5] Experimental results also show that word-sequence kernels perform better than string kernels on the complete set of documents, when string kernels must be approximated (Lodhi et al., 2002).

Compared to polynomial and RBF kernels, word-sequence kernels have the disadvantage of being computationally more demanding. However, as we have seen, word-sequence kernels allowed us to test the importance of different factors for text categorisation, an approach which was not possible before. Lastly, word-sequence kernels rely on several parameters which have been fixed in our experiments, but can in theory be optimised for a specific collection. We will investigate such a tuning in future work.

EXPERIMENTAL RESULTS SUMMARY

In summary, our experiments seem to show the following:

1. Taking word order into account (Table 4) has very little effect on performance. When IDF is applied precision is slightly increased and recall is slightly reduced;
2. Locality *per se* (Figure 3 and Table 5) has virtually no impact. Some improvement can be obtained by penalising more heavily those non-contiguous word combinations which are obtained by skipping terms with low document frequency;
3. Giving more relative importance to word combinations in the similarity measure increases precision at the expenses of recall (Figure 4 and Table 6);
4. Similarly, considering the combination of more words as indexing terms increases precision at the expenses of recall (Figure 5);
5. Weighting matches according to IDF-based λ_m's improves performance;
6. Finally, the results obtained with word-sequence kernels are comparable to state-of-the-art results on the same collection.

5. New Directions in Word-Sequence Kernels

In the previous section, we defined and illustrated the use of word-sequence kernels, and their extension to symbol-dependent decay factors. We introduce here two additional natural extensions of word-sequence

5. On the same 10 categories, Joachims (1998) reported micro-averaged break-even points of: 89.85 for $d = 1$ (linear), 91.13 for $d = 2$ (quadratic), 91.89 for $d = 3$, 92.06 for $d = 4$, 92.00 for $d = 5$ and 92.29 for the RBF kernel.

kernels. These extensions address two linguistically motivated problems: estimating the similarity between documents containing different words with similar meanings (synonyms), or between documents in different languages (cross-lingual).

5.1 Soft Matching

In the standard definition of word-sequence kernels (Sections 2.2 and 4), only exact symbol matches contribute to the similarity. One shortcoming of this approach is that synonyms or words with related meanings will never be considered similar. We address this problem by considering soft-matching of symbols, ie matching *equivalent* as well as identical symbols, ensuring that matching equivalent symbols contributes less to the similarity than an exact match.

Several IR techniques implement some kind of soft-matching. For example, the Generalised Vector Space Model (GVSM, cf. Wong et al., 1985, Sheridan and Ballerini, 1996) uses the document-term matrix to estimate a term-term similarity on the basis of co-occurrences of terms in documents. Similarly, in the context of sequence kernels, we can define a soft-matching extension by invoking a similarity matrix in the implicit feature space:

$$K_{\text{soft}}(s,t) = \phi(s)^\top \mathbf{A}\phi(t) = \sum_{u,v} \phi_u(s)\,\phi_v(t)\,A_{uv} \tag{20}$$

Setting $A_{uv} = 1$ iff $u = v$, we recover the original formulation. Equation 20 expresses a valid kernel as long as the similarity matrix $\mathbf{A} = [A_{uv}]$ is positive definite. Note that $\mathbf{A} \in \left(\mathbb{R}_0^+\right)^{\Sigma^n \times \Sigma^n}$ is indexed on the feature space dimensions, ie ordered subsequences of length n. In order to simplify the processing and be able to calculate the kernel value without feature space expansion, it may be convenient to express the similarity at subsequence level as a product of similarities at the symbol level: $A_{uv} = \prod_{k=1}^{n} a_{u_k v_k}$. In that case, since $\mathbf{A}$ is the n-fold tensor product of the *symbol* similarity matrix $\mathbf{a} = [a_{xy}] \in \left(\mathbb{R}_0^+\right)^{\Sigma \times \Sigma}$, $\mathbf{A}$ is positive definite whenever $\mathbf{a}$ is. The soft-matching word-sequence kernel may be calculated recursively using Equations 6 to 7, 10 and 11, replacing Equations 8 and 9 by the single equation:

$$K_i''(sx,ty) = \lambda K_i''(sx,t) + \lambda^2 a_{xy} K_{i-1}'(s,t), \quad \forall x,y,$$

and modifying Equation 12 to:

$$K_n(sx,t) = K_n(s,t) + \sum_{j}^{|t|} \lambda^2 a_{xt_j} K_{n-1}'(s,t[1:j-1])$$

