Protein analysis meets visual word recognition: a case for String kernels in the brain

Thomas Hannagan and Jonathan Grainger
Laboratoire de Psychologie Cognitive, CNRS
Aix-Marseille University
3, place Victor Hugo, 13331 Marseille, France
Corresponding author: thomas.hannagan@ens.fr
Keywords: String kernels, Visual word recognition, Orthographic coding, Open-bigrams, Visual Word Form Area.

Abstract
It has been recently argued that some machine learning techniques known as Kernel methods could be relevant for capturing cognitive and neural mechanisms (Jäkel et al., 2009). We point out that “String kernels”, initially designed for protein function prediction and spam detection, are virtually identical to one contending proposal for how the brain encodes orthographic information during reading. We suggest some reasons for this connection and we derive new ideas for visual word recognition that are successfully put to the test. We argue that the versatility and performance of String kernels makes a compelling case for their implementation in the brain.

Introduction
Models of human visual word recognition typically involve a lot of machinery and several processing levels, but it has become clear in the last decade that their behaviour is deeply affected by the coding scheme they use to represent letter order (Plaut et al., 1996). Far from being a defect of such models, the importance of this “orthographic coding” stage appears to mirror the finding that humans devote a well-defined and perfectly reproducible brain area to this end, the so-called Visual Word Form Area (Cohen et al., 2000). In recent years, one challenging task has thus been to determine which of a number of possible coding schemes can best capture human behavioural data, under various blends of parsimony and plausibility constraints. Although existing proposals bear little resemblance to one another, surprisingly, so far it has been very difficult to decide between them solely on the basis of their explanatory power. But here we wish to argue that one coding scheme, Open-bigrams, is also favoured by its generality and computational efficiency.

Indeed one argument that has not yet been fully exploited in this endeavour is the computational argument. Many researchers share the view that to a first approximation
and for all computational intents and purposes, characterising the orthographic code boils down to specifying the similarity between all possible strings of letters\(^\text{1}\). What then are the best tools available to date in order to compare sequences? In the field of machine learning, one powerful and versatile method that has emerged in the last decade is the so-called “String kernels” method. String kernels were originally developed to address questions in bioinformatics (Haussler, 1999), but have now found applications in natural language processing (Lodhi et al., 2002), or more recently in speech recognition (Gales, 2009).

In this article we will first describe the three main alternatives for orthographic coding and assess them on the existing behavioural, neural and computational evidence. We will then explain what String kernels are and how they have been used to tackle problems in natural language processing and computational biology, and will show that each existing variant of the Open-bigram scheme finds an almost identical counterpart in String kernels. We will point to novel insights that become available when the orthographic code is envisioned in this way, and we will present new simulation results on String kernels inspired from computational biology. Finally in light of this unexpected connection we will recapitulate and discuss the case for String kernels in the brain.

Orthographic coding

Orthographic coding is what every reader must do in order to recognise strings of letters. It is hard to see how we could retrieve meaning or pronunciation from a visual word if its representation was randomly assigned anew every time we came across it (for then how could we retrieve the same word meaning on two different occasions), or if we did not somehow keep track even partially of letter order (for then how could we distinguish between, say, anagrams). Yet orthographic coding is still an open problem. As a determinant of what counts as a neighbour for a word it is central in our understanding of the lexical system. But it is also a problem tied to general questions in the visual object recognition literature, such as the interplay between invariance and selectivity.

An intuitive way to represent order in a word (say TIME) would be to assign fixed positions to letters, starting from some origin (for instance, T1, I2, M3, E4). But because letters are exactly tied to positions, taken literally this “slot coding” approach fails to account for robust experimental findings such as transposition and relative position effects. For instance when this scheme is used to present “TMIE” to a state-of-the-art word recognition model like DRC or CDP++ (Coltheart et al., 2001; Perry et al., 2010), M2 and I3 cannot contribute any evidence for word TIME and the model is not driven towards the activation pattern that corresponds to word TIME more than it would have been if say TOBE had been presented. The resulting model behaviour is falsified by experimental data coming from priming experiments showing that TMIE primes identification of the word TIME better than TOBE, so-called transposed-letter (TL) priming (Perea & Lupker,\(^\text{1}\)Some researchers have raised doubts on the relevance of computing similarities outside of a full-fledged dynamical model of visual word recognition. This point is discussed in Hannagan et al. (2011a) where it is concluded that the approach is useful to capture the common bottom-up activation component in all models, which correlates highly with the final outcome.
Another challenging set of results for current models of lexical access is that humans can be remarkably unperturbed by several letter deletions (GRDN-GARDEN), or insertions (GARXYZDEN - GARDEN), so-called relative-position (RP) priming (Grainger et al., 2006; van Assche & Grainger, 2006), and we have no trouble recognising embedded or misaligned substrings (BOBCAT-CAT, Davis & Bowers (2006)).

Determining the signature of the orthographic code has been the focus of much research in the last decade (see for instance Grainger (2008) for a review), but one first and inescapable conclusion is that flexibility is one of its hallmarks.

