String kernels and similarity measures for information retrieval

André Martins
atm@priberam.pt

February 16, 2006

Abstract

Measuring a similarity between two strings is a fundamental step in many applications in areas such as text classification and information retrieval. Lately, kernel-based methods have been proposed for this task, both for text and biological sequences. Since kernels are inner products in a feature space, they naturally induce similarity measures.

Information-theoretical approaches have also been subject of recent research. The goal is to classify finite sequences without explicit knowledge of their statistical nature: sequences are considered similar if they are likely to be generated by the same source. There is experimental evidence that relative entropy (albeit not being a true metric) yields high accuracy in several classification tasks. Compression-based techniques, such as variations of the Ziv-Lempel algorithm for text, or *GenCompress* for biological sequences, have been used to estimate the relative entropy. Algorithmic concepts based on the Kolmogorov complexity provide theoretic background for these approaches.

This paper describes some string kernels and information theoretic methods. It evaluates the performance of both kinds of methods in text classification tasks, namely in the problems of authorship attribution, language detection, and cross-language document matching.

1 Introduction

Many applications in areas such as bioinformatics and natural language processing (NLP) require some kind of similarity measure between strings. In bioinformatics applications, strings are used to represent proteins as sequences of amino acids, or genomic DNA as sequences of nucleotides. Classification techniques for recognition of splice sites and other motifs in genomic sequences often require similarity measures [1]. Another example is multiple string comparison, either to infer evolutionary history from DNA sequences, or to characterize biological functions of families of proteins [2].

In NLP, strings are sequences of alphabet characters and represent text in natural language. Common applications that require the computation of

some sort of string similarity occur in the areas of text classification (categorization, authorship attribution, plagiarism detection, etc.) or information retrieval (IR) tasks [3].

In the last years, with the emergence of kernel-based methods for pattern analysis [4, 5] and classification techniques like support vector machines (SVM) [6], many string kernels have been proposed with different specificities according to the particular task and domain[1]. Since (positive-definite) kernels are inner products in a feature space, they naturally induce a similarity measure.

On the other hand, the framework provided by Shannon's information theory [7] suggests using information theoretic measures of (dis)similarity between sequences of symbols, such as relative entropy. Indeed, this have been lately a great focus of research. These methods do not assume explicit knowledge of the statistical nature of the sequences. Instead, they consider two sequences similar if they are likely to be generated by the same source. Different methods distinguish themselves on how they estimate the "divergence" between the two sequences. Particularly, variations of the well-known Ziv-Lempel algorithm for compression [8, 9] have been used for this task. The idea of using compression algorithms to estimate string similarity is also present in the formal definition of the non-computable "information distance" [10] based on the algorithmic notion of Kolmogorov complexity.

This paper is organized as follows: Section 2 gives a brief overview of kernel methods and describes some string kernels. Section 3 discusses the general notions of information and algorithmic theory and describes some information theoretic measures of string similarity. Section 4 describes how both kernel- and information theoretic-based approaches for computing the similarity of two strings can be implemented to run in linear time (with respect to the length of the strings). Section 5 plots the results obtained by applying both methods to the tasks of authorship attribution, language detection and cross-language document matching. Finally, Section 6 concludes the paper.

2 String kernels

2.1 Kernel methods

A full description of the underlying theory beyond kernel methods is given in [4, 5]. Roughly, given an input space $\mathcal{X}$, a (positive semi-definite) *kernel* is a function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ satisfying $\kappa(x, y) = \kappa(y, x)$ that is positive

[1] There is a web page dedicated to kernel methods that contains tutorials, publications, books, software tools, research events, etc.: `http://www.kernel-machines.org`.

semi-definite, i.e., such that for any $n \in \mathbb{N}$, $\{c_i\}_{i=1}^n \in \mathbb{R}^n$, and $\{x_i\}_{i=1}^n \in \mathcal{X}^n$:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j \kappa(x_i, x_j) \geq 0. \tag{1}$$

Given a set of observations $\{x_i\}_{i=1}^n$, the $n \times n$-matrix $K = [k_{ij}] := [\kappa(x_i, x_j)]$ is denoted *Gram matrix*. It is a direct consequence of Mercer's theorem that $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a (positive semi-definite) kernel if and only if there is a *feature space* $\mathcal{F}$ endowed with an inner product $\langle ., . \rangle$ and a map $\phi : \mathcal{X} \to \mathcal{F}$ satisfying for all $x, y \in \mathcal{X}$:

$$\kappa(x, y) = \langle \phi(x), \phi(y) \rangle, \tag{2}$$

Such feature space $\mathcal{F}$ can be regarded as a *reproducing kernel Hilbert space* of functions, i.e., where each $f \in \mathcal{F}$ satisfies the reproducing property $f(x) = \langle f, \kappa(x, .) \rangle = \langle f, \phi(x) \rangle$. Examples of basic kernels in vector spaces with inner product are the polynomial kernels, $\kappa(\mathbf{x}, \mathbf{y}) := (\langle \mathbf{x}, \mathbf{y} \rangle + R)^d$, where $d \in \mathbb{N}$ and R is a non-negative real parameter, and the Gaussian kernel, $\kappa(\mathbf{x}, \mathbf{y}) := \exp(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2})$, where $\sigma > 0$ is a parameter. Note that the Gaussian kernels may apply more generally to metric spaces, i.e., spaces endowed with a metric $d(x, y)$, via $\kappa(x, y) := \exp(-\alpha d(x, y)^2)$. They are also called radial basis function (RBF) kernels. The "kernel trick" allows to perform non-linear computations in the input space $\mathcal{X}$ by using classical well-known linear algorithms in a feature space $\mathcal{F}$, often with high (possibly infinite) dimension, without having to explicitly compute in that feature space. Instead, the kernel function in $\mathcal{X}$ is used whenever one needs to compute an inner product in $\mathcal{F}$. The support vector machines (SVMs) [6] are the most known example of applying the kernel trick. Learning algorithms that make use of kernels are generally called *kernel methods*.

2.2 The bag-of-words kernel

The *vector space model* (VSM) is a very well known model used in information retrieval (IR). Given a set of words (or *terms*) $\mathcal{T}$, and a set of documents $\mathcal{D}$, it considers each document as a "bag-of-words", i.e., an unordered multi-set of terms. Each document d is modeled as a vector $\mathbf{d}$ in a vector space $\mathbb{R}^{|\mathcal{T}|}$ indexed by terms, i.e., $\mathbf{d} := (w_d(t_1), \ldots, w_d(t_n))$, where $w_d(t)$ denotes the weight (e.g., the frequency) of the term t in document d. A common similarity measure used to compare documents d_1 and d_2 (e.g., a document and a query) is the *cosine measure*, that computes the cosine of the angle θ between the two vectors $\mathbf{d}_1$ and $\mathbf{d}_2$ through

$$\cos\theta = \frac{\langle \mathbf{d}_1, \mathbf{d}_2 \rangle}{\|\mathbf{d}_1\| \cdot \|\mathbf{d}_2\|}. \tag{3}$$

Although the vector space model was proposed before the generalized used of kernel-based algorithms, it is immediate that it can be easily "kernelized" by considering the input space $\mathcal{D}$, the feature space $\mathbb{R}^{|\mathcal{T}|}$, and the feature map $\phi(d) = \mathbf{d}/\|\mathbf{d}\|$, giving rise to the Joachims et al.'s *bag-of-words kernel* [3]:

$$\begin{aligned}\kappa(d_1, d_2) &= \langle \phi(d_1), \phi(d_2) \rangle = \\ &= \frac{\langle \mathbf{d}_1, \mathbf{d}_2 \rangle}{\|\mathbf{d}_1\| \cdot \|\mathbf{d}_2\|} = \\ &= \frac{\sum_{t \in \mathcal{T}} w_{d_1}(t) w_{d_2}(t)}{\sqrt{\left(\sum_{t \in \mathcal{T}} w_{d_1}^2(t)\right) \cdot \left(\sum_{t \in \mathcal{T}} w_{d_2}^2(t)\right)}},\end{aligned} \tag{4}$$

which is simply the cosine measure (3). It is shown [3] that the generalized VSM (GVSM) and the latent semantic indexing (LSI) techniques can also be described as kernels in a feature space, the latter being a consequence of applying the so-called *kernel principal component analysis* (kPCA).

