In Proceedings of the Nineteenth International Conference on Machine Learning (ICML '02)

Syllables and other String Kernel Extensions

Craig Saunders CRAIG@CS.RHUL.AC.UK
Hauke Tschach HAUKE@CS.RHUL.AC.UK
John Shawe-Taylor JOHN@CS.RHUL.AC.UK
Department of Computer Science, Royal Holloway, University of London, Egham, Surrey, TW20 0EX, UK

Abstract

Recently, the use of string kernels that compare documents as a string of letters has been shown to achieve good results on text classification problems. In this paper we introduce the application of the string kernel in conjunction with syllables. Using syllables shortens the representation of documents and as a result reduces computation time. Moreover syllables provide a more natural representation of text; rather than the traditional coarse representation given by the bag-of-words, or the too fine one resulting from considering individual letters only. We give some experimental results which show that syllables can be effectively used in text-categorisation problems. In this paper we also propose two extensions to the string kernel. The first introduces a new lambda-weighting scheme, where different symbols can be given differing decay weightings. This may be useful in text and other applications where the insertion of certain symbols may be known to be less significant. We also introduce the concept of 'soft matching', where symbols can match (possibly weighted by relevance) even if they are not identical. Again, this provides a method of incorporating prior knowledge where certain symbols can be regarded as a partial or exact match and contribute to the overall similarity measure for two data items.

1. Introduction

Over the past few years there has been considerable interest in kernel-based algorithms. These algorithms use a dual-based representation to facilitate the use of kernel-functions, which correspond to calculating an inner product between vectors after first (implicitly) non-linearly mapping them into a higher dimensional space. The 'kernel trick' allows many different types of linear classifier to be efficiently constructed in a high-dimensional space, and has lead to many successful kernel-based algorithms, the most prominent of which has been the Support Vector Machine (Boser et al., 1992).

Until recently, in order to use these algorithms one had first to create a vector representation of the data. For many types of data, attributes from test examples are already in this form and no real conversion process is necessary. When dealing with types of data that do not have a natural vector representation (e.g. text data, gene sequences) it is common practice to explicitly construct vectors of real valued features that then can be used as the data points above.

A standard approach for kernel based algorithms with text data is the bag of words approach as used by Joachims (Joachims, 1997). After some appropriate preprocessing for each document a vector is created with each entry containing the (weighted) frequency of one particular word. This is known as the TFIDF representation and is defined as follows. First the term frequencies are obtained, $TF(w_i, x)$ is the number of times that a word w_i occurs in document x. These values are then scaled by their inverse document frequency,

$$IDF(w_i) = \log\left(\frac{n}{DF(w_i)}\right)$$

where n is the total number of documents in the corpus and the document frequency $DF(w_i)$ is the number of documents in which the word w_i occurs. The weight is then given by $TF(w_i, x) \times IDF(w_i)$.

With this representation the information used is the number of appearances of different words – their order in the document is ignored. Recently kernels which operate on discrete structures have been developed

(Haussler, 1999; Watkins, 1999). One example of such a kernel is the string kernel, which works directly on strings of text without the need to first create a feature vector. This has successfully been applied to text categorisation problems (Lodhi et al., 2001), but does have some drawbacks in that it is computationally expensive to compute, and using the kernel with single letters can perhaps seem a little unnatural.

In this paper we examine the use of the string kernel technique with syllables. As mentioned before one disadvantage with the bag-of-words approach is that the information regarding the ordering of the words is lost. The use of syllables or even words as the tokens on which the string kernel is constructed would result in shorter documents which would reduce the computation time. Whereas letters are too fine a representation, it may be that words are too coarse. Marginally different spellings or prefixes/suffixes could distort a matching kernel based on words (unless a complex preprocessing algorithm is used to produce stems).

We also present two extensions to the string kernel, which can be used with letters, syllables, words or any other representation. The first extension introduces the possibility of different decay weights λ for different symbols. The second extension introduces the idea of 'soft matching', where symbols can count as a match (or weighted partial match) even if they are not identical.