Proposals

Several proposals have been put forward to increase flexibility in orthographic representations and reproduce this signature. One possible modification would be that in-word positions are not in fact perfectly defined over slots: for each letter the word representation would maintain some measure of uncertainty as to its position (with some letter positions being less uncertain than others). This idea has been formalised and studied in the Overlap model of orthographic coding (Gomez et al., 2008), where letters in candidate stimuli have Gaussian position functions (e.g. stimuli “TRIAL” and “TRAIL” in Figure 1.1, left and middle), whereas known target words have perfectly defined positions (e.g. target “TRAIL” in Figure 1.1 right). The similarity between candidate and target is calculated only from those of the candidate’s letters that are shared with the target, by summing their density functions when integrated within the correct boundaries (i.e. over the slots corresponding to the target letters). In this account, a transposed prime would be less effective than the identity prime because the integration boundaries have been switched, making the areas under curves for I and A smaller for “TRIAL” (Figure 1.1 left) than for “TRAIL” (Figure 1.1 middle). However prime “TRIAL” will still produce more facilitation than say “TRONL” because the former shares 5 letters with the prime against only 3 for the latter.

A second alternative is Spatial Coding (Davis, 2010), which starts by assigning positions to letters (in Figure 1.2, the height of the bar for each letter marks its position value) and where the similarity between two strings also requires the summing of Gaussian density functions across every common letter. However each Gaussian function is now centered on the difference between the positions occupied by the common letter in both strings, and the final similarity score is obtained not by summing the areas under curves, but by finding the peak of the summed curves (not shown). In this account, “TRIAL” is less effective a prime than “TRAIL” because transposing two letters displaces the center of their corresponding Gaussian functions, and the subsequent summing produces a lower peak. Note that unlike Overlap coding, the letter nodes activated by candidate strings are duplicated in several banks (the clones in Figure 1.2), which are wired in finely tuned competitive and cooperative networks where ambiguities produced by repeated letters are assumed to be resolved.

A coding scheme that we will not study here but that bears mentioning because it has appeared several times in the literature in one form or another is the so-called Both-ends coding, where letter positions are encoded from both ends of the word (Jacobs et al., 1998; Fischer-Baum et al., 2010; Cox et al., 2011).
Figure 1. Three orthographic coding schemes illustrated with the comparison of target “TRAIL” (last column) to candidates “TRIAL” and “TRAIL” (resp. first and middle columns). (1) Overlap coding: letter positions are encoded as Gaussian probability density functions (pdf), and similarities between candidate and target strings are computed as the sum of the pdf integrals for letters shared in both strings. (2) Spatial coding: letter positions are encoded by decreasing positional values, and similarities between candidate and target strings are calculated from the sum over each “signal-weight function” for letters shared in both strings (a signal-weight function is a Gaussian like function defined for each common letter, centered on the difference between positional values in both strings). (3) Open-bigram coding: string representations are mapped to a point in the space of all possible ordered letter pairs (bigrams), and similarities between candidate and target strings are obtained by dividing the number of shared bigrams by the number of target bigrams.

Spatial coding are inherently letter-based and absolute schemes (they use coordinates from an origin to define letter positions within the string), but alternatives have been proposed that use information about the relative positions of letters (Mozer, 1987; Whitney, 2001; Grainger & van Heuven, 2003), for instance by keeping track of ordered pairs of letters in the stimulus\(^2\). This amounts to representing letter strings as points in the high-dimensional space of all possible letter pairs (Figure 1.3), and is generally known as Open-bigram coding, a label that covers a range of schemes with different letter gaps and weight parameters. Whitney’s Open-bigram scheme had a restriction on the authorized gap, and weights that decreased with the gap (Whitney, 2001). Grainger and

\(^2\)Schemes with longer n-grams have also been proposed (Wickelgren, 1969; Dehaene et al., 2005).
Van Heuven considered unconstrained and constrained Open-bigrams, that is, respectively with or without gap restrictions (Grainger & van Heuven, 2003). Table 1 shows the Open-bigrams representations of the example word “ROOMS” for each of these schemes.

<table>
<thead>
<tr>
<th></th>
<th>RO</th>
<th>RM</th>
<th>RS</th>
<th>OO</th>
<th>OM</th>
<th>OS</th>
<th>MS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeriolOB</td>
<td>1</td>
<td>0.5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.7</td>
<td>1</td>
</tr>
<tr>
<td>COB</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>UOB</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Word “ROOMS” as represented in three variants of Open-bigram coding: Seriol, Constrained (COB) and Unconstrained (UOB).

The similarity between prime and target is then obtained by a (possibly weighted) count of how many bigrams they share, divided by the target’s norm. Note how only UOB takes bigram “RS” into account, the other schemes imposing a gap restriction of 2 letters. Also bigrams are counted only once, although a recent distributed implementation takes repeated bigrams into account with improved results (Hannagan et al., 2011a).

It might be worth noting at this point that all these string encoding schemes have predecessors in the memory for serial order literature: Slot coding can be compared to Conrad’s seminal Box model (Conrad, 1965), Both-Ends coding to the Start-End model (Henson, 1998), Overlap coding to the Overlap model (Burgess & Hitch, 1992), Spatial coding to the Primacy model (Page & Norris, 1998), and Open-bigrams to the so-called Compound chaining models (Ebbinghaus, 1964; Slamecka, 1985). That models which were designed to capture serial order turn out to be relevant for ordered strings of letters is rather natural, although it would seem to limit the importance of domain-specific aspects for word encoding, such as parallel presentation and large storage requirements. This raises the question of whether these two fields might share more processes than is usually thought—for instance could word recognition involve a serial beginning-to-end process? The single ocular fixation required by skilled readers to recognize a word, combined with the absence of a length effect is good evidence that information about letter identity and location is not extracted from left to right but rather in parallel (Tydgat & Grainger, 2009; Adelman et al., 2010), although probably not all letters at the same pace (Blais et al., 2009). Moreover it would be expected that the necessity to store tens of thousand of words would trigger specific adaptations to general purpose encoding schemes in order to minimize interference for long term storage and recall. In support of this, the processing of random letter strings appears to differ in important ways from the processing of sequences of symbols, as revealed by identification errors (Tydgat & Grainger, 2009).