2.3 The p-spectrum kernel

Consider now an alphabet Σ and its Kleene closure Σ^*, i.e., the set of all finite strings formed by characters in Σ together with the empty string ϵ. Denote by $|x| \in \mathbb{N}$ the length of $x \in \Sigma^*$ and by $uv \in \Sigma^*$ the concatenation of strings u and $v \in \Sigma^*$. If $x = uvw$, u, v and w are respectively called a *prefix*, a *substring* and a *suffix* of x. Furthermore, we say that t is a *subsequence* of x if there exists indices $\mathbf{i} = (i_1, \ldots, i_{|t|})$ such that $t_j = x_{i_j}$, for $j = 1, \ldots, |t|$, or $t = x[\mathbf{i}]$ for short. It is easy to see that both substrings and subsequences induce partial orders in Σ^*: if v is a substring of x we denote it by $v \sqsubseteq x$; if t is a subsequence of x we denote it by $t \preceq x$. Clearly, $v \sqsubseteq x \Rightarrow v \preceq x$ (every substring is a subsequence).

We saw above that the feature space associated with the bag-of-words kernel is indexed by words, but other kernels were devised that are based on exact and inexact string matching, respectively indexed by substrings and subsequences. Leslie et al. [11] introduced the *p-spectrum kernel*. This kernel is associated with a feature space indexed by Σ^p, i.e., strings of length p, with the embedding given by:

$$\phi_u^p(s) = |\{(v_1, v_2) : s = v_1 u v_2\}|, \quad u \in \Sigma^p, \tag{5}$$

and is defined as:

$$\begin{aligned}\kappa_p(s, t) &= \langle \phi^p(s), \phi^p(t) \rangle = \\ &= \sum_{u \in \Sigma^p} \phi_u^p(s) \phi_u^p(t).\end{aligned} \tag{6}$$

Thus, the p-spectrum kernel $\kappa_p(s,t)$ counts all the substrings of length p in both strings and computes the sum of the products. The special case $p = 1$, i.e., the 1-spectrum kernel, is called *bag-of-characters* kernel. A weighted variation of the p-spectrum kernel is obtained if we associate non-negative weights w_u to each substring u in (5), leading to

$$\phi_u^p(s) = w_u \cdot |\{(v_1, v_2) : s = v_1 u v_2\}|, \quad u \in \Sigma^p, \tag{7}$$

Later in Section 4 we show how the p-spectrum can be computed in $O(|s| + |t|)$ time (i.e., linearly in the length of the strings), independently of p, using suffix trees.

2.4 The weighted all-substrings kernel

More recently, Vishwanathan et al. [12] introduced a more general kernel that takes into account the contribute of all the substrings. We call this kernel the *weighted all-substrings kernel*. For proper weight functions, it can be viewed as a linear combination of weighted p-spectrum kernels with non-negative coefficients (hence, it is a valid kernel [5]). The embedding is given by:

$$\phi_u(s) = \sqrt{w_u} \cdot |\{(v_1, v_2) : s = v_1 u v_2\}|, \quad u \in \Sigma^*, \tag{8}$$

and the *all-substrings* kernel is defined as:

$$\begin{aligned} \kappa(s,t) &= \langle \phi(s), \phi(t) \rangle = \\ &= \sum_{u \in \Sigma^*} \phi_u(s) \phi_u(t), \end{aligned} \tag{9}$$

that is, the number of occurrences of every substring u in both s and t is counted and weighted by w_u, where the latter may be a weight chosen *a priori* or after seeing data, e.g. inverse document frequency counting (also used in the bag-of-words kernel). It is easy to see that the weighted all-substrings kernel is a generalization of the previous two kernels:

- The bag-of-words kernel is generated by requiring u to be bounded by whitespace, i.e., $w_u \neq 0$ iff u is bounded by a whitespace character on either side.
- The (weighted) p-spectrum kernel is achieved by setting $w_u = 0$ for all $|u| \neq p$.

Vishwanathan et al. propose several criteria to choose the weights w_u in (8): the length of u, the position of each occurrence of u in s and t, or a dictionary whose entries $u_1, \ldots, u_n$ are those substrings that have non-zero weights. Whatever criterium is used, a method based on suffix trees and matching statistics is described in [12] that is able to compute the kernel in time $O(|s| + |t|)$. Moreover, it is shown that with pre-processing $O(|s|)$, any kernel $\kappa(s, t_i)$, for $i = 1, \ldots, k$ can be computed in time $O(|t_i|)$, which is particularly useful for classification with SVMs.

2.5 Subsequence kernels and inexact matching

Lodhi et al. [13] proposed a kernel based on inexact matching, called string subsequence kernel (SSK). Given a subsequence u of s, i.e., $u = s[\mathbf{i}]$, the *length* $l(\mathbf{i})$ of the subsequence in s is defined as $l(\mathbf{i}) = i_{|u|} - i_1 + 1$. Consider now strings of fixed length n and the feature space $\mathcal{F}_n := \mathbb{R}^{\Sigma^n}$. The feature mapping ϕ for a string s is given by defining the u-coordinate $\phi_u(s)$ for each $u \in \Sigma^n$ as:

$$\phi_u(s) = \sum_{\mathbf{i}:u=s[\mathbf{i}]} \lambda^{l(\mathbf{i})}, \tag{10}$$

with $\lambda \in]0, 1]$ configurable. These features measure the number of occurrences of subsequences in s, weighting them according to their lengths. Unlike the previous kernels, here gaps are allowed, though penalized by the λ parameter. The SSK is defined as:

$$\begin{aligned} \kappa_n(s,t) &= \langle \phi(s), \phi(t) \rangle = \\ &= \sum_{u \in \Sigma^n} \phi_u(s)\phi_u(t) = \\ &= \sum_{u \in \Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{i})+l(\mathbf{j})}. \end{aligned} \tag{11}$$

Note that this kernel converges to the n-spectrum kernel when $\lambda \to 0$.

Typically the SSK is computed in quadratic time using dynamic programming techniques. Unfortunately, there is not any known linear time algorithm to perform the computation, although there are faster techniques to approximate the kernel by using only a subset of the features [13]. There are other variations of the subsequence kernel that require also quadratic time computation by dynamic programming, and thus are not tractable for large strings, reason why we did not perform experiments using this sort of kernels.

Examples of other string kernels that have been proposed are the gappy kernel, the edit distance kernel, the Fisher kernel and other kernels from generative models. There have been lately many approaches to devise a probabilistic kernel for biological sequences (e.g., [14]). A recent survey describing these and other string kernels can be found in [1].