The rest of this paper is laid out as follows. In Section 2 we briefly review the string kernel and its associated features. In Section 3 we consider the use of syllables in conjunction with the string kernel. In Section 4 we introduce the two variants of the string kernel and in Section 5 we present some initial experimental results. We end with some conclusions and suggestions for future work in Section 6.

2. The String Kernel

The string kernel has been shown to be successful when applied to some small text categorisation tasks (Lodhi et al., 2001). The basic idea is to compare two documents by looking at common subsequences of a fixed length. The subsequences do not have to be contiguous. That means that e.g. the subsequence cat is present both in the word **cat**erpillar and in the word **c**ar**t**. If the appearances of substrings are more coherent they receive a higher weighting than appearances with larger gaps. This is done by penalising the lengths of the gaps exponentially with a decay factor $\lambda \in (0, 1)$, see Table 1. Although the results of experiments conducted with the string kernel on subsets of the Reuters-21578 dataset (Lewis, 1997) look promising (Lodhi et al., 2001), it does have a high computational cost. In (Lodhi et al., 2000) a dynamic programming technique for calculating the kernel was given that had time complexity of $O(n|s||t|)$ where n is the length of sub-sequences being considered, and $|s|$ and $|t|$ are the lengths of the two documents. This is a high computational cost (even for relatively short documents) and makes applying it to even medium sized data sets impractical. Some recent work based on approximating the gram matrix generated by the string kernel (Lodhi et al., 2002) has made progress and applied the kernel to larger data sets, however here we would not only like to reduce computational costs, we would also like to explore a new (perhaps better) representation for text.

	c-a	c-r	a-r	c-t	a-t	c-u	u-t
ϕ(car)	λ^2	λ^3	λ^2	0	0	0	0
ϕ(cat)	λ^2	0	0	λ^3	λ^2	0	0
ϕ(cut)	0	0	0	λ^3	0	λ^2	λ^2

Table 1. The 2-character features of the words 'car', 'cat', and 'cut'.

3. The Syllable Kernel

The idea of the syllable kernel is to replace the individual letters that are used as the 'symbols' in the original string kernel by larger (and more informative) chunks of the documents in such a way that the documents consist of fewer 'symbols' and therefore become shorter. Motivated in part by this speed up advantage we considered syllables rather than letters as our underlying representation. If for example there are on average 3 letters to each syllable, then the time taken to analyse documents shrinks by a factor of 9.

In order to obtain syllables we used Franklin M. Liang's hyphenation algorithm as described in (Knuth, 1984) with American and French hyphenation pattern files. Note that any other syllabification algorithm could have been used at this point. We are aware that a certain number of errors are made by such algorithms (especially those that operate at speed), but as long as they are consistent across the corpus, we would expect that the effect of these errors on any results obtained would be negligible.

3.1 Syllables in Reuters-21578

As an example of the results of the syllabification process, Table 2 shows a list of words, and the resulting syllables produced. These words were taken from the Reuters-21578 collection, and are all the words in that

Table 2. An example of the results from syllabifying different words.

Word	Syllables	Word	Syllables
computeraided	com-put-eraid-ed	computer	com-put-er
computerknowledge	com-put-er-knowl-edge	computed	com-put-ed
supercomputer	su-per-com-put-er	computers	com-put-er-s
supercomputers	su-per-com-put-er-s	computing	com-put-ing
computerized	com-put-er-ized	ltcomputer	lt-com-put-er
computerbased	com-put-er-based		
microcomputer	mi-cro-com-put-er		
computerization	com-put-er-i-za-tion		

collection that contain the substring 'comput'. Notice how the different spellings and prefixes and suffixes change, but the syllables seem to capture a relationship. It is worth noting that with clever stemming the bag-of-words approach would have picked up matches for computer, computed, computing, etc. as well as matches for the two supercomputer words. It would not of course produce matches for all the words (although there is obviously some similarity), and it would not take into account the ordering of other neighbouring words. Similarly, although the standard string kernel based on letters would produce matches (all words have the substring 'comput', many other words contain the substring 'compu' and many more contain 'comp', for example comparable) – so that spurious (incorrect) matches would be introduced. Note that no other words in the entire set of documents contained both the syllables 'com' and 'put'. Also, the string and syllable kernels are able to cope with the misspelled entry 'ltcomputer', whereas the bag-of-words would not.