Indeed Open-bigram, Overlap and Spatial coding describe different orthographic landscapes using two main concepts: an orthographic representation for any string and a similarity measure between two representations. But not all researchers have found it neces-

---

3When weights are involved, only the bigram that contributes the most is considered.
sary to use representations in order to compute similarities. Inspired by string edit theory, Yarkoni et al. (2008) used the Levenshtein distance between strings to build a lexical neighbourhood index that has arguably more explanatory power than the commonly used metric (the so-called N metric), which is based on a single letter substitution radius. Although this is certainly an improvement for the design and analysis of word recognition studies, as it stands the Levenshtein distance falls short of explaining why certain basic transformations (such as deletion) are generally more disruptive than others (such as repetition) —indeed it explicitly assumes that an insertion, a deletion and a substitution have an equal editing cost. This is not to say that a principled explanation for different costs is beyond the reach of string edit models, as edit costs can be chosen on the basis of Minimum message length or Bayesian criteria (Allison et al., 1992; Dennis, 2005). Another option however —and the one chosen in this article— is to assume the existence of an underlying orthographic coding scheme, and let these differences be induced by the string representations thereby generated.

We now assess the behavioural, neural and computational evidence for and against these schemes.

**Behavioural Evidence**

The standard way to gather behavioural evidence on the orthographic code has been through masked priming lexical decision experiments. A robust finding in psycholinguistics is that human decisions on whether a given target string of letters is or is not a word (the so-called lexical decision task) are affected when a subliminal stimulus (a prime) has been flashed just before the target (Forster & Davis, 1984). Because subjects are unaware of the prime, this result is thought to be uncontaminated by conscious processes. What is more, it conveniently varies in strength when subliminal primes are more or less close in form to the target. For these reasons masked priming has become the experimental paradigm of choice to investigate the lexical system on its own.

Overlap coding has been shown to capture a large selection of phenomena observed in a less commonly used paradigm, the same-different judgment task, where participants have to determine whether two briefly presented strings are identical or not (Gomez et al., 2008). However its performance on masked priming data is often problematic as it predicts, for instance, no priming for 9 letter words when the final four letters 6789 are used as primes, and massive priming for 8 letter words with pairwise transposed primes like 21436587 (the so-called T-All condition). Both predictions have been falsified experimentally (Grainger et al., 2006; Guerrera et al., 2008). As for Spatial Coding, it consistently achieves superior agreement with masked priming constraints (see for instance Hannagan et al. (2011a)). In particular, when used as an input scheme in an 18 parameter word recognition model, it provides a very good fit to 61 masked priming conditions (Davis, 2010). Note, however, that the T-All condition can only be accounted for “outside” the input coding scheme, mostly as a consequence of competition at the model’s lexical level. This is because like Overlap coding, Spatial coding operates under the assumption that erroneous letters in the prime do

---

We thank one anonymous reviewer for pointing out this work to us.
not directly contribute negative evidence for the target. Only positive evidence is given by
shared letters, and erroneous letters simply weaken the positive evidence from correct letters
by disturbing their alignment with the target. Finally Open-bigram coding can generally
achieve good correlations with priming data including (when the scheme is implemented
with distributed representations) priming conditions such as reversed primes (4321) or
repeated letters (123245) that have been deemed problematic for this specific coding scheme
(Hannagan et al., 2011a). In summary the three schemes can account for many aspects
of the behavioural data, but as of today only Spatial coding has been studied in a full-
blown word recognition model where it achieves a very good fit to the masked priming data.

Biological evidence

fMRI studies show that faces, tools, houses, and word stimuli activate small
neighbouring cortical patches in the fusiform region, at the frontier between occipital and
temporal lobes (see for instance Hasson et al. (2003); Kanwisher (2006); Gaillard et al.
(2006). This proximal organisation is precisely what one would expect if some general
visual object processing mechanism was at work, with the particular location of each patch
reflecting idiosyncrasies such as the use of ordered symbols for word recognition.

Spatial coding has so far gathered little evidence from brain studies. Davis (2010)
argues that the monotonic activation gradient that is used in this scheme could be
implemented by dedicated neural populations for each letter, that would fire in a periodical
fashion yet consistently out of phase with one another. Learning the orthographic code
for word W would mean learning to delay the correct letter signals so that these initially
asynchronous signals eventually all arrive at the same time at a neural population dedicated
to W. It is argued that a misspelled word would then result in the letter signals being out
of phase, as described in the spatial coding scheme. But this elegant mechanism has yet to
be reported in the visual word form area following the presentation of word stimuli or letter
strings, and it also requires dedicated populations of neurons, a controversial assumption
(Bowers, 2010; Plaut & McClelland, 2010).