3 Information theoretic dissimilarity measures

3.1 Entropy and relative entropy

Information theoretic measures of (dis)similarity between sequences of symbols have been lately a great focus of research. Inspired by the universality (in the sense of asymptotically distribution independence) of some widely used sequence compression techniques, such as the Ziv-Lempel algorithm

[8, 9], these methods use the same ideas to provide an "universal" classification scheme. A recent survey and experimental evaluation of information theoretic methods for measuring similarity, with applications to text authorship attribution, can be found in [15]. For a detailed description of the fundamental concepts of Shannon's information theory, see [16].

Let X be a discrete random variable with values in $\mathcal{X}$ and probability mass function $p(x) = P(X = x), x \in \mathcal{X}$. The *entropy* $H(X)$ is defined by:

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x). \tag{12}$$

The *joint entropy* $H(X, Y)$ of a pair of discrete random variables X, Y with values respectively in $\mathcal{X}$ and $\mathcal{Y}$ is defined as

$$\begin{aligned} H(X, Y) &= -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(x, y) = \\ &= H(X) + H(Y|X), \end{aligned} \tag{13}$$

where $H(Y|X)$ is the *conditional entropy*:

$$H(Y|X) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(y|x). \tag{14}$$

The *relative entropy* or *Kullback-Leibler divergence* between two probability mass functions $p(x)$ and $q(x)$ is defined as:

$$D_{KL}(p\|q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}, \tag{15}$$

taking the natural conventions $0 \log \frac{0}{q} = 0$ and $p \log \frac{p}{0} = \infty$.

The relative entropy can be regarded as a dissimilarity measure between two distributions. It is always non-negative, and it is zero only if $p = q$. However, it is not a metric, since in general it is not symmetric and does not satisfy the triangle inequality. A symmetrized version of this divergence is given by $D_{KL}(p\|q) + D_{KL}(q\|p)$, although it also is not a metric. On the other hand, the *Jensen-Shannon divergence*, defined as:

$$D_{JS}(p\|q) = D_{KL}\left(p \middle\| \frac{p+q}{2}\right) + D_{KL}\left(q \middle\| \frac{p+q}{2}\right) \tag{16}$$

was recently proved to be the square of a metric [17].

3.2 Ziv-Lempel compression

In the late 70s, Ziv and Lempel proposed an algorithm for compressing finite sequences [8, 9]. Actually, they gave rise to two families of algorithms (known

as LZ77 and LZ78) that are asymptotically equivalent. In the following, we will talk about the LZ77 family whenever we mention the Ziv-Lempel (ZL) algorithm.

Consider a sequence of random variables $X_1, \ldots, X_n$. The *entropy rate* of the stochastic process $\{X_i\}$ is defined as $H(\{X_i\}) = \lim_{n\to\infty} \frac{1}{n} H(X_1, \ldots, X_n)$ when the limit exists. The asymptotic equipartition property [16] states that, for any stationary ergodic process (not necessarily Markovian):

$$-\frac{1}{n} \log p(X_1, \ldots, X_n) \to H(\{X_i\} \tag{17}$$

almost surely. For a stationary Markov chain of order p, it holds $H(\{X_i\}) = H(X_{p+1}|X_p, \ldots, X_1)$.

It is shown [8] that the ZL code length for a string $x \in \Sigma^n$, emitted by a stationary source, divided by n, converges almost surely to the entropy of the source as n increases. Let $c(x)$ denote the number of phrases in x resulting from the ZL sequential parsing of x into distinct phrases (i.e. substrings of x), such that each phrase is the shortest string which is not a previously parsed phrase. Then, the ZL code length for x can be approximated by

$$c(x) \log_2 c(x) \tag{18}$$

and it converges with probability 1 to $-(1/n) \log_2 p(x_1, \ldots, x_n)$ as $n \to \infty$ [16]. This suggests using the output of a ZL encoder to estimate the entropy of an unknown source without explicitly estimating its model parameters.

3.3 The Benedetto-Caglioti-Loreto method

Recently, Benedetto et al. [18] proposed a very simple way of using a ZL encoder (provided by the *gzip* tool) to estimate the relative entropy between two sources $\mathcal{A}$ and $\mathcal{B}$.

Let $|X|$ and L_X denote respectively the length (in bits) of the uncompressed string X and its length after the ZL compression. Denote by XY the concatenation of the string X and Y. Let A and B be "long" sequences emitted respectively by $\mathcal{A}$ and $\mathcal{B}$, and b a "small" sequence emitted by $\mathcal{B}$. As proposed by Benedetto et al., the relative entropy can be estimated by:

$$\hat{D}_{KL}(\mathcal{A}\|\mathcal{B}) = \frac{\Delta_{Ab} - \Delta_{Bb}}{|b|}, \tag{19}$$

where $\Delta_{Ab} = L_{Ab} - L_A$ and $\Delta_{Bb} = L_{Bb} - L_B$. Notice that $\Delta_{Ab}/|b|$ can be seen as the code length obtained when coding a sequence from $\mathcal{B}$ (string b) using a code optimized for $\mathcal{A}$, while $\Delta_{Bb}/|b|$ can be interpreted as an estimate of the entropy of $\mathcal{B}$. With this is mind, Benedetto et al. [18] defined a simplified "distance" function $d(A, B)$ between two strings,

$$d(A, B) = \Delta_{AB} = L_{AB} - L_A \tag{20}$$

and applied it successfully to a text authorship attribution problem. We call this the Benedetto-Caglioti-Loreto (BCL) method. Puglisi et al [19] later improved the BCL method to correctly handle the case of long strings.

3.4 The Ziv-Merhav method

Ziv and Merhav [20] proposed a theoretically well-defined method for estimating relative entropy, based on a variation of the ZL algorithm [8, 9] to perform the "cross-parsing" of two strings.

Let z and x be two strings of the same length n. First, z is parsed by the incremental ZL parsing algorithm into $c(z)$ distinct phrases such that each phrase is the shortest sequence that is not a previously parsed phrase. For example, if $n = 11$ and $z = abbbbaaabba$, then the self incremental parsing yields $a|b|bb|ba|aa|bba$, that is, $c(z) = 6$. Then, one applies a variation of the ZL algorithm that is a sequential cross-parsing of z with respect to x, where each phrase is the longest substring of z which was parsed in x. For example, if $x = baababaabba$, parsing z with respect to x yields $abb|bba|aabba$, that is, $c(z|x) = 3$.

Ziv and Merhav proved [20] that for two Markovian binary sequences x, z of length $n \to \infty$ the quantity

$$\Delta(z\|x) = \frac{1}{n}[c(z|x)\log_2 n - c(z)\log_2 c(z)] \tag{21}$$

converges almost surely to the relative entropy between the sources that emitted the sequences. Notice that $[c(z)\log_2 c(z)]/n$ can be seen as a measure of the entropy of the source that emitted z, while $[c(z|x)\log_2 n]/n$ provides an estimate of the code length obtained when coding z using a model for x. In Section 4 we show how the Ziv-Merhav (ZM) method can be implemented in linear time (i.e., $O(|x| + |z|)$) using suffix trees.

3.5 Other methods from information and algorithmic theory

More recently, El-Yaniv et al. [21], to overcome the fact that relative entropy is not a true metric, suggested using a generalization of the Jensen-Shannon divergence to "agnostically" classify Markovian sequences. To do this, they provide a way to merge two Markovian sequences as a mixture of the sources, and then use the ZM algorithm to estimate the two relative entropies in (16). They applied this method to build a phylogenetic tree of 18 languages.