It is important to note that the syllabification process does not introduce a large time overhead due to pre-processing. Once a list of unique words from a document has been obtained, the syllabification process is simply run on the list. This can then be used as a look-up table in order to replace words in a document with their respective syllables. Note that as more documents are processed any new words encountered can be added to the list and only these would require syllabification.

4. Extensions of the String Kernel

In this section we introduce two new extensions to the standard string kernel. The first uses different values of lambda to assign different weights to 'insertions'. The second method allows the introduction of soft matching, so that a partial match can contribute to the overall similarity of two documents. Both of these techniques would allow prior information about the representation being used to be added into the kernel.

4.1 Simple Weighting of 'Symbols'

First of all let us introduce some notation. A string s is a finite sequence of symbols from an alphabet Σ, including the empty sequence. A string $s = s_1, \ldots, s_{|s|}$ has length $|s|$ and st is the concatenation of two strings s and t. The string $s[i:j]$ is the substring $s_i, \ldots, s_j$ of s. We say that u is a subsequence of s if there exist indices $\mathbf{i} = (i_1, \ldots, i_{|u|})$ with $1 \leq i_1 < \ldots < i_{|u|} \leq |s|$, such that $u_j = s_{i_j}$, for $j = 1, \ldots, |u|$, or $u = s[\mathbf{i}]$ for short.

With the original string kernel we have one lambda. Here the idea is that different symbols might have different significance. We therefore use (potentially) different lambdas for different symbols. For this purpose we define for each symbol $c \in \Sigma$ its own decay factor $\lambda_c \in (0,1)$. The u coordinate of the feature vector for the string s is now defined by

$$\overline{\phi}_u(s) = \sum_{\mathbf{i}:u=s[\mathbf{i}]} \prod_{j=i_1}^{i_{|\mathbf{i}|}} \lambda_{s_j} .$$

That means that e.g. the feature 'cat' for the document 'cartridge' would receive the weighting $\lambda_{\mathrm{c}}\lambda_{\mathrm{a}}\lambda_{\mathrm{r}}\lambda_{\mathrm{t}}$.

We now define the weighted string kernel $\hat{K}$ of two strings s and t as

$$\begin{aligned} \hat{K}_n(s,t) &= \sum_{u\in\Sigma^n} \overline{\phi}_u(s)\overline{\phi}_u(t) \\ &= \sum_{u\in\Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \prod_{k=i_1}^{i_{|\mathbf{i}|}} \prod_{l=j_1}^{j_{|\mathbf{j}|}} \lambda_{s_k}\lambda_{t_l}. \end{aligned}$$

The evaluation of $\hat{K}$ can be computed in a similar way to that of the original string kernel if we define

$$\begin{aligned} \hat{K}'_i(s,t) &= \sum_{u\in\Sigma^i} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \prod_{k=i_1}^{|s|} \prod_{l=j_1}^{|t|} \lambda_{s_k}\lambda_{t_l}, \\ \text{for } i &= 1, \ldots, n-1 \end{aligned}$$

and

$$\hat{K}''_i(sx,t) = \sum_{j:t_j=x} \hat{K}'_{i-1}(s,t[1:j-1])\lambda_x \prod_{l=j}^{|t|} \lambda_{t_l},$$

and use the following recursions:

$$\begin{aligned}
\hat{K}'_0(s,t) &= 1, \text{ for all } s,t,\\
\hat{K}'_i(s,t) &= 0, \text{ if } \min(|s|,|t|) < i,\\
\hat{K}_i(s,t) &= 0, \text{ if } \min(|s|,|t|) < i,\\
\hat{K}'_i(sx,t) &= \lambda_x \hat{K}'_i(s,t) + \hat{K}''_i(sx,t),\\
\hat{K}_n(sx,t) &= \hat{K}_n(s,t)\\
&+ \sum_{j:t_j=x} \hat{K}'_{n-1}(s,t[1:j-1])\lambda_x^2,\\
\hat{K}''_i(sx,tu) &= \hat{K}''_i(sx,t) \prod_{j=1}^{|u|} \lambda_{u_j},\\
&\quad \text{provided that } x \text{ does not occur in } u,\\
\hat{K}''_i(sx,tx) &= \lambda_x \left(\hat{K}''_i(sx,t) + \lambda_x \hat{K}'_{i-1}(s,t) \right).
\end{aligned}$$

As mentioned above, using differing values of lambda is one way of incorporating prior knowledge into the string kernel. One possible application of this technique would be to assign different lambdas to syllables of different lengths while using the syllable kernel. If words could be grouped into syntactical groups (e.g. adverbs, verbs, nouns etc.) then one could use a lower penalty for adverbs for example as their insertion is less likely to change the overall meaning of the sentence.

4.2 Weighting of Soft Matches

Rather than the standard approach of only matching identical symbols, a refinement would be not only to match *equal* 'symbols' but also *similar* 'symbols' – with an extra factor that ensures that these soft matches gain a lower weight than exact matches.

Look at the following example: If we have the words 'calf' and 'calves' the original string kernel with subsequence length three would only find one common subsequence: 'cal'. Since the letters 'f' and 'v' sound very similar (and indeed we know that plurals of words ending in '-f' are sometimes formed using '-ves' after the stem), we might also want these to register as a match. However, we probably want to assign a smaller weight to this soft match to distinguish it from an exact match.

More generally we can construct for each pair $\{u,v\}$ of subsequences a similarity value $A_{u,v}$. (That means that if we define $A_{u,v} = \delta_{u,v}$ (i.e. $A_{u,v} = 1$ if $u = v$, $A_{u,v} = 0$ otherwise) we recover the original string kernel). This leads to a symmetric matrix $\mathbf{A} = (A_{u,v})_{u,v\in\Sigma^n} \in \mathbb{R}_0^{+|\Sigma^n|\times|\Sigma^n|}$.

If this matrix is positive semi-definite[1] we can define a Kernel $\tilde{K}$ that uses this soft matching technique:

$$\tilde{K}_n(s,t) = \phi(s)^\top \mathbf{A}\phi(t) = \sum_{u\in\Sigma^n}\sum_{v\in\Sigma^n} \phi_u(s)\phi_v(t)A_{u,v}.$$

The crucial part of this approach is the matrix $\mathbf{A}$ since the evaluation of $\phi(s)^\top \mathbf{A}\phi(t)$ is done in the feature space. However, if we define a positive semi-definite matrix $\mathbf{a} \in \mathbb{R}_0^{+\Sigma\times\Sigma}$ that measures the similarity between two individual symbols we can define a sensible matrix $\mathbf{A}$ by $A_{u,v} = \prod_{i=1}^k a_{u_i,v_i}$. Since this is the k-fold tensor product of positive semi-definite matrices, it is also positive semi-definite.