As for Overlap coding, its basic premise is consistent with the notion of overlapping
receptive fields which by definition would produce some uncertainty on the retinotopic
location of detected entities. However the uncertainty described in Overlap coding operates
at the level of in-word letter positions, not retinotopic letter locations. Thus, taken literally
this scheme requires the existence of at least two levels: an upper level with banks of
neural populations tuned to all possible letters at all possible in-word positions, that would
receive activity via fan-in overlapping connections from topologically corresponding units
in a lower retinotopic level. Although a retinotopic level and overlapping connectivity are
completely consistent with the primate visual system, the in-word position level remains
speculative. While location-specific letter detectors have been successful in theoretical
accounts of crowding effects (Tydgat & Grainger, 2009) and are supported by fMRI data
(Dehaene et al., 2004), there is so far no evidence coming from brain studies for banks of
location-invariant, but position-specific, letter detectors.
Turning to Open-bigram coding, Binder et al. (2006) found that activity in the VWFA was correlated with the frequency of (contiguous) bigrams in letter strings presented to subjects. Using similar methods, Vinckier et al. (2007) further found evidence for a posterior-to-anterior gradient of activity in the VWFA that correlates with letter combinations of increasing complexities, in a hierarchy involving an Open-bigram level (with gap 1 bigrams). This evidence supports a biologically inspired account of orthographic coding known as the Letter Combination Detector proposal (LCD), which, based on the general architecture of the primate visual system, hypothesized the existence of such a hierarchy (Dehaene et al., 2005). Indeed these findings may not be surprising considering the ample evidence provided by electrophysiological studies for the existence of neurons that code for invariant feature combinations in generic object recognition (Tsunoda, 2001; Kobatake & Tanaka, 1994).

In summary biological plausibility supports Open-bigram coding. Not only is there direct evidence for this scheme in the VWFA, but as predicted from theoretical considerations it uses the same basic mechanism as apparently used by object recognition in general, which is to build representations from collections of feature combinations.

**Computational evidence**

Computational models have more evidence to offer than just fits to the data, and indeed selecting a model exclusively on this criterion would expose one to the dangers of overfitting (Pitt & Myung, 2002). Even when considering two models with the exact same goodness-of-fit, one of these might take more time or more memory to run, or it might be operating at the limits of its computational abilities, or be using complex algorithmic contraptions and a host of parameters, or simply be much harder to learn. Formal notions have been proposed to quantify each of these four criteria, respectively time and memory complexity, complexity class (van Rooij & Wareham, 2008), parsimony as defined by Kolmogorov complexity (Li & Vitanyi, 1997), and learnability (Valiant, 1984). In this section we shall only consider informally how the different proposals fair in some of the previously cited respects, but model selection has been a very lively topic of research in recent years with several principles now available to decide between models (Pitt & Myung, 2002; Grünwald, 2005).

First let us look at how many parameters are being tuned in the three coding schemes. In order to fit behavioural data, Overlap coding uses as many parameters as the longest string involved in the comparison: each parameter determines the variance of the Gaussian uncertainty function of a letter. Instead, Spatial coding uses a two-parameter formula that computes the variance for each letter given its position, and one parameter to specify the importance of end letters. The type of Open-bigram coding used in Whitney’s Seriol model also involves edge bigrams, and 2 different bigram weights that depend on letter contiguity. Finally, one parameter is tuned in Grainger and van Heuven’s versions (the letter gap which is either fixed or absent).

Let us now consider what kind of computational mechanisms are at play in these
schemes. Overlap and Spatial coding share one critical feature, which is that similarity calculations require the prior knowledge of which letter from the target should be mapped to which letter from the candidate. For strings of equal length, this has been tackled by assuming that only the same letters at the same position are considered for comparison (an intuitive although strong hypothesis in itself). But this becomes problematic for strings with different lengths and repeated letters, because there is no easy way to tell how this mapping should proceed in general. Simply consider, for example, the candidate TENT and target METER: Overlap coding does not tell us which of the two very different uncertainty functions for T in the candidate string we should use to compute the similarity with T in the target\(^5\). Spatial coding faces the same issue, but proposes a complex solution that involves duplicated letter banks for each position (the so-called letter clones), competitive-cooperative interactions between and within banks, and the invocation of a new principle —the principle of clone equivalence. In Open-bigram coding, no prior knowledge of the letters to be compared is assumed, and repeated letters are not a problem when computing similarities because the basic unit of comparison, the bigram, does not have a position anymore: only its presence in both representations counts.

Computational evidence should also be sought on how these coding schemes can be learned. Computational models of object recognition have shown that units tuned to feature combinations can arise from training hierarchically organised layers of neurons with limited receptive fields (Wallis & Rolls, 1997), possibly with interleaved layers of simple and complex cells as found in the ventral route (Riesenhuber & Poggio, 1999). Nevertheless a neuro-computational demonstration that Open-bigrams can be learned is still unavailable, and neither is there a computational model that learns Spatial coding. In this respect Overlap coding might arguably have the advantage, given that a backpropagation network trained for location invariant word recognition has recently been shown to implement what can be described as a distributed version of this specific coding scheme (Hannagan et al., 2011b).

In summary the time and memory complexities of these schemes are still unknown, but a crude assessment of the number of parameters and the parsimony of the mechanisms involved is in support of Grainger and van Heuven’s Open-bigrams. We have also identified defects in the current specification of Overlap coding that limit its computational power relative to Open-bigram and Spatial coding. However learnability might still support a scheme in the spirit of Overlap coding. We now proceed to strengthen the computational evidence in favor of Open-bigram coding, by pointing out its relationship with a powerful machine learning tool known as String kernels.