Yet another approach to estimate the relative entropy between sources, called the *plug-in* method, consists on consider them as Markovian sources of some order p, and compute the empirical conditional probabilities and stationary probabilities by counting the number of symbols following each state, plugging the estimates in (15).

The idea of using compression algorithms to estimate string similarity is also present in the *GenCompress*-based algorithm to measure the relatedness

of two DNA sequences [22], and in a recent approach that uses the Burrows-Wheeler transform [23].

The concepts of (conditional) compression and string similarity come together in the definition of *information distance* by Li et al. [10]. This definition uses the concept of *Kolmogorov complexity* or *algorithmic entropy* of a string. Given a string x, the Kolmogorov complexity $K(x)$ is the length of the shortest binary program that outputs x. The computation is performed in an universal computer, such as an universal Turing machine. The *conditional Kolmogorov complexity* $K(x|y)$ of x relative to y is the length of the shortest program that outputs x if y is furnished as input. $K(x, y)$ denotes the length of the shortest program that prints out x and y and a description of how to tell them apart. The functions $K(.)$ and $K(.|.)$ are machine independent up to an additive constant. However, they are not computable; one can only obtain upper bounds $C(x) \geq K(x)$ and $C(x|y) \geq K(x|y)$ via compressors. That is, the Kolmogorov complexity $K(x)$ serves as the ultimate, lower bound of what a real-world compressor can possibly achieve. The *information distance* is defined as the length $E(x, y)$ of the shortest binary program that computes x from y as well as y from x. It is shown that it equals

$$E(x, y) = \max\{K(y|x), K(x|y)\} \tag{22}$$

up to an additive logarithmic term. Li et al. [10] propose the following normalized information distance:

$$d(x, y) = \frac{\max\{K(y|x), K(x|y)\}}{\max\{K(x), K(y)\}}. \tag{23}$$

Since this distance is not computable, it can only be approximated without convergence guarantees, using real-world compressors, i.e.,

$$\hat{d}(x, y) = \frac{\max\{C(y|x), C(x|y)\}}{\max\{C(x), C(y)\}}. \tag{24}$$

4 Implementation details

4.1 Suffix trees

A definition of suffix tree, together with full descriptions of related data structures and algorithms, can be found in [2].

We now describe how suffix trees can be used to implement the p-spectrum kernel, the weighted all-substrings kernel (using a length-dependent weight function), and the ZM algorithm for relative entropy estimation. Being $|s|$ and $|t|$ the lengths of the two strings from which one wants to measure the similarity, the use of suffix trees allows any of these methods to run in time $O(|s| + |t|)$, i.e., linearly with respect to the sum of their lengths.

4.2 The p-spectrum kernel

Recall the expression (6) for the p-spectrum kernel. It first counts all the occurrences of substrings of length p both in $|s|$ and $|t|$. Then, it computes the sum of the products. A naive implementation using an array of p-grams would have complexity $O(p|s|+p|t|+\Sigma^p)$, that would be exponential in p. On the other hand, dynamic programming would imply a quadratic complexity $O(|s| \cdot |t|)$.

A much better approach is to use tries (as proposed in [5]), a generalized suffix tree for s and t, or (even better) a suffix tree for the smallest of the two strings and matching statistics for the other (as proposed in [12]). A trie-based algorithm can be made to run in $O(p(|s| + |t|))$, which is acceptable for small values of p. It has the advantage of not being too space-consuming. The other methods achieve linear complexity, $O(|s| + |t|)$. We explain briefly the generalized suffix tree approach. The improvement with matching statistics is straightforward.

Append distinct terminating characters \$ and # to s and t, and build a generalized suffix tree $\mathcal{T}$ for $s\$$ and $t\#$ (equivalently, build a suffix tree for $s\$t\#$). This takes $O(|s| + |t|)$ time using, for instance, Ukkonen's algorithm [24]. For each node v in the tree, let $S(v)$ be the string label of v, and $|S(v)|$ the string depth of v. Consider the subtree rooted at v, denoted by $\mathcal{T}_v$. The number of occurrences of $S(v)$ in s (respectively t) is simply the number of leaves in $\mathcal{T}_v$ associated with s (resp. t). Denote this number by $n_s(v)$ (resp. $n_t(v)$), and the set of child nodes of v by $\texttt{Child}(v)$. Then, the following recurrence to compute $n_s(.)$ (and similarly to compute $n_t(.)$) is obvious:

$$n_s(v) = \begin{cases} 1 & \text{if } v \text{ is a leaf associated with } s, \\ \sum_{w \in \texttt{Child}(v)} n_s(w) & \text{otherwise.} \end{cases} \tag{25}$$

Hence, a simple depth-first search (DFS) on $\mathcal{T}$ (which takes $O(|s| + |t|)$ time) is able to obtain $n_s(v)$ and $n_t(v)$ for each node v of $\mathcal{T}$. A DFS is also able to calculate $|S(v)|$ for each v. Let $\pi(v)$ denote the parent of v if v is not the root. There is a p-gram in the edge that connects $\pi(v)$ to v if and only if $|S(\pi(v))| < p \leq |S(v)|$. Clearly, identical p-grams cannot appear in distinct edges. With this in mind, define the set $\mathcal{V}_p = \{v : |S(\pi(v))| < p \leq |S(v)|\}$. The p-spectrum kernel (6) can be obtained performing the following sum over the nodes in $\mathcal{V}_p$:

$$\kappa_p(s,t) = \sum_{v \in \mathcal{V}_p} n_s(v) n_t(v). \tag{26}$$

Since $|\mathcal{V}_p|$ cannot exceed the number of nodes in the tree, which in turn do not exceed the number of leaves $|s| + |t|$, we finally conclude that the overall complexity of this algorithm is $O(|s| + |t|)$.

Consider now the problem of filling the Gram matrix. Given n strings $s_1, \ldots, s_n$, each entry (i, j) of the Gram matrix must contain the value of

$\kappa_p(s_i, s_j)$. Due to symmetry, this task requires computing $\binom{2}{n} = O(n^2)$ kernels. If the length of each string is k, this means an overall complexity of $O(kn^2)$. For large values of n, this complexity can be reduced by a factor of b if we compute the Gram matrix by groups of $b \times b$ blocks. In the next section, that describes the computation of the weighted all-substrings kernel (from which the p-spectrum is, as stated in Section 2.4, a particular case), we will see how this "block computation" can be performed.

4.3 The weighted all-substrings kernel

The same reasoning as above may be applied for the weighted all-substrings kernel, using length-dependent weights. Recall expressions (8) and (9), and define $w_s = w'(|s|)$ to be a weight function that only depends on the substring size. Let $\mathcal{V}$ denote the set of all nodes in $\mathcal{T}$. Define $l_{\min}(v) = |S(\pi(v))| + 1$ and $l_{\max}(v) = |S(v)|$. Then, the following sum equals the weighted all-substrings kernel:

$$\begin{aligned}
\kappa(s,t) &= \sum_{v \in \mathcal{V}} \sum_{l=l_{\min}(v)}^{l_{\max}(v)} w'(l) n_s(v) n_t(v) = \\
&= \sum_{v \in \mathcal{V}} n_s(v) n_t(v) \sum_{l=l_{\min}(v)}^{l_{\max}(v)} w'(l) = \\
&= \sum_{v \in \mathcal{V}} n_s(v) n_t(v) w''(l_{\min}(v), l_{\max}(v)), \qquad (27)
\end{aligned}$$

where we introduced the function $w''(.,.)$ defined as $w''(l_1, l_2) = \sum_{l=l_1}^{l_2} w'(l)$.

