Computational Statistics and Data Analysis 54 (2010) 290–297

Contents lists available at ScienceDirect

Computational Statistics and Data Analysis

journal homepage: www.elsevier.com/locate/csda

Kernel-based machine learning for fast text mining in R

Alexandros Karatzoglou [a,*], Ingo Feinerer [b]

[a] LITIS, INSA de Rouen, Avenue de Université, 76801 Saint-Etienne du Rouvray, France

[b] Database and Artificial Intelligence Group, Institute of Information Systems, Vienna University of Technology, Austria

ARTICLE INFO

Article history:
Received 5 November 2008
Received in revised form 15 September 2009
Accepted 19 September 2009
Available online 25 September 2009

ABSTRACT

Recent advances in the field of kernel-based machine learning methods allow fast processing of text using string kernels utilizing suffix arrays. **kernlab** provides both kernel methods' infrastructure and a large collection of already implemented algorithms and includes an implementation of suffix-array-based string kernels. Along with the use of the text mining infrastructure provided by **tm** these packages provide R with functionality in processing, visualizing and grouping large collections of text data using kernel methods. The emphasis is on the performance of various types of string kernels at these tasks.

1. Introduction

Kernel-based methods are state of the art machine learning methods which have gained prominence mainly through the successful application of Support Vector Machines (SVMs) in a large domain of applications. SVMs have also been applied in classification of text documents typically using the inner product between two vector representations of text documents.

1.1. Kernel methods

Kernel-based machine learning methods use an implicit mapping of the input data into a high-dimensional feature space defined by a kernel function, i.e., a function returning the inner product $\langle \Phi(x), \Phi(y) \rangle$ between the images of two data points x and y in the feature space. The learning then takes place in the feature space, and the data points only appear inside dot products with other points. This is often referred to as the "kernel trick" (Schölkopf and Smola, 2002). More precisely, if a projection $\Phi : X \rightarrow H$ is used, the dot product $\langle \Phi(x), \Phi(y) \rangle$ can be represented by a kernel function k

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle, \tag{1}$$

which is computationally simpler than explicitly projecting x and y into the feature space H.

One interesting property of kernel-based learning systems is that, once a valid kernel function has been selected, one can practically work in spaces of any dimension without any significant additional computational cost, since feature mapping is never effectively performed. In fact, one does not even need to know which features are being used.

Another advantage of kernel-based machine learning methods is that one can design and use a kernel for a particular problem that could be applied directly to the data without the need for a feature extraction process. This is particularly important in problems where a lot of structure of the data is lost by the feature extraction process (e.g., text processing).

1.2. Kernel methods for text mining

New methods developed within the past years range from algorithms for e.g. clustering (Shi and Malik, 2000; Ng et al., 2001) over dimensionality reduction (Song et al., 2008) to two-sample tests (Song et al., 2008). The availability of kernels that

* Corresponding author.
E-mail addresses: alexandros.karatzoglou@insa-rouen.fr (A. Karatzoglou), feinerer@dbai.tuwien.ac.at (I. Feinerer).

0167-9473/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.csda.2009.09.023

can be used directly on text, e.g. Cancedda et al. (2003), without the need for a feature extraction step makes kernel methods a promising class of machine learning algorithms in the area of text mining. This class of methods is particularly appealing in an exploratory or inferential setting where outright performance is not as significant as it might be in a production system.

String kernels (Watkins, 2000; Herbrich, 2002) have been shown to be a promising alternative to vector representations of text. They provide excellent results in text classification and clustering, using SVMs and kernel-based clustering methods like spectral clustering (Karatzoglou and Feinerer, 2007), respectively. For the analysis of DNA sequences (Usotskaya and Ryabko, 2009) string kernels have also been frequently used. One of the main drawbacks in the application of string kernels to large document collections is that until recently most algorithms and implementations where both slow and scaled usually in $O((n+m)^2)$ where n and m are the number of characters in the two documents. String kernels based on suffix trees which scaled $O(n+m)$ where introduced in Vishwanathan and Smola (2004) but still had drawbacks since the construction and use of a suffix tree requires a large amount of memory (typically $40n$) and has poor locality. Suffix-tree-based string kernels scale in theory in $O(n)$ but have been proven to be relatively slow for large scale text processing due to this inherent weakness. The use of suffix arrays (Teo and Vishwanathan, 2006) resolved most of these issues and has transformed string kernels into a very useful option for large scale text mining using kernel-based machine learning.