String kernels

Standard data analyses are typically carried out on vectors, but most complex real life phenomena do not fall naturally into this category. Raw input data must usually go through a preprocessing stage before they can be suitably analysed, which raises the

\(^5\)Such conditions are not covered in Gomez et al. (2008)
Figure 2. 1. Illustrating how String kernels can support decision processes in three different research domains. (A) Bioinformatics: Protein function prediction using String kernels on sequences of amino-acids. (B) Cognitive science: Lexical decision using String kernels on short letter sequences (C) Natural language processing: Text classification using String kernels on concatenated words.

2. Once sequences have been mapped to the space of domain-specific feature combinations, decision processes such as Support Vector Machines or Neural Networks can be trained or tuned to perform classification tasks.

questions of how to select relevant features and how this selection will affect subsequent stages. Kernel methods provide a general way to tackle this issue, which is especially relevant to cognitive science (Jäkel et al., 2009). The gist of Kernel methods is to map the initial structure onto a feature vector that lives in a vastly higher dimensional space than the original structure — for instance the space of all possible feature combinations of a given length. The high-dimensionality somewhat alleviates the feature selection step\(^6\), and the vectorial nature of feature space then enables standard vector analyses to operate on real-life discrete structures such as trees, graphs, and most relevant to this article, sequences (Hofmann et al., 2008).

More formally, a kernel is a function that takes two structures and returns an inner product between the mapped structures in some feature space (Jäkel et al., 2007). The feature map is chosen so as to facilitate the separability of the mapped structures, and is

---

\(^6\) Especially when the so-called “regularization” is applied
usually of high-dimensionality. Because of this, in general the explicit computation of all
the mapped structures’ coordinates is prohibitive, but one strength of kernel methods is
that for astutely chosen feature maps the inner products can nevertheless be calculated
implicitly at a much smaller cost (the so-called “Kernel trick”\(^7\)). Note however that the
String kernels of interest in this article are simple ones where the explicit computation of
coordinates is actually feasible.

We will now illustrate the performance and versatility of String kernels by reporting
on the results obtained by two recent studies on problems that arise in Natural language
processing and in Bioinformatics.

**Text classification**

Human activities are increasingly mediated by the Web, and the ability to categorise
text content in a quick and accurate way has become of utmost importance when designing
search engines. This is for instance required to automatically prevent spurious “webspam”
results from invading google search results, or simply to avoid spams in one’s mail box.
In Lodhi et al.’s string subsequence kernel for text classification (Lodhi et al., 2002), letter
strings generated from a given alphabet \(\sum\) are compared in two steps. First, each letter
string is mapped onto a vector of dimension \(\sum^k\) where each component registers a possible
sub-string of size \(k\) (i.e. an ordered combination of \(k\) letters), weighted by a given contiguity
factor \(\lambda\) that diminishes exponentially with the gap. For example, a 2-kernel mapping for
words TIME and MEMES would produce two vectors of dimension \(26^2\). In this case each
vector would be very sparse, consisting mostly of zeros except for the few components that
respectively correspond to existing substrings in TIME and MEMES, as described in Table 2.

\[
\begin{array}{cccccccccc}
TI & TM & TE & IM & IE & ME & MM & MS & EE & ES \\
\phi(TIME) & \lambda^2 & \lambda^3 & \lambda^4 & \lambda^2 & \lambda^3 & 0 & 0 & 0 & 0 \\
\phi(MEMES) & 0 & 0 & 0 & 0 & 0 & 2\lambda^2 + \lambda^4 & \lambda^2 & \lambda^3 + \lambda^5 & \lambda^2 & \lambda^3 \\
\end{array}
\]

Table 2: A 2-String kernel for words TIME and MEMES.

Note that the dimension of the feature vector does not depend on string lengths, and
in case of repeated letter combinations, weighted contributions are simply added up in the
corresponding feature vector component. The kernel value between both strings is then
obtained by taking the inner product of the normalised feature vectors. Assuming \(\lambda = 1\)
for simplicity:

\[
K_2(TIME, MEMES) = \frac{\phi(TIME)^t \cdot \phi(MEMES)}{||\phi(TIME)|| \cdot ||\phi(MEMES)||} = \frac{1 \cdot (2 + 1)}{\sqrt{6 \cdot 1^2} \cdot \sqrt{3^2} + 3 \cdot 1^2 + 2^2}
\]

\(^7\)In the machine learning literature this ‘Kernel trick” has come to be associated with the mere notion of
a kernel function, although it is not involved in its actual definition.
When applied to entire webpages rather than single words for example, Lodhi et al.’s string subsequence kernel simply treats the text as a long string of concatenated words and spaces (without punctuation and stop words such as “the” and “at”). The resulting feature vector is still of the same dimension—or rather, of dimension \((26 + 1)^k\) because of the blank character—but it is much denser. The similarity between two strings of lengths \(S\) and \(T\) can nevertheless be computed in linear time of \(k, S\) and \(T\).

The authors used this kernel to train a Support Vector Machine to perform binary classifications on a sample of documents taken from the Reuters dataset (Lewis, 1987). SVMs are supervised binary classifiers that operate in conjunction with some kernel methods (Cristianini & Shawe-Taylor, 2000). When presented during training with exemplars of two categories that have been mapped onto feature space, an SVM tries to find a hyperplane that separates exemplars using a principle of maximum margin distance (illustration of the “decision process” in the right hand side of Figure 1). The SVM is then tested for its generalisation performance in a subsequent phase, where it is presented with new exemplars of both categories. Performance is assessed using standard quantities such as precision, recall and \(F_1\)\(^k\). 470 documents were chosen from four of the most frequent categories in the Reuters database. 380 documents were used during training and 90 during the test phase. The SVM achieved state-of-the-art performance in binary classification, as measured by \(F_1\) scores when compared to two SVMs that used other kernels (n-gram or word kernels). When averaged across all categories, performance also peaked for subsequences of length \(k = 5\) or for low values of the gap penalty \(\lambda\).