If we define

$$\tilde{K}'_n(s,t) = \sum_{u\in\Sigma^n}\sum_{v\in\Sigma^n}\sum_{\mathbf{i}:u=s[\mathbf{i}]}\sum_{\mathbf{j}:v=t[\mathbf{j}]} \lambda^{|s|+|t|-i_1-j_1+2}\prod_{i=1}^n a_{u_i,v_i}$$

and

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

we are able to define some new recursions for this approach:

$$\begin{aligned}
\tilde{K}'_0(s,t) &= 1,\\
\tilde{K}'_i(s,t) &= 0, \text{ if } \min(|s|,|t|) < i,\\
\tilde{K}_i(s,t) &= 0, \text{ if } \min(|s|,|t|) < i,\\
\tilde{K}'_i(sx,t) &= \lambda\tilde{K}'_i(s,t) +\\
&\underbrace{\sum_{j=i}^{|t|} \tilde{K}'_{i-1}(s,t[1:j-1])\lambda^{|t|-j+2}a_{x,t_j}}_{=\tilde{K}''_i(sx,t)},\\
\tilde{K}_n(sx,t) &= \tilde{K}_n(s,t) +\\
&\sum_{j=n}^{|t|} \tilde{K}'_{n-1}(s,t[1:j-1])\lambda^2 a_{x,t_j},\\
\tilde{K}''_n(sx,ty) &= \lambda\tilde{K}''_i(sx,t) +\\
&\lambda^2\tilde{K}'_{i-1}(s,t)a_{x,y}\ \forall x,\, y\in\Sigma.
\end{aligned}$$

It can be seen that a combination of the two approaches for weighting as described above can easily

[1] Note that these matrices do not have to be positive semi-definite since e.g. $\begin{pmatrix} 1 & 0.8 & 0.2 \\ 0.8 & 1 & 0.8 \\ 0.2 & 0.8 & 1 \end{pmatrix}$ is not positive semi-definite.

be constructed. One application of the soft matching approach could be used at the word level. If a look-up table of synonyms or verbs and their future/past tense equivalents existed, then soft matches could be defined for words such as 'taught', 'teaching', 'instructed', etc.

5. Experiments

In order to present a fair comparison with the standard string kernel and the techniques presented here, we performed tests on small data sets which unless stated otherwise were subsets of the Reuters-21578 collection. In order to get an idea of the timings involved we conducted a timing experiment by generating a 1000×1000 kernel matrix on a 1000 document subset of the Reuters data set. So that a direct time comparison could be made between different representations, we used $k = 3$ for characters, syllables, and words when generating the matrix. Note however that for characters a larger value of k is needed in practice to achieve good results, so the efficiency increase in using larger symbols may in practice be larger (see Section 5.1). For $k = 3$ the timing values for the generation of the kernel matrix is as follows:

Characters	149m 18s
Syllables	29m 24s
Words	14m 40s

As can be seen from the timings, syllables give approximately a factor of 5 speed up in efficiency over characters and using words is twice as fast as using syllables. Obviously bag-of-words is much faster, here the kernel matrix creation takes approximately 10 seconds, however in this paper we are interested in test-set performance and can address the efficiency question at a later stage.

The following sections give the results of comparing the techniques in this paper to the string, word and bag-of-words representations on the Reuters data set and an English/French bilingual data set. We used a simple kernel-adatron algorithm (Cristianini & Shawe-Taylor, 2000) in all of our experiments. As well as the standard error rate, we also used the standard $F1$ measure to evaluate the different approaches. This is defined as follows:

$$F1 = \frac{2PR}{P + R}$$

where P = precision and R = recall.

5.1 Comparing the Syllable and String Kernels

For our first experiments we used cross-validation to determine the best parameters for each method. We used the first 150 positive and the first 150 negative examples of the acq-category as our tuning subset. First we ran 20 random splits with different values of C (powers of 10 from 0.01 to 100 000 000) and different values of λ to determine the 'optimal' values for C and λ using 200 examples for training and 100 examples for testing. After choosing the best parameters these 300 examples were used as a training set, and the next 700 examples (350 positive, 350 negative) of the acq-category were used as an independent test set.