In this paper we will demonstrate the potential of kernel-based machine learning for text mining using suffix-array-based string kernel implementations in the **kernlab** package (Karatzoglou et al., 2004) of R Development Core Team (2008) and we will benchmark these string kernels. We will also briefly present the **tm** (Feinerer et al., 2008; Feinerer, 2008) R package which provides text mining functionality for R.

2. String kernels using suffix arrays

The main idea behind string kernels is to calculate a dot product between two documents or strings by counting the occurrence of common substrings in the two documents. These substrings do not need to be contiguous and can be weighted according to the matching length. String kernels have been designed for a specific task in mind such as e.g. classification of protein sequences (Leslie et al., 2002) where mismatches are allowed in matching the substrings between two sequences. Most string kernels can be used with both protein or DNA sequences and text data with excellent results (Lodhi et al., 2002).

In general string kernels work by calculating a weighted sum over the common substrings between two strings, as shown in Eq. (2). Different types of kernels arise by the use of different weighting schemas. The generic form of string kernels between two sets of characters x and x' is given by the equation

$$k(x, x') = \sum_{s \sqsubseteq x, s' \sqsubseteq x'} \lambda_s \delta_{s,s'} = \sum_{s \in A^+} \mathrm{num}_s(x)\mathrm{num}_s(x')\lambda_s, \tag{2}$$

where A^+ represents the set of all non-empty strings and λ_s is a weight or decay factor which can be chosen to be fixed for all substrings or can be set to a different value for each substring. $\mathrm{num}_s(x)$ denotes the number of occurrences of string s in x. This generic representation includes a large number of special cases, e.g. setting $\lambda_s \neq 0$ only for substrings that start and end with a whitespace character gives the "bag of words" kernel. In this paper we consider four different types of string kernels:

- k-spectrum (spectrum): This kernel considers only matching substrings of exactly length n, i.e. $\lambda_s = 1$ for all $|s| = n$.
- Constant (constant): All common substrings are matched and weighted equally.
- Exponential decay (exponential): All common substrings are matched but the substring weight decays as the matching substring gets shorter.
- Bounded range (boundrange) kernel where $\lambda_s = 0$ for all $|s| > n$ that is comparing all substrings of length less or equal to n.

To illustrate one of the simpler kernels, the spectrum kernel, consider the following example: computing the spectrum kernel between two strings `proposal` and `proposition` with matching string length set to 4 and the lambda parameter of 1 we get three matching substrings `prop`, `ropo` and `opos` and thus a kernel value of 3.

String kernels can be computed by building the suffix tree of a string x and computing the matching statistics of a string x' by traversing string x' through the suffix tree of x. A suffix tree is a data structure that represents the suffixes of a string S in a tree-like structure. The edges of a suffix tree are labeled with strings so that each path from the root to a leaf corresponds to a suffix of S. Given a suffix tree $S(x)$ it can be shown that the occurrence of a certain substring y can be calculated by the number of leaf nodes at the end of the path of y in the suffix tree (see Fig. 1). Auxiliary suffix links, linking identical suffixes in the tree are utilized to speed up the computations. Two main suffix tree operations are required to compute string kernels, a top down traversal for annotation and a suffix link traversal for computing matching statistics, both operations can be performed more efficiently on a suffix array.

The enhanced suffix array (Abouelhoda et al., 2004) of a string x is an array of integers corresponding to the lexicographically sorted suffixes of x with additional information stored to allow for the reproduction of almost all operations available on a suffix tree. Suffix arrays bring the advantage of better memory use and locality thus most operations can be performed faster than on the original suffix trees.

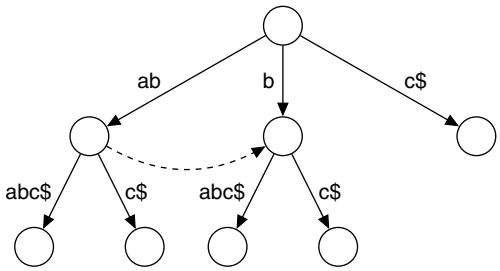

Fig. 1. Suffix tree of the string ababc$. To count the occurrence of a substring one needs to traverse the suffix tree to the substring and count the leaf nodes. Note e.g. the two leaf nodes below the ab suffix node. The dashed line represents a suffix link to speed up computation.