### Protein classification

Coming from computational biology, remote protein homology detection is another problem where researchers have found it useful to use String kernels. Consider the SCOP database (Murzin et al., 1995), a registry of proteins for which both structure and function is known, organised hierarchically into families, superfamilies and folds. Two proteins are deemed remote homologs if they belong to different families but have the same superfamily. The remote protein homology problem consists in distinguishing between homologs and non-homologs, among a sample of proteins that span a number of families.

To tackle the remote protein homology detection problem, Leslie & Kuang (2004) proposed a “restricted gappy” String kernel that uses the same feature space as Lodhi et al (that is a feature space indexed by all letter combinations of size \(k\) in the alphabet) but where only a restricted number of gaps is allowed when matching to the subsequence of the target string. Let us assume an alphabet \(\sum\), a string \(S\) written in this alphabet, and a chunk \(\alpha\) of \(g\) contiguous letters in \(S\). The feature map \(\phi_{g,k}\) for \(\alpha\) is a vector of \(\sum^k\) components, with each component \(i\) standing for a possible combination of \(k\) letters in \(\sum\).

\[F_1 = 2 \ast \text{Recall} \ast \text{Precision} / (\text{Recall} + \text{Precision}),\] where \(\text{Precision} = \text{True Positives} / (\text{True Positives} + \text{False Positives})\) and \(\text{Recall} = \text{True Positives} / (\text{True Positives} + \text{False Negatives})\).
and is defined in the following way:

\[ \forall i \in \sum^{k}, \phi_{g,k}^{i}(\alpha) = \begin{cases} 
1, & \text{if } i \text{ appears in } \alpha \text{ with up to } g-k \text{ gaps;} \\
0, & \text{otherwise.} 
\end{cases} \]

The feature map for any string \( S \) is the sum of the feature maps for all \( \alpha \) in \( S \). This is effectively equivalent to computing all feature maps for a window of \( g \) letters that is sliding across string \( S \). Again, the \((g,k)\)-gappy kernel between strings \( S \) and \( T \) is then simply defined as the normalised inner product of the feature maps for \( S \) and \( T \), that is:

\[ K_{g,k}(S,T) = \frac{\phi_{g,k}^{S}(T)^{t} \cdot \phi_{g,k}^{T}}{||\phi_{g,k}^{S}(T)|| \cdot ||\phi_{g,k}^{T}||} \]

Leslie and Kuang also envisioned a system of gap penalties very much in line with Lodhi et al, whereby substrings are assigned exponentially large penalties as a function of the gap with which they occur in the string.

The authors then used the restricted gappy kernel as a base to train an SVM on the remote protein homology problem. During training, the SVM was presented with exemplar proteins that belonged to several possible families (with the exclusion of one, the target family) within a given superfamily, as well as non-exemplar proteins coming from other superfamilies. Once satisfactory classification performance was reached on the training base, the SVM was tested on new proteins coming from the target family. Here assessment was carried out using the Receiver Operating Characteristic with a standard Area Under Curve score (AUC), which is another performance measure for classifiers\(^9\) that returns 1 for perfect classification, and 1/2 for a random classification. With an AUC of 0.851 over 54 target families, the SVM exhibited good performance relatively to SVMs trained with other kernels. Performance was essentially similar with or without gap penalties, and was also correlated with an a priori similarity score defined between the kernel and the target domain. In other words, SVM performance on a given training base could be estimated a priori from the restricted gappy String kernel.

**Open-bigrams as String kernels**

There are two small but significant differences between Open-bigrams and String kernels. The first is that repeated letter combinations are added up in String kernels (feature vectors take integer values), but not in Open-bigrams (feature vectors take binary values). The second difference is that similarities are symmetric in String kernels, but not in Open-bigrams, where one distinguishes between candidate (prime stimulus) and target strings and the similarity is given by the strings’ inner product divided by the squared target’s norm. Except for these two differences, the reader will have

---

\(^9\)Receiver Operating Characteristic assesses how the ratio of true positives on false positives is affected by variations in the classification threshold.
recognised that Leslie and Kuang’s Restricted gappy kernel with $g = 4$, $k = 2$, and a non-exponential gap penalty is essentially Whitney’s weighted Open-bigram scheme. Without gap penalty, we recover Grainger and van Heuven’s constrained Open-bigram scheme. Finally, Lodhi et al’s substring kernel with $k = 2$ is the unconstrained Open-bigram scheme.

A remarkable fact about the literatures on String kernels and Open-bigrams is that they do not cross-reference at all. It thus appears that the same technique has been introduced in distinct scientific areas at about the same time, unbeknownst to their actors. This is yet another example of parallel meme evolution, and a modest addition to an already long list of simultaneous scientific findings. What is perhaps striking in the String kernels/Open-bigrams case is how this parallel evolution took place in disciplines driven by quite different pressures: efficiency of prediction/analysis for computational biology and natural language processing, but fidelity to human behaviour and neural mechanisms for cognitive science.

What is it that String kernels seem to capture in protein prediction, in text classification, and in visual word recognition? Could it be one and the same thing? Indeed it could be argued that there are really two main ideas at the origin of String kernels: the idea of mapping structures onto high-dimensional vectors, and the idea that in so doing vector components should stand for feature combinations. How can we tell whether the success of String kernels in different fields comes from one idea or the other, or both?