A common choice of the length-dependent weight function is:

$$w_s = w'(|s|) = \begin{cases} \lambda^{|s|} & \text{if } p_{\min} \leq |s| \leq p_{\max}, \\ 0 & \text{otherwise,} \end{cases} \qquad (28)$$

where $0 < \lambda < 1$ is a decaying factor. With this choice, the function $w''(.,.)$ becomes simply the sum of a geometric sequence:

$$w''(l_1, l_2) = \begin{cases} \frac{1-\lambda^{l_2-l_1+1}}{1-\lambda} & \text{if } p_{\min} \leq l_1 \leq l_2 \leq p_{\max}, \\ \frac{1-\lambda^{l_2-p_{\min}+1}}{1-\lambda} & \text{if } l_1 < p_{\min} \leq l_2 \leq p_{\max}, \\ \frac{1-\lambda^{p_{\max}-l_1+1}}{1-\lambda} & \text{if } p_{\min} \leq l_1 \leq p_{\max} < l_2, \\ \frac{1-\lambda^{p_{\max}-p_{\min}+1}}{1-\lambda} & \text{if } l_1 < p_{\min} \leq p_{\max} < l_2, \\ 0 & \text{otherwise,} \end{cases} \qquad (29)$$

and thus its value can be obtained in constant time for any l_1 and l_2. Plugging this into (27), we are led to an overall complexity $O(|s| + |t|)$ also for the weighted all-substrings kernel with length-dependent weights. The computation of the Gram matrix is similar to the case of the p-spectrum

kernel: the most obvious algorithm require a time $O(kn^2)$. However, this complexity can be reduced by a factor of b if we compute the Gram matrix by groups of $b \times b$ blocks. The procedure is as follows: at each stage, build a generalized suffix tree for b strings at once, say $s_i, \ldots, s_{i-1+b}$. Then, for each following group of b strings, say $s_j, \ldots, s_{j-1+b}$, get a copy of the previous tree and add these strings to it, yielding a generalized suffix tree for $2b$ strings (thus requiring a space $\Theta(2bk)$). We are going to see that this tree allows to directly compute every kernels $\kappa(s_{i+p}, s_{i+q})$, $\kappa(s_{i+p}, s_{j+q})$, and $\kappa(s_{j+p}, s_{j+q})$, for $1 \leq p, q \leq b$, with additional time and space complexities respectively $O(bk)$ and $\Theta(bh)$, where h is the height of the tree, i.e., the number of nodes traversed in the path from the root to the deepest leave. This allows to fill at least two new $b \times b$ blocks of the Gram matrix. Since there are $\Theta(n^2/b^2)$ such blocks, the overall time complexity is $O(kn^2/b)$. The only drawback is space: at each instant, besides the $\Theta(n^2)$ Gram matrix, there is a copy of a generalized suffix tree of length $\Theta(bk)$, another generalized suffix tree of length $\Theta(2bk)$, and we need additional space $\Theta(bh)$ to compute the kernel (we are going to see how). This requires a total memory of $\Theta(n^2 + 3bk + bh) = \Theta(n^2 + bk + bh)$. In particular, if $b = \Theta(n)$ (e.g., if we simply build a generalized suffix tree for the n strings at once), time and space complexities become respectively $O(kn)$ and $\Theta(n^2 + kn + hn)$.

We now describe how to compute each kernel $\kappa(s_{i+p}, s_{i+q})$, $\kappa(s_{i+p}, s_{j+q})$, and $\kappa(s_{j+p}, s_{j+q})$, for $1 \leq p, q \leq b$, from the generalized suffix tree for $s_i, \ldots, s_b, s_j, \ldots, s_b$ with additional time and space complexities $O(bk)$ and $\Theta(bh)$. This is a particular case of computing the Gram matrix for n strings from their generalized suffix tree, which can be done with a single DFS as follows: let v be the current node. We need to compute a n-length vector N_v where each entry is the above value $n_{s_i}(v)$ (the number of s_i-leaves in the subtree rooted by v). If v is a s_j-leaf, then $n_{s_i}(v) = \delta_j$, that is, N_v is a vector with all elements zero except the j-th, which is one. If v is a internal node, N_v can be computed from its child nodes through (25). This N_v vector is used to increment each entry in the Gram matrix by $n_{s_i}(v) n_{s_j}(v) w''(l_{\min}(v), l_{\max}(v))$ according to (27). $w''(.,.)$ can be computed in time $O(1)$ via (29) if the string depths of the nodes are transmitted in the DFS. All this can be done in a single DFS, provided we save a copy of N_v to be used by the descendants of v to compute their contribution to the kernel. This implies keeping a total stack of h copies of n-length vectors, where h is the number of nodes traversed from the root to the deepest leave. Hence, an additional space of $\Theta(hn)$ is needed. The time complexity is proportional to the total number of nodes in the tree, thus is $O(kn)$.

4.4 The ZM algorithm for relative entropy estimation

Suffix trees may also be used to implement the ZL algorithm [2]. The idea is to take profit of their most important feature: linear time search.

At each step of the ZL algorithm, we want to obtain the shortest phrase not already parsed. This is the same as the longest parsed phrase plus the subsequent character. Suppose that at stage j we are in position i of a string s. We want to find a position p_j in the past, i.e., $1 \leq p_j < i$ such that $s[i..i + l_j - 1] = s[p_j..p_j + l_j - 1]$ and l_j is maximal. Then, code the j-th phrase with the triplet $(p_j, l_j, s[i + l_j])$ and repeat the procedure starting at position $i + l_j + 1$. Finding p_j and l_j can be done by an $O(l_j + 1)$-time search in the suffix tree for $s[1..i + l_j]$ (or $s[1..i]$ if we forbid overlapping). This search can be done in the full suffix tree for s, if we pre-process it to write at each node the biggest leaf number that descends from it. Alternatively, it can be done during the Ukkonen's algorithm for building the suffix tree for s, since this algorithm has the property of being "on-line" [24]. In both cases, the number of ZL phrases that result from the self-parsing of s, $c(s)$, can be found in $O(|s|)$ time, since $\sum_{j=1}^{c(s)}(1 + l_j) = |s|$.

As shown in (21), the ZM method to estimate the relative entropy of x and z counts the total number of phrases of the self-parsing of z, $c(z)$, and the total number of phrases of the cross parsing of z with respect to x, $c(z|x)$. The former can be obtained in $O(|z|)$ time as we have just seen; the latter can be obtained in $O(|x| + |z|)$ time as follows. First build a suffix tree for x in time $O(|x|)$. To perform the cross-parsing, we need at each stage j, starting from a position i of z, to find a position p_j in x such that $z[i..i + l_j - 1] = x[p_j..p_j + l_j - 1]$ and l_j is maximal. The j-th phrase is then $z[i..i + l_j - 1]$, the longest phrase already parsed in x, and can be represented by a pair (p_j, l_j). This can be done in time $O(|z|)$ as in self-parsing. Notice that it corresponds to a partial computation of the matching statistics of z with respect to x. The overall complexity is $O(|x| + |y|)$.

Given n strings $s_1, \ldots, s_n$, each of length $O(k)$, the computation of the whole matrix of relative entropies also requires $O(kn^2)$ time and $\Theta(k)$ space. However, in practice it is faster than the previous implementations, since it does not require building generalized suffix trees.