3. R infrastructure

R provides a unique environment for text mining but has until recently lacked tools that would provide the necessary infrastructure in order to handle text and compute basic text related operations, e.g. the computation of a term matrix. Package **tm** provides this functionality.

3.1. *tm*

The **tm** package provides a framework for text mining applications in R. It offers functionality for managing text documents, abstracts the process of document manipulation and eases the usage of heterogeneous text formats in R. The package has an integrated database backend support to minimize memory demands. An advanced metadata management is implemented for collections of text documents to alleviate the usage of large and metadata enriched document sets. Its data structures and algorithms can be extended to fit custom demands, since the package is designed in a modular way in order to enable easy integration of new file formats, readers, transformations and filter operations.

tm provides easy access to preprocessing and manipulation mechanisms such as whitespace removal, stemming, or conversion between file formats. Furthermore a generic filter architecture is included in order to filter documents given certain criteria, or to perform a full text search.

3.2. kernlab

kernlab is an extensible package for kernel-based machine learning methods in R. The package contains implementations of most popular kernels, and also includes a range of kernel methods for classification, regression (Support Vector Machine, Relevance Vector Machine), clustering (kernel *k*-means, Spectral Clustering), ranking, nonparametric two-sample tests and Principal Component Analysis (PCA).

A very important text mining task is the grouping of a set of documents into clusters of documents with the same topic, e.g., imagine a collection of news articles where one would like to group them according to topic similar to the way *Google News* does. To this end we employ kernel-based clustering methods in conjunction with string kernels.

We shortly describe two kernel-based clustering methods we are using for our experiments.

3.2.1. Kernel k-means

One of the drawbacks of the *k*-means clustering algorithm is that it cannot separate clusters that are not linearly separable in input space. One technique for dealing with this problem is mapping the data into a high-dimensional nonlinear feature space with the use of a kernel. Denoting clusters by π_j and a partitioning of points as $\pi_{j_{j=1}}^{k}$ and if Φ is the mapping function then the *k*-means objective function using Euclidean distances becomes

$$\mathcal{D}(\pi_{j_{j=1}}^{k}) = \sum_{j=1}^{k} \sum_{a \in \pi_j} \|\Phi(a) - m_j\|^2, \tag{3}$$

where $m_j = \frac{1}{\|\pi_j\|} \sum_{a \in \pi_j} \Phi(a)$ and in the expansion of the square norm only inner products of the form $\langle \Phi(a), \Phi(b) \rangle$ appear which are computed by the kernel function $k(a, b)$.

Kernel *k*-means is implemented in function `kkmeans()` in the **kernlab** package.

3.2.2. Spectral clustering

Spectral clustering (Shi and Malik, 2000; Ng et al., 2001) stems from the graph theoretical criterion for segmentation which is minimizing the *NCut*:

$$NCut(A, \hat{A}) = \left(\frac{1}{vol\, A} + \frac{1}{vol\, \hat{A}} \right) \sum_{i \in A, j \in \hat{A}} S_{ij}, \tag{4}$$

where S is the adjacency matrix of the graph, $d_i = \sum_{j \in I} S_{ij}$ denotes the degree of node i and $vol\, A = \sum_{i \in A} d_i$ is the volume of a set of nodes A. Optimizing *NCut* is NP-hard and thus Shi and Malik (2000) introduced the *NCut algorithm* which is a continuous approximation for solving the *NCut* problem by using the eigenvalues and eigenvectors of the normalized affinity matrix. This lead to the development of a new class of partitioning methods referred to as spectral clustering.

The main idea behind spectral clustering is to embed the data points of the partitioning problem into the subspace of the k largest eigenvectors of a normalized affinity matrix. The use of an affinity matrix also brings one of the advantages of kernel methods to spectral clustering, since one can define a suitable affinity for a given application. In our case we use a string kernel to define the affinities between two documents and construct the kernel matrix. The data is then embedded into the subspace of the largest eigenvectors of the normalized kernel matrix. This embedding usually leads to more straightforward clustering problems since points tend to form tight clusters in the eigenvector subspace. Using a simple clustering method like k-means on the embedded points usually leads to good performance.