Soft matching can be combined with the use of distinct decay factors for gaps and matches:

$$\begin{aligned} K_{\text{ds}n}(s,t) &= \hat{\phi}(s)^\top \mathbf{A}\hat{\phi}(t) \\ &= \sum_{u\in\Sigma^n}\sum_{v\in\Sigma^n}\sum_{\mathbf{i}:u=s[\mathbf{i}]}\sum_{\mathbf{j}:v=t[\mathbf{j}]} \lambda_m^{2n} \prod_{k=1}^{n} a_{u_k,v_k} \prod_{i_1<l<i_n, l\notin\mathbf{i}} \lambda_{g,s_l} \prod_{j_1<p<j_n, p\notin\mathbf{j}} \lambda_{g,t_p} \end{aligned} \tag{21}$$

and the recursion formulas can be adapted accordingly.

5.2 Cross-Lingual Document Similarity

Once soft matching between different symbols is introduced, it is actually possible to define a similarity between sequences from different alphabets. In the context of word-sequence kernels, this means that words may be from different languages. Indeed, the kernel $K_{\text{soft}}(s,t)$ from (21) can be directly applied, with a symbol similarity matrix encoding weighted translation between words in the two languages, derived from bilingual dictionaries or corpora. Several vector models developed for monolingual IR have been extended to the multi-lingual case using a parallel corpus. We will now see how the word-sequence kernel relates to two models previously proposed for cross-language IR, namely the Generalised Vector Space Model (GVSM) and Latent Semantic Indexing (LSI).

Let Σ_1 and Σ_2 be two alphabets corresponding to the dictionaries of the two languages under consideration ($|\Sigma_1| = m$, $|\Sigma_2| = q$, $\Sigma = \Sigma_1 \cup \Sigma_2$). As a "document" in the parallel collection is actually a pair of aligned documents, we will designate it as a *p-document* for clarity. In the following, we decompose the p-document-term matrix $\mathbf{D}$ into language specific parts $\mathbf{D_1}$ and $\mathbf{D_2}$, such that $\mathbf{D} = [\mathbf{D_1}\ \mathbf{D_2}]$.

5.2.1 GENERALISED VECTOR SPACE MODEL

The use of the GVSM in cross-language IR amounts to substitute the original documents with their projection into a dual space, the dimensions of which are induced from clusters of p-documents. In the case where each such cluster contains only one p-document, and using the standard cosine measure, one gets:

$$K_{\text{GVSM}}(d,d') = \cos(\mathbf{D_1}d, \mathbf{D_2}d') = \frac{d^\top \mathbf{D_1}^\top \mathbf{D_2}\, d'}{\|\mathbf{D_1}d\|\ \|\mathbf{D_2}d'\|} \tag{22}$$

with d in language 1 and d' in language 2. Assuming that raw frequencies are used as term frequencies in the document representation, we can expand the numerator of (22) in the following way:

$$d^\top \mathbf{D_1}^\top \mathbf{D_2}\, d' = \sum_{u\in\Sigma_1} \sum_{v\in\Sigma_2} \sum_{s:d[s]=u} \sum_{t:d'[t]=v} (\mathbf{D1}^\top \mathbf{D2})_{uv}$$

where $d[s]$ corresponds to the term occurring at position s in d (and similarly for $d[t]$). Lastly, considering the $(m+q)$ square matrix $A = \mathbf{D}^\top \mathbf{D} = \begin{bmatrix} \mathbf{D1}^\top \mathbf{D1} & \mathbf{D1}^\top \mathbf{D2} \\ \mathbf{D2}^\top \mathbf{D1} & \mathbf{D2}^\top \mathbf{D2} \end{bmatrix}$, we obtain the following symmetric expression:

$$d^\top \mathbf{D_1}^\top \mathbf{D_2}\, d' = \sum_{u\in\Sigma} \sum_{v\in\Sigma} \sum_{s:d[s]=u} \sum_{t:d'[t]=v} (A)_{uv} \tag{23}$$

which corresponds to the soft-matching word-sequence kernel (eq. 21) with $n = 1$ and $\lambda_m^2 a_{uv} = (A)_{uv}$. The complete cosine similarity is then obtained through standard length normalisation.