5 Experiments

We evaluated the performance of the above described algorithms (the p-spectrum kernel, the weighted all-substrings kernel, and the Ziv-Merhav estimation of the relative entropy) by applying them to the tasks of text authorship attribution, language detection, and cross-language document matching. We now summarize the results for each of these tasks.

5.1 Text authorship attribution

In the first task, a collection of texts from Portuguese authors, publicly available through Project Gutenberg[2], was employed. The list of authors and books is listed in Table 1.

Author	**Book**	**Identifier**
Alexandre Herculano	*Lendas e Narrativas (Tomo I)*	AHERCU00
Alexandre Herculano	*Lendas e Narrativas (Tomo II)*	AHERCU01
Alexandre Herculano	*Opúsculos (Tomo I)*	AHERCU02
Alexandre Herculano	*Opúsculos (Tomo II)*	AHERCU03
Alexandre Herculano	*Opúsculos (Tomo IV)*	AHERCU04
Alexandre Herculano	*Opúsculos (Tomo VII)*	AHERCU05
António Nobre	*Só*	ANOBRE00
Camilo C. Branco	*Amor de Perdição*	CCBRAN00
Cesário Verde	*O Livro de Cesário Verde*	CVERDE00
Eça de Queiroz	*O Mandarim*	EQUEIR00
Fernão Lopes	*Chronica de el-rei D. Pedro I*	FLOPES00
Guerra Junqueiro	*Contos para a Infância*	GJUNQU00
Júlio Dinis	*Os fidalgos da Casa Mourisca*	JDINIS00
Júlio Dinis	*Uma família ingleza*	JDINIS01
Júlio de Mattos	*A paranoia*	JMATOS00
Lopes Netto	*Lendas do Sul*	LNETTO00
Luis V. Camões	*Os Lusíadas*	LVCAMO00
Possidónio da Silva	*Noções elementares de archeologia*	NSILVA00
Nicolau Tolentino	*Obras poéticas*	NTOLEN00
Ruy de Pina	*Chronica d'El-Rei D. Affonso III*	RDPINA00
Ruy de Pina	*Chronica d'El-Rei D. Diniz (Vol. I)*	RDPINA01
Ramalho Ortigão	*As Farpas (1873-01/02)*	RORTIG00
Ramalho Ortigão	*As Farpas (1873-03/04)*	RORTIG01
Ramalho Ortigão	*As Farpas (1873-10/11)*	RORTIG02
Ramalho Ortigão	*As Farpas (1877-01/02)*	RORTIG03
Ramalho Ortigão	*As Farpas (1877-05/06)*	RORTIG04
Ramalho Ortigão	*As Farpas (1878-01)*	RORTIG05
Ramalho Ortigão	*As Farpas (1878-02/05)*	RORTIG06
Ramalho Ortigão	*As Farpas (1882-06/07)*	RORTIG07
Ramalho Ortigão	*As Farpas (1882-11/12)*	RORTIG08
Ramalho Ortigão	*As Farpas (1883-06)*	RORTIG09
Ramalho Ortigão	*As Farpas (1887-08-09)*	RORTIG10
Ramalho Ortigão	*As Farpas (1883-06)*	RORTIG11
Ramalho Ortigão	*As Farpas (1883-06)*	RORTIG12

Table 1: List of the Portuguese authors and books used in the text authorship attribution task.

Since most authors have only one book available, each book was split into passages of about 50 KB each. The Gram matrices and the matrix of relative entropy estimations were then computed using the above methods. Each entry in these matrices is indexed by a pair of passages. The Gram matrices give rise to distance matrices via the cosine measure (3). The matrix of relative entropies was symmetrized via the symmetrized Kullback-Leibler divergence. We ignored the fact that this is not a valid metric. As for the weighted all-substrings kernel (see (28)), we let $p_{\min} = p$ vary and fixed $p_{\max} = \infty$ (as in [12]). This means no upper bound on the length of

[2]See `http://www.gutenberg.org`.

the string features.

The nearest neighbor estimator was used to attribute an author to each passage. This was done using the leave-one-out cross validation performance measure: for each passage, the estimated author is the author of the closest passage excluding itself. Tables 2-6 summarize the results.

Author	**No. pass.**	**No. correct attributions**				
		$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 10$
A. Herculano	37	31	35	36	37	26
A. Nobre	2	2	2	2	2	2
C. C. Branco	5	5	5	5	5	5
E. Queiroz	2	2	2	2	2	0
F. Lopes	4	4	4	4	4	4
G. Junqueiro	3	3	3	3	3	3
J. Dinis	30	29	29	29	30	30
J. Mattos	5	5	5	5	5	5
L. Netto	2	2	2	2	2	0
L. V. Camões	6	6	6	6	6	6
P. Silva	6	4	4	4	4	5
N. Tolentino	2	2	2	2	2	2
R. Pina	4	4	4	4	4	4
R. Ortigão	17	17	17	17	17	17
Tot.	**125**	**116**	**120**	**121**	**123**	**109**
Acc. (%)		**92.8**	**96.0**	**96.8**	**98.4**	**87.2**

Table 2: Results of the text authorship attribution task using the p-spectrum kernel for $p = 2, 3, 4, 5$ and 10.

Author	**No. pass.**	**No. correct attr.** ($\lambda = 0.25$)			
		$p = 2$	$p = 3$	$p = 4$	$p = 5$
A. Herculano	37	32	35	36	37
A. Nobre	2	2	2	2	2
C. C. Branco	5	5	5	5	5
E. Queiroz	2	2	2	2	2
F. Lopes	4	4	4	4	4
G. Junqueiro	3	3	3	3	3
J. Dinis	30	29	29	30	30
J. Mattos	5	5	5	5	5
L. Netto	2	2	2	2	2
L. V. Camões	6	6	6	6	6
P. Silva	6	4	4	4	4
N. Tolentino	2	2	2	2	2
R. Pina	4	4	4	4	4
R. Ortigão	17	17	17	17	17
Tot.	**125**	**117**	**120**	**122**	**123**
Acc. (%)		**93.6**	**96.0**	**97.6**	**98.4**

Table 3: Results of the text authorship attribution task using the weighted all-substrings kernel for $\lambda = 0.25$ and $p = 2, 3, 4$ and 5.

One can see that, by fine-tuning some parameters, the kernel-based methods yield the same success rate (98.4%) as the (unparameterized) Ziv-Merhav method. Furthermore, there is not a considerable difference between the performances of the p-spectrum and the weighted all-substrings kernel. There is a compromise in the choice of the parameter p of the two kernel-

Author	**No. pass.**	**No. correct attr.** ($\lambda = 0.50$)			
		$p = 2$	$p = 3$	$p = 4$	$p = 5$
A. Herculano	37	32	36	37	36
A. Nobre	2	2	1	2	2
C. C. Branco	5	5	5	5	5
E. Queiroz	2	2	2	2	2
F. Lopes	4	4	4	4	4
G. Junqueiro	3	3	3	3	3
J. Dinis	30	29	29	30	30
J. Mattos	5	5	5	5	5
L. Netto	2	2	2	2	2
L. V. Camões	6	6	6	6	6
P. Silva	6	4	4	4	4
N. Tolentino	2	2	2	2	2
R. Pina	4	4	4	4	4
R. Ortigão	17	17	17	17	17
Tot.	**125**	**117**	**120**	**123**	**122**
Acc. (%)		**93.6**	**96.0**	**98.4**	**97.6**

Table 4: Results of the text authorship attribution task using the weighted all-substrings kernel for $\lambda = 0.50$ and $p = 2, 3, 4$ and 5.