Function `specc()` in package **kernlab** implements a spectral clustering algorithm based on the algorithm of Ng et al. (2001).

3.3. Framework for kernel-based machine learning for text data in R

Interoperability between **kernlab** and **tm** was a major design goal. In **tm**, `TextDocument` collections are essentially lists of character vectors where each character vector contains a text document, and can be used directly as input data for most machine learning functions in **kernlab** with the use of a string kernel. Functions in **tm** can be used to e.g. preprocess the text collections and perform part of speech tagging operations, or searches utilizing full text and metadata. In combination the two packages provide a unique framework in R for applying modern kernel-based machine learning methods to large text document collections.

A typical process flow for text mining with string kernels using **kernlab** and **tm** consists of following steps:

Data import. First, we can use **tm**'s functionality to read in our desired texts. **tm** provides enough abstraction to handle different input locations and input formats via the concept of so-called sources and readers, respectively. E.g., assume we have our dataset stored in a directory on a local hard disk. Then we can simply use a predefined source like `DirSource` which delivers its content. The reader now specifies how to actually parse each item delivered by the source (like XML or HTML documents). E.g., for some news stories from Reuters (see Section 4.1 for a detailed description) in XML format which are stored in the directory `Data/reuters` we can construct the corpus in R via

```
corpus <- Corpus(DirSource("Data/reuters"),
                 list(reader = readReut21578XML))
```

Fortunately, the **tm** package already provides a mini dataset containing some Reuters documents on the topic acquisitions. We will use it for demonstration (it can be loaded via `data(acq)`) as it provides easy reproducibility.

Preprocessing. For the experiments in this paper we conduct two preprocessing procedures: stemming and stopword removal.

Stemming denotes the process of deleting word suffixes to retrieve their radicals. It typically reduces the complexity without any severe loss of information (especially for bag of words). One of the best known stemming algorithm goes back to Porter (1980) describing an algorithm that removes common morphological and inflectional endings from English words. The **tm** function `stemDoc()` provides an interface to the Porter stemming algorithm, e.g., via

```
tmMap(acq, stemDoc)
```

we can easily stem all documents from `acq`, i.e., we map (apply) the stemming function on all documents.

Stopwords are words that are so common in a language that their information value is almost zero, i.e., their entropy is very low. Therefore it is a common procedure to remove such stopwords. Similarly to stemming, this functionality is already provided by **tm** via the `removeWords()` function. A single call suffices to remove all English stopwords from a text corpus:

```
tmMap(acq, removeWords, stopwords("english")).
```

String kernel construction. Next, after the corpus has been prepared, we build the desired string kernels via the `stringdot()` function. E.g., we can construct a spectrum string kernel where string sequences of 5 consecutive characters are considered:

```
sk <- stringdot(type = "spectrum", length = 5).
```

String kernel application. The previously constructed string kernel can now be applied to all kernel aware functions in R, like clustering or classification. E.g., a spectral clustering into two disjoint clusters can be done via:

```
specc(acq, centers = 2, kernel = sk).
```

4. Experiments

4.1. Data

Our first dataset is a subset of the Reuters-21 578 dataset (Lewis, 1997) containing stories collected by the Reuters news agency. The dataset is publicly available and has been widely used in text mining research within the last decade. Our subset contains 800 documents in the category "acq" (articles on acquisitions and mergers) and 400 in the category "crude" (stories in the context of crude oil). In case of clustering this is the full dataset, for classification randomized samples out of this dataset form the training set. The classification test set are further randomized samples with 200 acquisition and 100 crude oil stories.

The second dataset is a subset of the SpamAssassin public mail corpus (http://spamassassin.apache.org/publiccorpus/). It is freely available and offers authentic e-mail communication with classifications into normal (ham) and unsolicited (spam) mail. For the experiment we use 800 ham documents and 400 spam documents for clustering and as training set (again by building randomized samples) for our classification tasks. For the latter the test set consists of additional (randomly sampled) 100 ham and 100 spam documents.

4.2. Experiment setup

We evaluate the performance of the kernel-based clustering and classification algorithms using the suffix-array-based string kernels on the raw text data and on the preprocessed text data. First, we performed clustering on the Reuters and SpamAssassin datasets. We use the kernel k-means and spectral clustering algorithms implemented in the **kernlab** package. We manually set the number of desired clusters to two (i.e., $k = 2$) for both datasets since in both cases we know the true labels for each document (for the Reuters set the categories acquisitions and crude oil, for the SpamAssassin set ham and spam, respectively). Secondly, we perform a classification on both datasets using the Support Vector Machines (SVM) implementation in function `ksvm()` in package **kernlab**. After some tuning we set the cost parameter C of the SVM to 1 for all our experiments.