5.2.2 CROSS-LINGUAL LSI

The cross-language Latent Semantic Indexing model (Dumais et al. (1996)) is similar to the monolingual setting (Deerwester et al. (1990)), except that it is based on the singular value decomposition of the combined p-document-term matrix $\mathbf{D}$. Assuming that we select the dimensions with the highest k singular values, the similarity is calculated after projecting documents using $\mathbf{U_k}$, the $(m+q)\times k$ matrix containing the first k columns of $\mathbf{U}$. Accordingly, the cosine similarity measure for cross-language LSI is given by:

$$K_{\text{LSI}}(d,d') = \cos(\mathbf{U_k}^\top d, \mathbf{U_k}^\top d') \tag{24}$$

where documents d and d' are appropriately zero-padded into $(m+q)$ dimensional vectors. In a way similar to how we derived a word sequence kernel for GVSM from Equation 22, we can derive a word-sequence kernel with $n = 1$ for cross-language LSI, provided once again that raw frequencies are used as term frequencies. In this case, the similarity coefficients are $\lambda_m^2 a_{uv} = (\mathbf{U_k}\ \mathbf{U_k}^\top)_{uv}$. Once again, the complete equivalence with the cosine measure relies on length normalisation.

Note that, to be used in cross-language IR, both GVSM and LSI need a parallel corpus, which is not the case for word-sequence kernels, which can be computed based on a bilingual lexicon derived from a comparable corpus for example.

6. Conclusions

Kernel methods allow the use of efficient learning algorithms in cases where the document representation is not necessarily a vector. For example, the sequence kernels (string kernel, syllable kernel, word-sequence kernel) represent one of the first competitive alternatives to the traditional bag-of-words representation.

In this paper we first tried to analyse why sequence kernels operating at the character level, with no higher level information, have been so successful. We formulated the *noisy stemming* hypothesis, according to which sequence kernels perform well because they implicitly select relevant word stems as indexing units. We showed that this hypothesis is supported by empirical observations. Moreover, our observations suggest that the subsequences which are more discriminant are essentially distributed on more than one word stem, and therefore contribute to the effectiveness of the categorisation by indexing pairs of consecutive words.

In this paper we focussed on the word-sequence kernels, where documents are considered as sequences of words. One advantage over the string kernel is that we use more semantically meaningful indexing units. In addition, word-sequence kernels are significantly less computationally demanding. Finally, they lend themselves to a number of natural extensions. We described the use of symbol-dependent decay factors and independent decay factors for gaps and matches in the context of word-sequence kernels. This allows the use of linguistically motivated background knowledge, and of traditional weighting schemes. We showed that the use of independent IDF-based decay factors λ_m and λ_g yields better performance than fixed λ.

The flexibility of word-sequence kernels allowed us to test a number of hypotheses regarding the impact of word order, locality or multi-term matches on the categorisation performance. Our results suggest that order and locality have essentially no effect on the performance, while the length of multi-term matches and their weight in the similarity measure has a clear influence on the precision, recall and F-score.

Finally, we envisioned the possibility of using soft-matching of symbols in order to take into account a similarity between different but related words. This opens up the possibility of using word-sequence kernels in the context of multi-lingual document processing, and apply kernel methods to documents from different languages.

Word-sequence kernels are a particular case of *convolution kernels*, and our work contributes to the investigation of document representations that are more structured than the bag-of-words model, in the context of IR and document categorisation.