Author	**No. pass.**	**No. correct attr.** ($\lambda = 0.75$)			
		$p = 2$	$p = 3$	$p = 4$	$p = 5$
A. Herculano	37	35	36	37	36
A. Nobre	2	2	1	2	2
C. C. Branco	5	5	5	5	5
E. Queiroz	2	2	2	2	2
F. Lopes	4	4	4	4	4
G. Junqueiro	3	3	3	3	3
J. Dinis	30	29	29	29	30
J. Mattos	5	5	5	5	5
L. Netto	2	2	2	2	2
L. V. Camões	6	6	6	6	6
P. Silva	6	4	4	4	4
N. Tolentino	2	2	2	2	2
R. Pina	4	4	4	4	4
R. Ortigão	17	17	17	17	17
Tot.	**125**	**120**	**120**	**122**	**122**
Acc. (%)		**96.0**	**96.0**	**97.6**	**97.6**

Table 5: Results of the text authorship attribution task using the weighted all-substrings kernel for $\lambda = 0.75$ and $p = 2, 3, 4$ and 5.

based methods. For the p-spectrum, one can see that a value too high (in the case, $p = 10$) strongly decreases the performance of the algorithm. This upper bound on the choice of p is understandable; in particular, if we set p higher than the length of the greatest substring common to the pair of strings, the kernel becomes zero. On the other side, small values of p discard important features as long substrings. The Ziv-Merhav classifier, in contrast, has "unbounded" capacity of dealing with long substrings, since there are no bounds on the length of the phrases obtained during the cross-parsing.

Curiously, one of the two misclassifications provided by the Ziv-Merhav classifier revealed a strong connection between two texts in the collection. In fact, Alexandre Herculano's book *Opúsculos* is subject of analysis in an

Author	No. pass.	No. correct attr.
A. Herculano	37	36
A. Nobre	2	2
C. C. Branco	5	5
E. Queiroz	2	2
F. Lopes	4	4
G. Junqueiro	3	3
J. Dinis	30	30
J. Mattos	5	5
L. Netto	2	2
L. V. Camões	6	6
P. Silva	6	5
N. Tolentino	2	2
R. Pina	4	4
R. Ortigão	17	17
Tot.	**125**	**123**
Acc. (%)		**98.4**

Table 6: Results of the text authorship attribution task using the Ziv-Merhav method to estimate the symmetrized relative entropy.

edition of *As Farpas*, by Ramalho Ortigão, reason why it is misclassified as having the latter as author. The first paragraph of that edition of *As Farpas* is:

> O sr. Alexandre Herculano acaba de publicar sob o titulo de _Opusculos_ um livro em que, além de uma refutação erudictamente argumentada e inedita da portaria que suspendeu as conferencias democraticas do Casino Lisbonense, se encontram apenas reedições de algumas antigas obras do illustre escriptor.

This fact suggests the ability of information-theoretic approaches to perform tasks such as citation or plagiarism detection.

We used the whole books of each author (listed in Table 1) to build a phylogenetic tree of Portuguese authors, using the ZM method. The idea is to characterize the "style" of each author based on information theoretic methods. The phylogenetic tree in Figure 1 was obtained using the package PHYLIP[3] (phylogeny inference package), which basically constructs a tree by minimizing the net disagreement between the matrix pairwise distances and the distances measured on the tree. Notice the ability of this method to discriminate among 15th/16th century chroniclers (Fernão Lopes and Ruy de Pina), 19th century novelists (Camilo Castelo Branco, Júlio Dinis, Guerra Junqueiro, Eça de Queiroz, Ramalho Ortigão), 19th century essayists or historian authors (Alexandre Herculano, Possidónio da Silva, Júlio de Mattos) and poetry writers from different ages (Luis de Camões, Nicolau Tolentino, Cesário Verde, António Nobre), with the sole exception of Lopes Netto, a 20th century Brazilian novelist who is misplaced too close to ancient poets perhaps due to some similarity between Brazilian Portuguese and European Portuguese near the period of discoveries.

[3]See `http://evolution.genetics.washington.edu/phylip.html`.

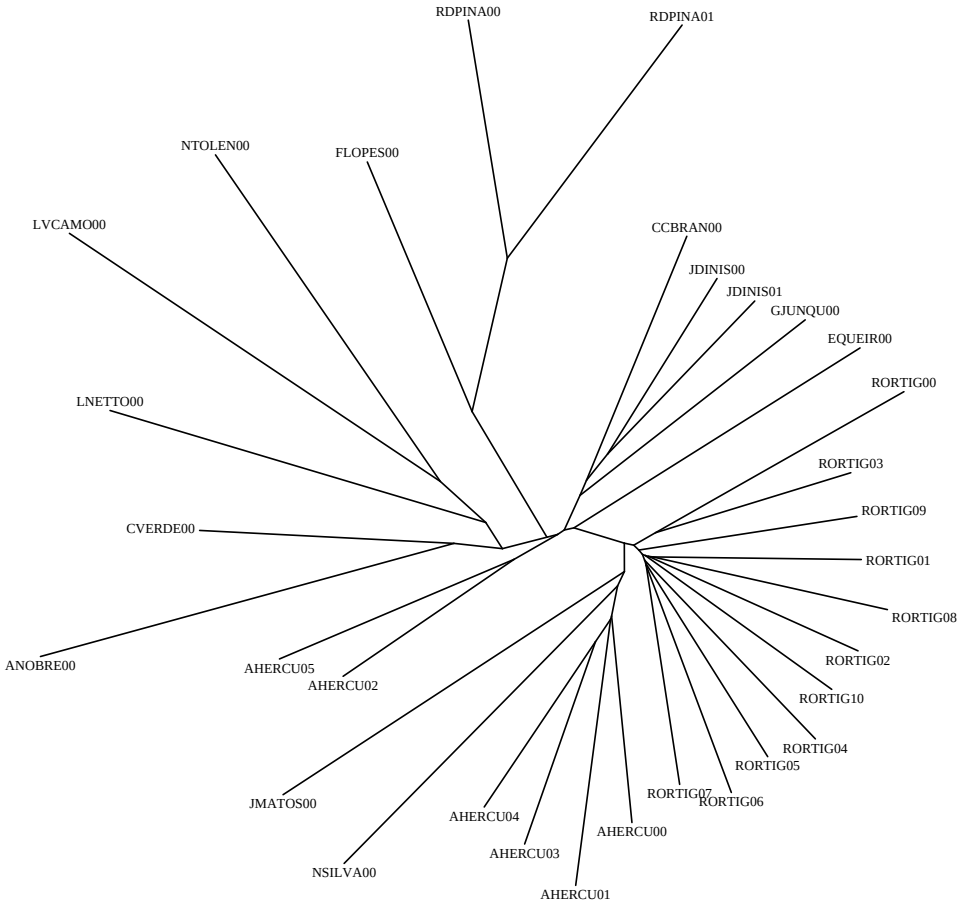

Figure 1: Phylogenetic tree of Portuguese authors and books obtained by Ziv-Merhav method.

5.2 Language detection

For the second task, we used the *Europarl Parallel Corpus* [25], containing multilingual corpora extracted from the proceedings of the European Parliament. In the experiments, we selected 5 parallel documents in 10 languages: French, Italian, Spanish, Portuguese, English, Dutch, German, Danish, Swedish, and Finnish. In average, each document is about 350 KB in size.