Happily, it turns out that researchers in natural language processing have studied ways to map documents onto high-dimensional vectors but without using feature combinations, using so-called “Bag of Words” approaches. In Latent Semantic Analysis (Landauer & Dumais, 1997; Joachims, 1998; Landauer et al., 2007) for instance, a text database is turned into a large matrix where lines represent words, columns represent documents, and each matrix entry is weighed proportionally to the word frequency in the document and to its inverse frequency in the database. Singular value decomposition is then used to produce a single large vector for each document, where each component reflects the importance of a word in the document (rather than a combination of words or a subset of letters). This scheme has been quite successful and indeed illustrates that the mapping of structures onto high-dimensional spaces is a key to success in classification, as expected from Cover’s theorem (Cover, 1965). However Lodhi et al showed that the substring kernel outperformed the Bag of Words approach on all the categories it had been applied to —although it took longer to compute. The conservative expectation to form for other fields, then, would be that beyond the high-dimensionality aspect, it is really the idea of representing structures as relationships between their parts that makes a difference.

We would argue that this key concept of letter combinations used in String kernels is successful across fields because it provides a way around similar issues: tolerating global displacements and local changes when comparing two sequences. In the domain of protein analysis for instance, non-contiguous combinations are useful to reflect the fact that strings

---

10 We thank one anonymous reviewer for pointing out that the “Follows relation” in computational linguistics also encapsulates this same idea of distant ordered relationships between elements.
of amino acids that share displaced subsequences will probably have similar functions, since they likely evolved from a common string by way of insertion or deletion of an amino acid. In orthographic coding, non-contiguous combinations can be used to represent words independently of their location on the retina, or in other terms, to evoke the same representation despite global displacements of a given stimulus. Although the general class of string kernel methods has some generality, this is not to say however that the success of a particular string kernel in one domain automatically guarantees its success in another. The tuning of parameters such as substring length and gap penalty will be dictated by the specifics of each domain, and in particular the size and number of all sequences that are to be distinguished as well as the statistical regularities between them. For proteins and text classification large sequence sizes call for larger substrings and the underlying regularities are best captured by exponential gap penalties, whereas in the word domain a smaller grain of letter combinations is more effective and the gap penalties that have been proposed are relatively weak.

**Insights and tools from String kernels**

The classification of Open-bigram coding as a String kernel method comes with a number of insights.

**Formalism.** For one thing, String kernels have been specified in a more formal way. The mathematics are simple and require only a mapping function from input to feature space (what we have referred to as a feature map), and a kernel function that returns the similarity between any two points in feature space. The requirement on the kernel function is that it satisfies Mercer’s conditions (i.e. it must be symmetric and positive semidefinite). In all the kernels we have considered, the kernel function is the normalised inner product which automatically satisfies Mercer’s conditions. However Open-bigram schemes do not comply to this requirement because in general distances are not symmetric (similarities only involve the target’s norm), and we will see that restoring symmetry promotes better performance.

**Computational complexity.** A strong appeal of String kernels is that they can be computed without actually representing structures in feature space, using string alignment and dynamic programming techniques (Lodhi et al., 2002) or data structures like Tries (Leslie & Kuang, 2004). Given two strings S and T and noting |·| as the length, Lodhi et al’s k-substring kernel can be computed in \(O(k|S||T|)\) time, whereas Leslie and Kuang’s restricted gappy kernel is computed in \(O(C_{g,k}(|S| + |T|))\), where \(C_{g,k}\) is a constant (in the sense that it is determined by the kernel’s g and k parameters, not by string lengths or by alphabet size). In other words, this means that similarities in all Open-bigram schemes can be computed in linear time with respect to string lengths. An important point is that because structures do not need to be explicitly mapped to feature space in order for the kernel to be computed. Computation time for all kernels does not depend on the dimensionality of the feature space. Hence just as much time is needed to compute a String kernel for dna sequences, where the alphabet is restricted to 4 nucleotids, than for proteins.
or for words. Finally results are also available on how to approximate String kernels, and how to implement them as a particular kind of automata (transducers, Leslie & Kuang (2004)).

However such complexity analyses could be seen as irrelevant to the brain, given first that they do not involve the actual calculation of coordinates in feature space, whereas we have argued that the brain performs this mapping explicitly, and second that they are derived from the assumption of sequential processes (automata and Turing machines), whereas the neural machinery operates in parallel. For these reasons, demonstrations coming from the branch of complexity theory, called circuit complexity, would appear to be more directly relevant, especially with a view to an actual implementation of the feature map using standard connectionist networks (Shawe-Taylor et al., 1992). We would simply argue that determining the circuit complexity in which String kernels fall could be facilitated by these results and the existing theorems between sequential and parallel complexity theory (Papadimitriou, 1993). Note that a related reason why one should not discard sequential complexity results is that as we have seen, not all orthographic coding schemes claim a proximity to the neural substrate or assume fully parallel computations (Spatial Coding for instance requires the rapid sequential scan of the string in order to assign the activation gradient). The bottomline is that orthographic coding schemes have been implemented as computational models, but strikingly, until today no information was available on the computational complexity of these schemes. We think that the results on String kernels are a first step toward this end.