7. Acknowledgements

The work presented in this paper was sponsored by the European Commission through the IST Programme under Contract IST-2000-25431. We are extremely grateful to our partners from the EU KerMIT (Kernel Methods for Images and Text) project for discussing these topics with us. In particular we wish to thank John Shawe-Taylor for many comments and suggestions for possible extensions of the word-sequence kernel, and Craig Saunders for help with the string kernel experiments. We also acknowledge many insightful comments from the anonymous reviewers.

Appendix A. Recursive Formulation of Sequence Kernels

This appendix gives an intuitive description of the basic sequence kernel, as well as a derivation of an efficient recursive formulation based on dynamic programming.

We first recall some basic notation and definitions:

Let Σ be a finite alphabet, and let $s = s_1 s_2 \cdots s_{|s|}$ be a sequence over such alphabet (i.e. $s_i \in \Sigma, 1 \leq i \leq |s|$). Let $\mathbf{i} = [i_1, i_2, ..., i_n]$, with $1 \leq i_1 < i_2 < ... < i_n \leq |s|$, be a subset of the indices in s: we will indicate as $s[\mathbf{i}] \in \Sigma^n$ the subsequence $s_{i_1} s_{i_2} \ldots s_{i_n}$. Let us write $l(\mathbf{i})$ the value $i_n - i_1 + 1$, i.e. the length of the window in s spanned by $s[\mathbf{i}]$.

The basic Sequence Kernel works in the feature space of all possible subsequences of length n, where the value associated with the feature u is defined by:

$$\phi_u(s) = \sum_{\mathbf{i}:u=s[\mathbf{i}]} \lambda^{l(\mathbf{i})} \tag{25}$$

where $\lambda \in]0;1]$ is a decay factor used to penalise non-contiguous subsequences. Actually, as extensively explained and exploited in Section 4.1.2, this decay factor is applied not only to the "gaps", but also to the matching symbols. For instance, if λ is given the value 1, the gaps are not taken into account when computing the value of the feature $\phi_u(s)$. When λ is given the value 0.5, each gap symbol contributes to dividing the feature value by 2 (without gap, $\phi_u(s) = (0.5)^n$).

The sequence kernel of two strings s and t over Σ is defined as the inner product:

$$K_n(s,t) = \sum_{u \in \Sigma^n} \phi_u(s) \cdot \phi_u(t) = \sum_{u \in \Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \lambda^{l(\mathbf{i})} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{j})} = \sum_{u \in \Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{i})+l(\mathbf{j})} \tag{26}$$

Intuitively, this means that we match all possible subsequences of n symbols, with each occurrence "discounted" according to the size of the window that it spans. Consider for example the alphabet $\Sigma = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$, and the two elementary sequences:

$$s = CATG \qquad \text{and} \qquad t = ACATT$$

For $n = 3$, the weights of all subsequences (or features) for sequences s and t are, according to (25):

u	$\phi_u(s)$	$\phi_u(t)$	u	$\phi_u(s)$	$\phi_u(t)$
AAT	0	$\lambda^4+\lambda^5$	CAG	λ^4	0
ACA	0	λ^3	CAT	λ^3	$\lambda^3+\lambda^4$
ACT	0	$\lambda^4+\lambda^5$	CTG	λ^4	0
ATG	λ^3	0	CTT	0	λ^4
ATT	0	$\lambda^3+\lambda^5$	others	0	0

where for instance the value of the feature $u =$AAT for sequence $t =$ACATT is $\lambda^4 + \lambda^5$ because there are two occurrences of AAT in ACATT. The first spans a window of width four (first, third and fourth symbols) and the second spans a window of width five (first, third and fifth symbols). The similarity score is then:

$$K_3(CATG, ACATT) = \langle \phi(s), \phi(t) \rangle = \sum_u \phi_u(s)\phi_u(t) = \lambda^3(\lambda^3 + \lambda^4)$$

as the only feature for which both sequences have a non-null value is CAT.