The above methods were used to detect the language of each document. We used the same approach as in the text authorship attribution task: a nearest neighbor estimator was used to classify each document with a language: the language of the closest document excluding itself. This was done again using the leave-one-out cross validation performance measure. All methods (p-spectrum kernel and weighted all-substrings kernel

with the same parameters as above, and the ZM estimation of relative entropy) achieved 100% success, i.e., recognized the language of each of the 50 documents.

Finally, we used the dissimilarities computed by each method to build a phylogenetic tree of languages. The goal is to provide some clue about the process of language formation. The language tree obtained with the ZM method is plotted in Figure 2. Notice that this method was able to "discriminate" the Romance family (Portuguese, Spanish, Italian, French) from the Germanic languages (English, German, Dutch, Swedish and Danish) and the Finnish language.

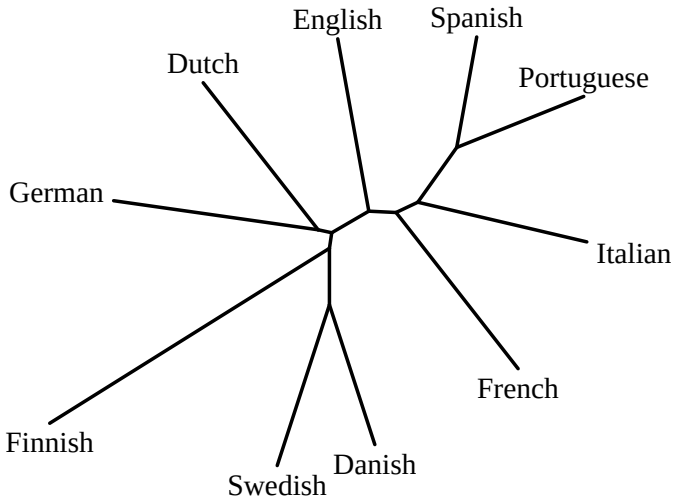

Figure 2: Phylogenetic tree of languages obtained by Ziv-Merhav method.

5.3 Cross-language document matching

A much more difficult task is cross-language document matching. Here, the goal is, for each pair of distinct languages A and B, find the most feasible document in B that is the translation of each document in A. Since each document on each language is tested against all the documents on the other languages, a total of $5 \times 10 \times 9 = 450$ classifications are to be performed. Notice that there is no translation mechanism available in this experience: this is a sort of language-independent document classification. Table 7 shows the results obtained by each method. Clearly, the Ziv-Merhav method outperforms all the other methods. This again emphasizes the adequacy of information theoretic methods to perform this kind of tasks.

Method	No. corr. est.	Acc. (%)
ZM	403	89.6%
PSK, $p = 2$	116	25.8%
PSK, $p = 3$	170	37.8%
PSK, $p = 4$	229	50.9%
PSK, $p = 5$	280	62.2%
WASK, $\lambda = 0.25, p = 2$	126	28.0%
WASK, $\lambda = 0.25, p = 3$	182	40.4%
WASK, $\lambda = 0.25, p = 4$	239	53.1%
WASK, $\lambda = 0.25, p = 5$	280	62.2%
WASK, $\lambda = 0.50, p = 2$	134	29.8%
WASK, $\lambda = 0.50, p = 3$	194	43.1%
WASK, $\lambda = 0.50, p = 4$	256	56.9%
WASK, $\lambda = 0.50, p = 5$	287	63.8%
WASK, $\lambda = 0.75, p = 2$	145	32.2%
WASK, $\lambda = 0.75, p = 3$	216	48.0%
WASK, $\lambda = 0.75, p = 4$	277	61.6%
WASK, $\lambda = 0.75, p = 5$	277	61.6%

Table 7: Comparison of the performance of the above methods on the cross-language document matching task. The total number of estimations is $450 = 10 \times 9 \times 5$. ZM stands for the Ziv-Merhav method, PSK for the p-spectrum kernel, and WASK for the weighted all-substrings kernel.

6 Conclusions

Throughout the paper, a brief survey was presented about techniques based on kernel methods and on information theory for measuring string similarity. Special focus was given on the p-spectrum kernel, the weighted all-substrings kernel, and the Ziv-Merhav method for relative entropy estimation.

A description was provided concerning the implementation of these methods using suffix trees. Furthermore, each was tested on three important problems in the area of text classification: text authorship attribution, language detection, and cross-language document matching. For the first two tasks, all methods yield accuracies close to 100%, provided the kernel parameters are properly fine-tuned. For the last task, the Ziv-Merhav method clearly outperforms the others.

Future work will concern studying further the relationship between (relative) entropy and (generalized) suffix trees, verifying if some information theoretic measures are "kernelizable", and testing experimentally other dissimilarity measures, such as the Jensen-Shannon divergence.

References

[1] Orlando Anunciação. Classificadores baseados em kernels para o reconhecimento de splice sites (draft version). Master's thesis, Instituto Superior Técnico, 2005.

[2] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[3] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. Technical Report LS VIII-Report, Universität Dortmund, Dortmund, Germany, 1997.

[4] B. Schölkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, Cambridge, MA, 2002.

[5] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. CUP, jun 2004.

[6] V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[7] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[8] Ziv and Lempel. A universal algorithm for sequential data compression. *IEEETIT: IEEE Transactions on Information Theory*, 23, 1977.

[9] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

[10] Li, Chen, Li, Ma, and Vitanyi. The similarity metric. *IEEETIT: IEEE Transactions on Information Theory*, 50, 2004.

[11] Christina Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for svm protein classification. In *Proceedings of the Pacific Symposium on Biocomputing 2002 (PSB 2002)*, pages 564–575, 2002.

[12] S.V.N. Vishwanathan and A. J. Smola. Fast kernels for string and tree matching. In K. Tsuda, B. Schölkopf, and J.P. Vert, editors, *Kernels and Bioinformatics*, Cambridge, MA, 2003. MIT Press.

[13] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

[14] Marco Cuturi and Jean-Philippe Vert. A covariance kernel for proteins, October 16 2003. Comment: 12 pages, 1 figure.

[15] David Pereira Coutinho and Mário A. T. Figueiredo. Information theoretic text classification using the ziv-merhav method. In *IbPRIA (2)*, pages 355–362, 2005.

[16] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.

[17] Dominik M. Endres and Johannes E. Schindelin. A new metric for probability distributions. *IEEE Transactions on Information Theory*, 49(7):1858–1860, 2003.

[18] D. Benedetto, E. Caglioti, and V. Loreto. Language trees and zipping. *Physical Rev. Lett.*, 88(4), January 2002.

[19] A. Puglisi, D. Benedetto, E. Caglioti, V. Loreto, and A. Vulpiani. Data compression and learning in time sequences analysis, July 12 2002. Comment: 15 pages, 6 figures, submitted for publication.

[20] Ziv and Merhav. A measure of relative entropy between individual sequences with application to universal classification. *IEEETIT: IEEE Transactions on Information Theory*, 39, 1993.

[21] Ran El-Yaniv, Shai Fine, and Naftali Tishby. Agnostic classification of markovian sequences. In *NIPS*, 1997.

[22] Xin Chen, Sam Kwong, and Ming Li. A compression algorithm for DNA sequences and its applications in genome comparison. In *RECOMB*, page 107, 2000.

[23] Haixiao Cai, Sanjeev R. Kulkarni, and Sergio Verdú. Universal entropy estimation via block sorting. *IEEE Transactions on Information Theory*, 50(7):1551–1561, 2004.

[24] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, September 1995.

[25] Philipp Koehn. Europarl: A multilingual corpus for evaluation of machine translation (unpublished), 2002.