We used four types of recent string kernels: exponential, constant, spectrum, and boundrange. We conducted runs with various values of the string length parameter used by the spectrum and bounded range kernel. We evaluate the quality of the computed cluster and classification results via the cross-agreements between the cluster or classification labels and the true labels of each dataset. In addition we measured the CPU time required for the computation of the kernel matrices (denoted as kernel time) for each step. The actual clustering and classification time once the kernel matrices were computed is about 0.5 to 3 s and is thus not reported.

4.3. Results

From the results of our experiments on the Reuters dataset (see Table 1) we can conclude that the spectrum kernel is a generally good choice for processing text documents. It performs well overall compared to the alternative kernels. This can be attributed to the fact that it considers matches of exact length k while the other kernels also match all substrings of length smaller than k thus introducing additional information into the kernel matrix that might not be of benefit in the case of the rather lengthy text documents where the probability of matches will be high and thus additional noise is introduced. We also observe that performance is mostly better on the preprocessed documents. Preprocessing seems to bring benefits both in terms of the actual clustering and classification performance and the training time since preprocessed text documents are smaller than the original text documents and thus the kernel computations are performed faster.

The results of the experiments on the SpamAssassin dataset (see Table 2) show that it is important to also consider the nature of the text that is being processed. In the case of the SpamAssassin dataset alternative kernel types seem to perform on the same level if not better than the spectrum kernel. This could be attributed to the particular composition of spam messages where short words occur more frequently and arbitrary text and string sequences are added into the messages, to bypass filters. Kernels such as the boundrange kernel will capture more of this information. The spectral clustering algorithm performs in a slightly more consistent way across different kernel and parameter configurations than the kernel k-means although the latter seems to outperform the former for some configurations. Spectral clustering algorithms are known to produce relatively stable cluster partitions and we observe this in our experiments where the standard deviation on the spectral clustering runs is very close to zero. Again considering the type of text seems to be of importance in preprocessing since we observe that preprocessed documents seem to be slightly more difficult to group than the full texts. This can be attributed to the fact that some of the discriminatory features contained in the spam mails might be affected or removed by the preprocessing.

The results in Tables 3 and 4 confirm the excellent performance of SVMs in text classification when using string kernels. Note that until recently string kernels have been computationally expensive and this prohibited applications in many real world datasets. This constraint has been lifted as the comparison of the performance of the suffix-based kernels with a typical dynamic programming implementation shows (see Tables 5 and 6). The new version of the kernels outperforms the older (fullstring and string) on the Reuters dataset in terms of running time by a factor of 4 to 8 depending on the kernel

Table 1
Cross-agreement results and standard deviation for kernel k-means and spectral clustering over 20 runs on the original and preprocessed Reuters data for various types of string kernels set and different values for string length parameter. Differences between the top three performing kernel k-means configurations are not statistically significant (p-value > 0.05) while differences in the spectral clustering configurations are highly significant since the standard deviation is practically 0.

Type	Length	Plain agree	Prepr. agree	Plain agree	Prepr. agree
Algorithm		Kernel k-means		Spectral	
Exponential		0.6863 ± 0.04	0.705 ± 0.04	0.6891 ± 0.00	0.6725 ± 0.00
Constant		0.5141 ± 0.00	0.5142 ± 0.01	0.7958 ± 0.00	0.7883 ± 0.00
Spectrum	4	0.7766 ± 0.08	0.8941 ± 0.12	0.8408 ± 0.00	$\mathbf{0.8858 \pm 0.00}$
Spectrum	6	0.8641 ± 0.08	0.8421 ± 0.11	0.8575 ± 0.00	0.8725 ± 0.00
Spectrum	8	0.8641 ± 0.07	$\mathbf{0.8916 \pm 0.04}$	0.8566 ± 0.00	0.8550 ± 0.00
Spectrum	10	0.6833 ± 0.06	0.7283 ± 0.09	0.8683 ± 0.00	0.8283 ± 0.00
Boundrang	4	0.6900 ± 0.07	0.6500 ± 0.09	0.6208 ± 0.00	0.6366 ± 0.00
Boundrang	6	0.6491 ± 0.05	0.7825 ± 0.08	0.6975 ± 0.00	0.6833 ± 0.00
Boundrang	8	0.6991 ± 0.04	0.6133 ± 0.07	0.7700 ± 0.00	0.7175 ± 0.00
Boundrang	10	0.7283 ± 0.06	0.6341 ± 0.04	0.7816 ± 0.00	0.7358 ± 0.00
Fullstring	8			0.8266 ± 0.00	0.7923 ± 0.00
String	8			0.8558 ± 0.00	0.8553 ± 0.00