A direct computation of all the terms under the nested sum in (26) becomes impractical even for small values of n. In addition, in the spirit of the kernel trick discussed in Section 2, we wish to calculate $K_n(s,t)$ directly rather than to perform an explicit expansion in feature space. This can be done using a recursive formulation proposed by Lodhi et al. (2001), which leads to a more efficient dynamic-programming implementation. The recursive formulation is based on the following reasoning. Suppose that we already know the value of the kernel for two strings s and t: what do we need to compute the value of the kernel for sx and t, for some $x \in \Sigma$? Notice that:

1. All subsequences common to s and t are also common to sx and t.
2. In addition, we must consider all new matching subsequences ending in x which occur in t and whose (n-1)-symbol prefix occur in s (possibly non-contiguously).

For the latter point, consider the example in Figure 6. Let u' and u'' be two distinct sequences of length $n-1$. They occur in both s and t and, as occurrences can contain gaps, each occurrence spans, in principle, a window of different length. Let's focus, for instance, on the occurrence of u'' in s marked as a. If $\mathbf{i}^a$ is the set of indices of the occurrence, i.e. if $u'' = s[\mathbf{i}^a]$, then the length of the window spanned by the occurrence is $i^a_{n-1} - i^a_1 + 1$. The occurrence a will give raise to two new matches of length n for the subsequence $u''x$ between sx and t, due to the presence of the occurrence of u'' marked as b in t and to the two occurrences of x to the right of b in t. These two matches will contribute to the kernel according to their lengths:

$$\lambda^2(\lambda^{|s|-i^a_1+1}\lambda^{j_{l1}-j^b_1} + \lambda^{|s|-i^a_1+1}\lambda^{j_{l2}-j^b_1})$$

where $\mathbf{j}^b$ is the set of indices of b in t, and j_{l1} and j_{l2} are the indices of the relevant occurrences of x in t, as indicated in the figure. Note the λ^2 factor, which is the contribution of the matching x's themselves, the rest

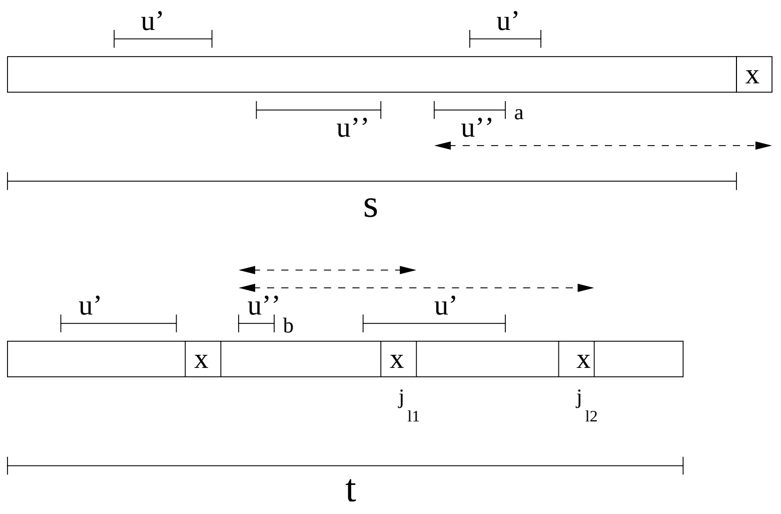

Figure 6: Contributions from different occurrences of common subsequences in two strings s and t.

being the contribution of the gaps to the kernel. Similar inputs will be given by all occurrences of u', u'' and all other subsequences of length $n-1$ in the two strings.

We thus rewrite the kernel for sx and t as:

$$\begin{aligned} K_n(sx,t) &= K_n(s,t) + \sum_{u\in\Sigma^{n-1}} \sum_{\mathbf{i}:s[\mathbf{i}]=u} \sum_{j:t_j=x} \sum_{\mathbf{l}:t[\mathbf{l}]=u,l_{n-1}<j} \lambda^{|s|+1-i_1+1}\lambda^{j-l_1+1} \\ &= K_n(s,t) + \sum_{j:t_j=x} \lambda^2 \sum_{u\in\Sigma^{n-1}} \sum_{\mathbf{i}:s[\mathbf{i}]=u} \sum_{\mathbf{l}:t[\mathbf{l}]=u,l_{n-1}<j} \lambda^{|s|-i_1+1}\lambda^{j-1-l_1+1} \end{aligned} \qquad (27)$$