In order to make a true comparison between our methods and the bag-of-words approach we are interested in comparing results to the TF-only bag-of-words model. This meant that all kernels perform a frequency count on their respective representations,[2] whereas the re-weighting of individual features (as done with TFIDF) could be achieved in all string kernel variants used by using the λ-weighting scheme introduced in Section 4.1. In order to see how re-weighting affects performance however, we will also include results for bag-of-words with TFIDF weightings. The results for the experiments described above are given in Table 3.

As can be seen from the table all methods have similar performance. Nonetheless the syllable kernel is clearly best among the unweighted kernels, with all of the string-kernel variants outperforming the TF only representation. If the classification error of the syllable kernel and the TF-only kernel is examined more closely, it is found that out of a total of 81 errors which at least one of the algorithms made, there is a total of 54 examples where their predictions differed. Of these 54 errors only 6 were made by the syllable kernel, which gives a strongly significant result in the improvement for the syllable kernel (a one tailed test gives a value of approx 1.2×10^{-8}). Note that the syllable kernel gets better results using $k = 3$ rather than $k = 4$ for the string kernel. This combined with the speedup gained by the representation gives an efficiency increase of just over a factor of 7. Notice however that the inclusion of the IDF weightings for bag-of-words increases performance. This suggests that an appropriate λ-weighting scheme for symbols in the string kernel variants would also yield gains in performance. The experiment was repeated using the same subset, but with the first 600 documents as training and the remaining 400 documents as a test set.

[2]Using TF-only representation with a linear kernel is identical to using the word kernel with $k = 1$

Table 3. Experiments with a subset of the Reuters data.

	String Kernel	Syllable Kernel	Word Kernel	B-o-w (TF only)	B-o-w (TFIDF)
Best C	1e5	1e7	1e5	1e5	1e5
Best λ	0.1	0.05	0.01	–	–
Best k	4	3	2	–	–
Classification error	0.1514	0.1229	0.1814	0.1829	0.0800
Value for F1	0.8675	0.8900	0.8457	0.8443	0.9251
Number of SVs	184	218	157	150	252
Time	208 min	29 min	9 min	10 sec	10 sec

The performance of all algorithms increased, though in this case the TF-only and string and word kernels performed equally well. There was a slight drop in performance however of both the syllable kernel and TFIDF. These results are shown in table 4.

5.2 Soft-matching

For the soft matches we calculated for each pair (s,t) of syllables the sum $a_{s,t} := K_1(s,t) + K_2(s,t)$ where K_i denotes the basic string kernel with $\lambda = 0.5$ and subsequence length i. Since this leads to a kernel matrix, the matrix $\mathbf{a}$ (that is implicitly used to construct the matrix $\mathbf{A}$ (see Section 4.2)) is positive definite. Obviously a more rigorous approach to obtaining figures for the "quality" of matches could be used, however we wanted to see if a simple approach could yield any improvements in performance. Experiments were performed on the same subsets as above, and the results are shown in Table 5, column 1. Unfortunately a slight drop in performance was observed, however the number of Support Vectors did decrease. One possible reason for the slight decrease in performance is that it is unclear if is that using a string kernel with $k = 1$ is a sensible metric for determining if two syllables are similar. Common syllables should perhaps be indicated by matching two or three letter subsequences between them. We have however demonstrated in principle that this type of technique can be used, and further experimentation using different methods to obtain 'matching scores' would be necessary to assess the true performance of this method when using syllables.

5.3 Combined Kernels and Uneven Weighting

In addition to the straight application of soft-matching to syllables, Table 5 also shows the results of two further experiments. The first combined several different length syllable-kernels. The syllable kernel was used with lengths 1–4 to produce a new kernel which was weighted in the following way: $K(\cdot,\cdot) = 0.2 \times K_1 + 0.3 \times K_2 + 0.3 \times K_3 + 0.2 \times K_4$. It is hoped that by combining these kernels we would be able to 'pick out' the important groups of syllables. When using only the first 300 examples to train on, the performance is slightly worse than the syllable approach, however for the larger data this technique outperforms all other methods. The second experiment (results in the third column in the table) was used to test the method introduced in Section 4.1 where different weightings are given to different symbols. For this experiment a syllable kernel was used where the weighting given to the syllable was proportional to its length, i.e. $\lambda_s = \lambda^{|s|}$ where $|s|$ is the length of the syllable s. The performance of this approach is very similar to the syllable kernel approach, however using a different weighting scheme may yield better results.