Table 2
Cross-agreement results and standard deviation for kernel k-means and spectral clustering on the original and preprocessed SpamAssassin dataset under different string kernel parameters. Differences between the top three performing kernel k-means configuration are not statistically significant (p-value > 0.05) while differences in the spectral clustering configurations are highly significant since the standard deviation is practically 0.

Type	Length	Plain agree	Prep. agree	Plain agree	Prep. agree
Algorithm		Kernel k-means		Spectral	
Exponential		0.7358 ± 0.06	0.7308 ± 0.05	0.6183 ± 0.00	0.6150 ± 0.00
Constant		0.6170 ± 0.02	0.5975 ± 0.01	0.5991 ± 0.00	0.6008 ± 0.00
Spectrum	4	0.6525 ± 0.04	0.6341 ± 0.04	$\mathbf{0.7491 \pm 0.00}$	0.5867 ± 0.00
Spectrum	6	0.6366 ± 0.05	0.5883 ± 0.05	0.6691 ± 0.00	0.6692 ± 0.00
Spectrum	8	0.6366 ± 0.06	0.5467 ± 0.05	0.6691 ± 0.00	0.6692 ± 0.00
Spectrum	10	0.5891 ± 0.05	0.5517 ± 0.04	0.6675 ± 0.00	0.6675 ± 0.00
Boundrang	4	$\mathbf{0.7750 \pm 0.04}$	0.6741 ± 0.04	0.7200 ± 0.00	0.6467 ± 0.00
Boundrang	6	0.7583 ± 0.05	0.7433 ± 0.05	0.7300 ± 0.00	0.6200 ± 0.00
Boundrang	8	0.7541 ± 0.05	0.6917 ± 0.05	0.6175 ± 0.00	0.6175 ± 0.00
Boundrang	10	0.7450 ± 0.05	0.6350 ± 0.04	0.6158 ± 0.00	0.6116 ± 0.00

Table 3
Cross-agreement results and standard deviation on the test set for SVM classification on the Reuters dataset under different string kernel parameters on 20 runs using different samples for the test set. Differences are statistically significant since the standard deviation is very close to zero.

Type	Length	Plain agree	Prep. agree
Exponential		0.9633 ± 0.00	0.9766 ± 0.00
Constant		0.8333 ± 0.00	0.8633 ± 0.00
Spectrum	4	$\mathbf{0.9900 \pm 0.00}$	$\mathbf{0.9900 \pm 0.00}$
Spectrum	6	0.9800 ± 0.00	0.9866 ± 0.00
Spectrum	8	0.9766 ± 0.00	0.9700 ± 0.00
Spectrum	10	0.9566 ± 0.00	0.9300 ± 0.00
Boundrang	4	0.9600 ± 0.00	0.9666 ± 0.00
Boundrang	6	0.9666 ± 0.00	0.9833 ± 0.00
Boundrang	8	0.9633 ± 0.00	0.9833 ± 0.00
Boundrang	10	0.9633 ± 0.00	0.9833 ± 0.00

Table 4
Cross-agreement results on the test set for SVM classification on the SpamAssassin dataset over 20 runs under different string kernel parameters and test and train splits. The difference of performance over the best two performing configurations is not statistically significant (p-value > 0.05) while all other differences are statistically significant.

Type	Length	Plain agree	Prepr. agree
Exponential		0.9400 ± 0.01	0.9200 ± 0.01
Constant		0.8850 ± 0.02	0.9033 ± 0.02
Spectrum	4	$\mathbf{0.9700 \pm 0.02}$	0.9650 ± 0.02
Spectrum	6	0.9500 ± 0.02	0.9350 ± 0.02

type. Fig. 2 depicts the CPU time in seconds required to compute a full kernel matrix of varying sizes of text documents for the fast implementation of the spectrum kernel compared to a conventional implementation.