Notice that the part of the second term within the three innermost sums looks quite similar to the definition of the kernel for sequences of length $n-1$, although the contribution decays over $|s|-i_1+1$ and $j-1+l_1+1$ rather than $i_{n-1}-i_1+1$ and $l_{n-1}-l_1+1$ as in (26). Defining:

$$K'_{n-1}(s,t) = \sum_{u\in\Sigma^{n-1}} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{|s|-i_1+1}\lambda^{|t|-j_1+1}$$

Equation 27 can be rewritten as:

$$K_n(sx,t) = K_n(s,t) + \sum_{j:t_j=x} \lambda^2 K'_{n-1}(s,t[1:j-1])$$

where $t[1:j-1]$ refers to the first $j-1$ symbols of t. Intuitively, $K'_{n-1}(s,t)$ counts matching subsequences of $n-1$ symbols, but instead of discounting them according to the length of the window they span as in $K_{n-1}(s,t)$, it discounts them according to the distance from the first symbol in the subsequence to the end of the complete sequence. In the example used above we have, for $K'_2(s,t)$:

u	s	t	u	s	t
AA	0	λ^5	GA	0	0
AC	0	λ^5	GC	0	0
AG	λ^3	0	GG	0	0
AT	λ^3	$2\lambda^3+2\lambda^5$	GT	0	0
CA	λ^4	λ^4	TA	0	0
CC	0	0	TC	0	0
CG	λ^4	0	TG	λ^2	0
CT	λ^4	$2\lambda^4$	TT	0	λ^2

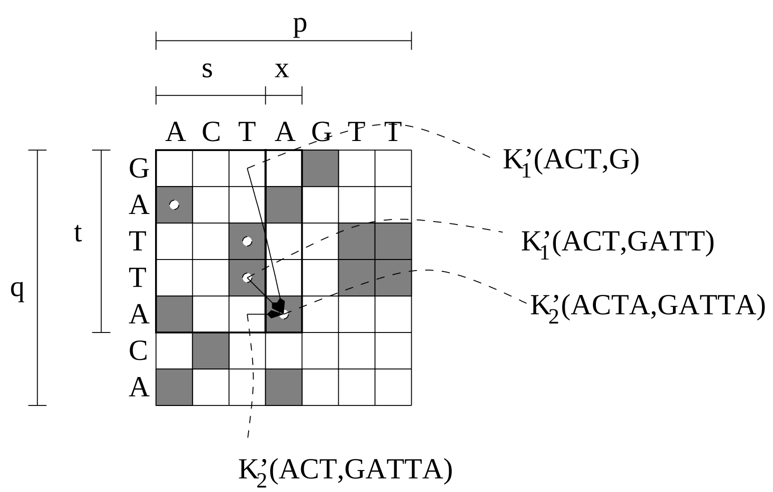

Figure 7: Illustration of the recursion to calculate the similarity between GATTACA and ACTAGTT.

where the value for the feature u =CT for sequence t=ACATT is $2\lambda^4$ because both occurrences of CT start on the second symbol C which is 4 symbols away from the end of t. Hence:

$$K_2'(\text{CATG},\text{ACATT}) = \lambda^3(2\lambda^3+2\lambda^5)+\lambda^4\lambda^4+\lambda^4(2\lambda^4) = 2\lambda^6+5\lambda^8$$

The values of K_n' can be calculated recursively as:

$$\begin{aligned} K_n'(sx,t) &= \sum_{u\in\Sigma^n}\sum_{\mathbf{i}:u=s[\mathbf{i}]}\sum_{\mathbf{j}:u=t[\mathbf{j}]}\lambda^{(|s|+1)-i_1+1}\lambda^{|t|-j_1+1} \\ &\quad + \sum_{v\in\Sigma^{n-1}}\sum_{\mathbf{i}:v=s[\mathbf{i}]}\sum_{j:t_j=x}\sum_{\mathbf{j}:t[\mathbf{j}]=v,j_{i-1}<j}\lambda^{(|s|+1)-i_1+1}\lambda^{|t|-j_1+1} \\ &= \lambda\sum_{u\in\Sigma^n}\sum_{\mathbf{i}:u=s[\mathbf{i}]}\sum_{\mathbf{j}:u=t[\mathbf{j}]}\lambda^{|s|-i_1+1}\lambda^{|t|-j_1+1} \\ &\quad + \sum_{j:t_j=x}\sum_{v\in\Sigma^{n-1}}\sum_{\mathbf{i}:v=s[\mathbf{i}]}\sum_{\mathbf{j}:t[\mathbf{j}]=v,j_{n-1}<j}\lambda^{|s|-i_1+1}\lambda^{(j-1)-j_1+1}\lambda^{|t|-j+2} \\ &= \lambda K_n'(s,t)+\sum_{j:t_j=x}K_{n-1}'(s,t[1:j-1])\lambda^{|t|-j+2} \qquad (28) \end{aligned}$$

For some additional insight, an example can be useful (Figure 7). Given two sequences p and q, a binary matrix M can be defined such that:

$$M_{ij} = \begin{cases} 1 & \text{if } p_i{=}q_j \\ 0 & \text{otherwise} \end{cases}$$

The value $K_n'(sx,t)$ ($K_2'(\text{ACTA},\text{GATTA})$ in the figure) counts the number of matches of sequences of length n (of length 2 in the figure) appropriately discounted from the first element of the match to the end of the two sequences. In the example in the figure, two 2-matches are taken into account by the term $K_2'(\text{ACT},\text{GATTA})$ (one given by $\langle p_1=q_2=A\ ,\ p_3=q_3=T\rangle$, a second given by $\langle p_1=q_2=A\ ,\ p_3=q_4=T\rangle$), appropriately discounted by λ for the additional distance to the end of the sequence sx caused by the final $x=A$. In addition, three more 2-matches, appropriately discounted as well, are taken into account by the term $K_1'(\text{ACT},\text{GATT})$ (one given by $\langle p_3=q_3=T\ ,\ p_4=q_5=A\rangle$, a second by $\langle p_3=q_4=T\ ,\ p_4=q_5=A\rangle$ and a third by $\langle p_1=q_2=A\ ,\ p_4=q_5=A\rangle$). Notice that the contribution of the term $K_1'(\text{ACT},\text{G})$ is null, as none of the symbols in ACT matches the symbol G.

Intuitively, $K_{n-1}'(s,t)$ is used by the algorithm to store, as an intermediate result, the total discounted "mass" of matches of length $n-1$ ready to be turned into matches of length n should the next symbol in

s match some of the symbols in t. This "mass" is propagated according to the recursive formulation in Equation 28. In terms of the matrix in Figure 7, this means that values of K'_n are percolated from left to right along the rows, discounting them by an additional λ for each new step. Moreover, if the position at which the value is computed corresponds itself to a symbol match, then the value is accrued by the masses of sequences of length $n-1$ stored in the immediately previous column and in the rows from 1 to one less than the current position.

To summarise, the recursive formulation inclusive of the base steps is given by:

$$\begin{aligned}
K'_0(s,t) &= 1, \text{ for all } s,t, \\
K'_i(s,t) &= 0, \text{ if } \min(|s|,|t|) < i, \quad (i = 1,\dots,n-1) \\
K_n(s,t) &= 0, \text{ if } \min(|s|,|t|) < n, \\
K'_i(sx,t) &= \lambda K'_i(s,t) + \sum_{j:t_j=x} K'_{i-1}(s,t[1:j-1])\lambda^{|t|-j+2} \quad (i = 1,\dots,n-1) \\
K_n(sx,t) &= K_n(s,t) + \sum_{j:t_j=x} \lambda^2 K'_{n-1}(s,t[1:j-1]),
\end{aligned}$$