5.4 Experiments on Bilingual Data

In this section we give some results on English and French documents taken from the Hansard collection of Canadian parliament proceedings. English and corresponding French documents for different sessions of parliament can be extracted giving the same data set but in two different languages. The collection is however not a classification collection so we constructed a "health" category by using all documents with headings which could be considered to be based around a health issue (these included documents headed "Health", "Breast Cancer", "World Aids Day", etc.). A total of 500 positive and 500 negative examples were extracted for use as a subset (the headings were removed from the documents) in both the French and English versions. We then conducted experiments similar to those in Section 5.1 using splits of the first 300 examples to choose parameters and then using that as a training set whilst testing on 700 independent examples.

Table 6 shows the results on each corpus. Once again the results are similar for the different techniques, however for bag-of-words the performance actually decreases with the introduction of IDF. Both the TF representation and the word kernel perform well, with the word kernel with $k = 2$ performing better on the English documents and TF achieving good results on the French corpus, with the syllable kernel outper-

Table 4. Experiments with a subset of the Reuters data.

	String Kernel	Syllable Kernel	Word Kernel	B-o-w (TF only)	B-o-w (TFIDF)
Best C	1e5	1e5	1e5	1e5	1e5
Best λ	0.1	0.05	0.01	–	–
Best k	4	3	2	–	–
Classification error	0.0325	0.0350	0.0325	0.0325	0.0350
Value for F1	0.9682	0.9657	0.9682	0.9684	0.9659
Number of SVs	305	379	281	287	443

Table 5. Using a soft-matching approach, a combined kernel approach and when using an uneven weighting scheme.

300/700	Soft Syllable Kernel	Combined	Weighted
Best C	1e5	1e7	1e5
Best λ	0.01	0.05	0.7
Best k	3	–	3
Classification error	0.1500	0.1343	0.1243
Value for F1	0.8645	0.8810	0.8886
Number of SVs	141	242	169
600/400	**Soft Syllable Kernel**	**Combined**	**Weighted**
Best C	1e5	1e7	1e5
Best λ	0.01	0.05	0.7
Best k	3	–	3
Classification error	0.0400	0.0250	0.0400
Value for F1	0.9604	0.9754	0.9612
Number of SVs	260	359	328

Table 6. Experiments using an 'artifical' classification problem from the Canadian parliament proceedings.

English	String Kernel	Syllable Kernel	Word Kernel	B-o-w (TF only)	B-o-w (TFIDF)
Best C	1e5	1e7	1e5	1e5	1e5
Best λ	0.1	0.05	0.01	–	–
Best k	4	3	2	–	–
Classification error	0.0814	0.0771	0.0729	0.0757	0.0929
Value for F1	0.9153	0.9199	0.9251	0.9224	0.9040
Number of SVs	189	211	190	186	266
French	**String Kernel**	**Syllable Kernel**	**Word Kernel**	**B-o-w (TF only)**	**B-o-w (TFIDF)**
Best C	1e5	1e7	1e5	1e5	1e5
Best λ	0.1	0.05	0.01	–	–
Best k	4	3	2	–	–
Classification error	0.0971	0.0757	0.0771	0.0714	0.0971
Value for F1	0.8988	0.9219	0.9211	0.9269	0.8985
Number of SVs	205	230	207	197	271

Table 7. Experiments using and English and French corpus, 600 examples were used for training and 400 for testing