Table 5
Time needed, in seconds, to compute the kernel matrix for the Reuters data for different kernel types on the full text data (normal) and the data after preprocessing. Computations on the preprocessed data are about 25% faster. The fullstring and string kernels are dynamic programming implementations similar to the boundrange and spectrum kernel respectively, the suffix array implementation outperforms the older versions by a factor of 4 to 8.

Kernel	Exponential	Constant	Spectrum	Boundrange	Fullstring	String
Normal	558	514	520	520	2195	4521
Preproc.	414	382	390	386		

Table 6
Time needed, in seconds, to compute the kernel matrix for the SpamAssassin data for different kernel types on the full text data (normal) and the data after preprocessing. Computations on the preprocessed data are about 10% faster.

Kernel	Exponential	Constant	Spectrum	Boundrange
Normal	1765	1674	1672	1670
Preproc.	1580	1485	1501	1490

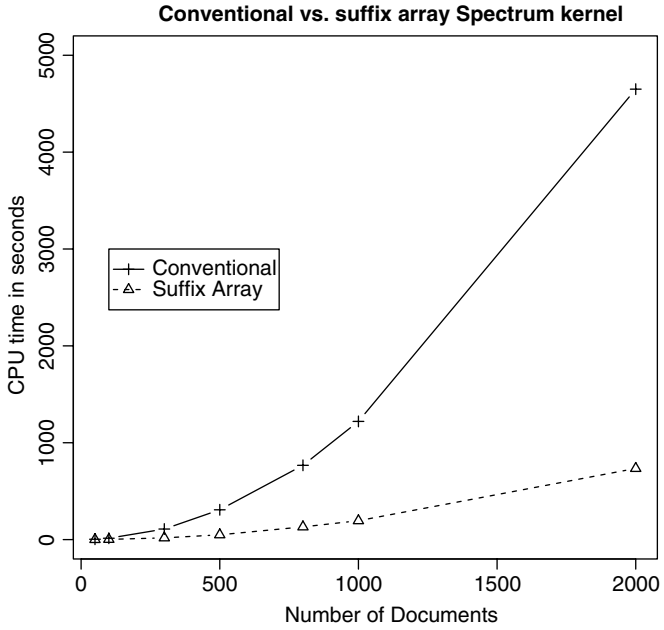

Fig. 2. CPU time in seconds to compute the full kernel matrix for different numbers of text documents with a conventional spectrum kernel and one based on suffix arrays.

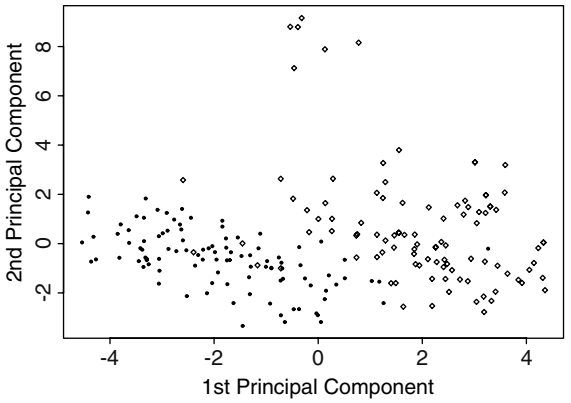

Fig. 3. The projection on two principal components of 200 Reuters text documents (100 crude in diamonds and 100 acquisition in dots) using a spectrum kernel and kernel PCA. The 6 points on the top of the image stem from articles on oil companies from Texas and Louisiana, while diamonds in the acquisition cluster are about oil company acquisitions.

Fig. 3 illustrates the projection of 200 text documents from the Reuters dataset on two principal components using kernel PCA with a spectrum string kernel. This type of visualization can be very useful as an exploratory tool in order to e.g. visually test for the existence of clusters in a set of documents.

5. Conclusion

We presented a fast implementation of string kernels with excellent performance in clustering and classification using spectral clustering and support vector machines. The availability of the kernels in **kernlab** along with the text processing infrastructure in the **tm** package provides R with advanced algorithms for the processing of text documents for the practitioner.