The time required to compute the kernel according to this formulation is $O(n|s||t|^2)$. This can be seen by observing that the outermost recursion is for increasing values of subsequence lengths ($i = 1,\dots,n-1$), and that for each length and each additional symbol in s and in t a sum over the whole prefix of t up to the position being considered is required. Notice however that complexity can be reduced by storing intermediate values of the latter sum. We can define the additional function:

$$K''_i(sx,t) = \sum_{j:t_j=x} K'_{i-1}(s,t[1:j-1])\lambda^{|t|-j+2} \quad (i = 1,\dots,n-1)$$

Intuitively, $K''_i(sx,t)$ stores the sum of the discounted masses of matches of subsequences of length $i-1$ ending somewhere in the column just before the one being considered in the matrix and in some previous row. It is easy to see that:

$$\begin{aligned}
K''_i(sx,ty) &= \sum_{j:t_j=x} K'_{i-1}(s,t[1:j-1])\lambda^{(|t|+1)-j+2} \\
&= \lambda K''_i(sx,t)
\end{aligned}$$

if $x \neq y$ and

$$\begin{aligned}
K''_i(sx,tx) &= \sum_{j:t_j=x} K'_{i-1}(s,t[1:j-1])\lambda^{(|t|+1)-j+2} \\
&+ K'_{i-1}(s,t)\lambda^{(|t|+1)-(|t|+1)+2} \\
&= \lambda K''_i(sx,t) + \lambda^2 K'_{i-1}(s,t)
\end{aligned}$$

otherwise. K' can thus be expressed as function of K'' as:

$$K'_i(sx,t) = \lambda K'_i(s,t) + K''_i(sx,t) \quad (i = 1,\dots,n-1)$$

Having introduced K'', for each new element in s and t a single sum for updating K'' is sufficient, instead of a sum over all values for $j : t_j = x$, and the overall time complexity is reduced to $O(n|s||t|)$.

References

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, 1992.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

N. Cristianini and J. Shawe-Taylor. *Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000.

N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola. On kernel-target alignment. Technical Report 2001-099, NeuroCOLT, 2001. http://www.neurocolt.com/tech_reps/2001/01099.ps.

S. C. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

S. T. Dumais, T. K. Landauer, and M. L. Littman. Automatic cross-linguistic information retrieval using latent semantic indexing. In *Proceedings of the ACM SIGIR Conference on Research and Devlopment in Information Retrieval (SIGIR'96)*, 1996.

E. Gaussier, G. Grefenstette, D. Hull, and C. Roux. Recherche d'information en franais et traitement automatique des langues. *Traitement Automatique des Langues*, 41(2), 2000. Hermes.

D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, Department of Computer Science, University of California at Santa Cruz, Santa Cruz, CA, 1999.

R. Herbrich. *Learning Kernel Classifiers: Theory and Algorithms*. The MIT Press, Cambridge, Mass., 2002.

T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning (ECML98)*, number 1398 in Lecture Notes in Computer Science, pages 137–142. Springer Verlag, 1998.

T. Joachims. Making large-scale svm learning practical. In Bernhard Schölkopf, Chris Burges, and Alex Smola, editors, *Advances in Kernel Methods — Support Vector Learning*. MIT Press, 1999.

H. Lodhi, N. Cristianini, J. Shawe-Taylor, and C. Watkins. Text classication using string kernel. In *Advances in Neural Information Processing Systems 13*. MIT Press, 2001.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

J. Perez-Carballo and T. Strzalkowski. Natural language information retrieval: progress report. *Information Processing and Management*, 36(1):155–178, 2000.

G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, Mass., 2002.

P. Sheridan and J.-P. Ballerini. Experiments in multilingual information retrieval using the SPIDER system. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'96)*, pages 58–65, 1996.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

G. Wahba. *Spline Models for Observational Data*. Number 59 in CBSM-NSF Regional Conf. Ser. Appl. Math. SIAM, 1990.

C. Watkins. Dynamic alignment kernels. Technical Report CSD-TR-98-11, Department of Computer Science, Royal Holloway University of London, 1999.

S. K. M. Wong, W. Ziarko, and P. C. N. Wong. Generalized vector space model in information retrieval. In *Proceedings of the ACM SIGIR Conference on Research and Devlopment in Information Retrieval (SIGIR'85)*, pages 18–25, 1985.

Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 42–49, 1999.