English	String Kernel	Syllable Kernel	Word Kernel	B-o-w (TF only)	B-o-w (TFIDF)
Best C	1e5	1e5	1e5	1e5	1e5
Best λ	0.1	0.05	0.01	–	–
Best k	4	3	2	–	–
Classification error	0.0900	0.0825	0.0850	0.0825	0.0900
Value for F1	0.9062	0.9147	0.9124	0.9152	0.9072
Number of SVs	287	327	281	282	462
French	**String Kernel**	**Syllable Kernel**	**Word Kernel**	**B-o-w (TF only)**	**B-o-w (TFIDF)**
Best C	1e5	1e5	1e5	1e5	1e5
Best λ	0.1	0.05	0.01	–	–
Best k	4	3	2	–	–
Classification error	0.0950	0.0875	0.0800	0.0850	0.925
Value for F1	0.9000	0.9091	0.9179	0.9124	0.9034
Number of SVs	294	352	327	301	498

forming the string kernel. Once again the experiments were repeated using the first 600 documents as training and 400 for testing (see Table 7). For this experiment a similar trend in results is observed. The performance decrease for IDF is unexpected, however the large number of distinct words in the French corpus (approx 12000 compared to 9000 for the English corpus) could account partially for this, along with the consideration that the topic in this case was artificially created, however further experimentation is necessary.

6. Conclusions

In this paper we have introduced a technique for applying the string kernel to representations of documents based on syllables. One advantage of using syllables is that document lengths are decreased which results in an increase in the speed of computation of the kernel. Initial experiments show that this technique is successful in practice, however further experimentation is needed (perhaps on larger data sets in conjunction with an approximation technique) in order to fully evaluate the approach. Experimental results were also given using an English and French corpus on a comparable document set. These results also indicated good performance for the string kernel variants, which outperformed the TFIDF bag of words approach.

We have also presented two extensions to the string kernel which allow prior knowledge to be captured and used. The first introduces a different λ (decay weighting) value for different underlying symbols. An application of this technique would be to syntactically group words and provide lower-penalties for the introduction of certain groups of words (such as adverbs) that do not necessarily change the overall meaning of the text. A successful term weighting scheme can improve overall performance, as was shown in the comparisons between TF and TFIDF bag-of-words representations. The second approach extends the kernel to allow the soft matching of underlying symbols. This would allow information regarding synonyms and other semantic knowledge to be incorporated into the matching process. We have demonstrated through experimentation with the syllable representation that both of these extensions work in principle, and these additions therefore provide a more general form of the string kernel which can be applied to different application areas. One application area in which we believe the string kernels would perform well is that of information retrieval. These kernels tend to perform well on short documents, and therefore using them to process queries in an information retrieval setting could yield good performance and would also be fast (due to queries being very short).

Acknowledgements

This research was supported in part by the EU under the KerMIT project no. IST-2000-25341.

References

Boser, B., Guyon, I., & Vapnik, V. (1992). A training algorithm for optimal margin classifiers. In *Fifth annual workshop on computational learning theory*. ACM Press.

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines*. Cambridge University Press.

Haussler, D. (1999). *Convolution kernels on discrete structures* (Technical Report UCSC-CRL-99-10). University of California, Santa Cruz.

Joachims, T. (1997). *Text categorization with support vector machines: Learning with many relevant features* (Technical Report). Universität Dortmund, Fachbereich Informatik. Revised: 19 April 1998.

Knuth, D. E. (1984). *The tex book*, vol. A of *Computers & Typesetting*. Addison Wesley. Seventh printing, June 1986.

Lewis, D. D. (1997). Reuters-21578 text categorization test collection, distribution 1.0. Available from http://www.research.att.com/~lewis.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*. to appear.

Lodhi, H., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2000). *Text classification using string kernels* (Technical Report NC-TR-2000-079). NeuroColt Technical Report.

Lodhi, H., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2001). Text classification using string kernels. *Advances in Neural Information Processing Systems 13* (pp. 563–569). MIT Press.

Watkins, C. (1999). *Dynamic alignment kernels* (Technical Report CSD-TR-98-11). Royal Holloway, University of London.