We utilized some of the functionality in **tm** to demonstrate both the usefulness of the package as well as the performance benefits which already some basic text preprocessing brings to both classification and clustering performance and running time. In our experiments we demonstrated the excellent performance of the kernels both in terms of computational cost and results. We showed that the spectrum kernel is a good choice when working with normal text data, and demonstrated that preprocessing the text is a good practice while the nature of the text needs to be considered.

Acknowledgment

Alexandros Karatzoglou was supported by a grant of the ANR - CADI project.

References

Schölkopf, B., Smola, A., 2002. Learning with Kernels. MIT Press.
Shi, J., Malik, J., 2000. Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (8), 888–905.
Ng, A., Jordan, M., Weiss, Y., 2001. On spectral clustering: Analysis and an algorithm. In: Advances in Neural Information Processing Systems, vol. 14. pp. 1–8.
Song, L., Smola, A., Borgwardt, K., Gretton, A., 2008. Colored maximum variance unfolding. In: Platt, J.C., Koller, D., Singer, Y., Roweis, S. (Eds.), Advances in Neural Information Processing Systems 20. MIT Press, Cambridge, MA, USA, pp. 1385–1392.
Cancedda, N., Gaussier, E., Goutte, C., Renders, J.-M., 2003. Word-sequence kernels. In: On Machine Learning Methods for Text and Images. Journal of Machine Learning Research 3 (6), 1059–1082 (special issue).
Watkins, C., 2000. Dynamic alignment kernels. In: Smola, A., Bartlett, P.L., Schölkopf, B., Schuurmans, D. (Eds.), Advances in Large Margin Classifiers. MIT Press, Cambridge, MA, pp. 39–50.
Herbrich, R., 2002. Learning Kernel Classifiers Theory and Algorithms. MIT Press.
Karatzoglou, A., Feinerer, I., 2007. Text clustering with string kernels in R. In: Decker, R., Lenz, H.-J. (Eds.), Advances in Data Analysis (Proceedings of the 30th Annual Conference of the Gesellschaft für Klassifikation e.V., Freie Universität Berlin, March 8–10, 2006). Studies in Classification, Data Analysis, and Knowledge Organization. Springer-Verlag, pp. 91–98.
Usotskaya, N., Ryabko, B., 2009. Application of information-theoretic tests for the analysis of DNA sequences based on Markov chain models. Computational Statistics & Data Analysis 53 (5), 1861–1872.
Vishwanathan, S., Smola, A.J., 2004. Fast kernels for string and tree matching. In: Schölkopf, B., Tsuda, K., Vert, J.P. (Eds.), Kernel Methods in Computational Biology. MIT Press, Cambridge, MA, pp. 113–130.
Teo, C.H., Vishwanathan, S.V.N., 2006. Fast and space efficient string kernels using suffix arrays. In: Proceedings of the 23rd International Conference on Machine Learning. ACM Press, Pittsburgh, Pennsylvania, pp. 929–936.
Karatzoglou, A., Smola, A., Hornik, K., Zeileis, A., 2004. kernlab—An S4 package for kernel methods in R. Journal of Statistical Software 11 (9), 1–20. http://www.jstatsoft.org/v11/i09/.
R Development Core Team, 2008, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0. http://www.R-project.org.
Feinerer, I., Hornik, K., Meyer, D., 2008. Text mining infrastructure in R. Journal of Statistical Software 25 (5), 1–54. http://www.jstatsoft.org/v25/i05.
Feinerer, I., 2008, tm: Text mining package, R package version 0.3-2, http://CRAN.R-project.org/package=tm.
Leslie, C.S., Eskin, E., Weston, J., Noble, W.S., 2002. Mismatch string kernels for SVM protein classification. In: Becker, S., Thrun, S., Obermayer, K. (Eds.), Advances in Neural Information Processing Systems, vol. 15. MIT Press, Cambridge, MA, pp. 1417–1424.
Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., Watkins, C., 2002. Text classification using string kernels. Journal of Machine Learning Research 2, 419–444.
Abouelhoda, M.I., Kurtz, S., Ohlebusch, E., 2004. Replacing suffix trees with enhanced suffix arrays. Journal of Discrete Algorithms 2, 53–86.
Porter, M., 1980. An algorithm for suffix stripping. Program 3, 130–137.
Lewis, D., 1997, Reuters-21578 text categorization test collection.