New ideas for visual word recognition. A first insight from String kernels concerns the use of a gap penalty in Open-bigram coding. Lodhi et al noticed that weak gap penalties in the substring kernel/unconstrained Open-bigram scheme allowed successful classification on text corpora that contained polysemic words, that is words with the same orthography but different meanings. This is because polysemic words tend not to be surrounded by the same words, something that is picked up by taking more context into account (weak gap penalties). Applying this insight to orthographic coding suggests that on the contrary, languages where many semantically related words are also orthographically related would need to use a relatively high gap penalty in order to keep words well separated in orthographic feature space and thus facilitate access to semantics. This would be the case, for instance, in agglutinative languages such as Turkish or Basque.

Another insight is that some String kernels have been described and tested that have currently no counterpart in visual word recognition, although they might well be relevant for it. One example of a potentially interesting kernel is the so-called “wildcard” kernel of Leslie and Kuang. Starting from a normal substring kernel, the idea is to introduce a wildcard character “*” in the alphabet, that matches all characters. That is, feature space is expanded to dimension $(\sum +1)^k$ (or if the blank character is also included, $(\sum +2)^k$). When $k = 3$, this means that in the feature vector for word TIME, all components that correspond to $*I*M$, $TI*$, $T*I*M$, $IM*$, and so on would be incremented. Anticipating some interesting new behaviour for Open-bigram and trigram coding, we can propose
a gappy version of a wildcard kernel (a Gappy 3-wildcard Kernel) where string TIME activates feature components (*IM, T*M, TI*, *IE, T*E, TI*, *ME, T*E, TM*, *ME, I*E, IM*). This scheme is simply obtained from Grainger and van Heuven’s (2003) UOB scheme—henceforth GvH UOB—by substituting the wildcard character at every possible position in each bigram, and is tested in the following section.

Simulations of String kernels. We tested the GvH UOB scheme (without repeated bigrams and with asymmetrical similarities), the Gappy String kernel from Leslie and Kuang, the Spatial Coding scheme (rather than the Spatial Coding model) and the Gappy 3-wildcard Kernel described above, on two recent benchmarks in the orthographic coding literature: Davis (2010) and Hannagan et al. (2011a). Overlap coding was not included in these simulations due to the previously mentioned under-specification on how to calculate similarities when letter repetitions are involved. These two benchmarks are somewhat complementary in that the first is more exhaustive but only provides correlations to human data, which are susceptible to being artificially improved when schemes largely underestimate similarities, whereas the second involves fewer conditions but assesses what type of relationship must hold between them so as to satisfy a number of constraints consistently reported in the literature. We selected all the conditions from Davis (2010) that involved standard masked priming experiments and that did not manipulate frequency and neighbourhood, since these were not relevant to assessing orthographic coding schemes which are blind to these factors. There were 45 such conditions out of the original 61 in Davis (2010), each corresponding to a point in the correlation graphs with human data shown in Figure 3.

Table 3 shows how these 4 schemes satisfy the constraints from Hannagan et al. (2011a), which we briefly describe here: no string representation should be more similar to another than to itself (stability constraint), a letter substitution occurring at the edge of a string should be more disruptive than in the middle (edge effects), transposing two adjacent letters should be less disruptive than substituting them altogether (local TL, where TL refers to “transposed letter”), transposing two distant letters should be more disruptive than substituting one of them, but less so than substituting both (distant TL), a substitution combined with a transposition should be more disruptive than a single substitution (Compound TL), transposing all adjacent letter pairs in a word should be maximally disruptive (global TL), removing letters from a string should conserve some similarity (distinct RP, where RP refers to “relative position”), and finally in a string with two repeated letters it should be equally disruptive to remove one of the repeated letters than one of the non-repeated letters (repeated RP). A detailed presentation of these constraints can be found in Hannagan et al. (2011a).

Figure 3 shows a good general agreement of simulations with human data but also significant differences between schemes, with the lowest correlation starting at 0.67 for UOB, then Spatial Coding (0.73), Gappy String kernel (0.75), and finally the 3-Wildcard kernel which achieves the best correlation at 0.8. This variability is amplified in the Hannagan et al. (2011a) benchmark reported in Table 3, where the ranking is changed to
Figure 3. Similarity values between four coding schemes and human priming data for 45 conditions selected from Davis (2010). Upper left: GvH UOB scheme. Upper right: Gappy String kernel. Lower left: Spatial Coding scheme. Lower right: A Gappy 3-Wildcard Kernel.

<table>
<thead>
<tr>
<th></th>
<th>GvH UOB</th>
<th>Gappy String kernel</th>
<th>Spatial Coding</th>
<th>3-Wildcard Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stability</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Edge effects</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Local TL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Distant TL</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Compound TL</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Global TL</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Distinct RP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Repeated RP</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3: Performance of the coding schemes on the constraints from Hannagan et al. (2011a).

UOB (2 constraints met), Gappy String kernel (3 constraints), 3-Wildcard (6 constraints) and finally Spatial Coding (7 constraints). One might wonder why Spatial Coding ranks only third on Davis (2010) but first on Hannagan et al. (2011a). This is because the edge constraint is only met in Spatial Coding by making the explicit but ad-hoc specification
of an End-of-Word parameter that increases the weight of edge letters. Without this additional degree of freedom, Spatial Coding and 3-Wildcard kernel would both satisfy 6 constraints, restoring the 3-Wildcard kernel to the first rank as for the Davis (2010) benchmark.

Our benchmarks also agree on two remarkable points. First, UOB is always outperformed by the Gappy String kernel —or in other words UOB is always improved by taking into account repeated bigrams in the string representation and by switching to symmetric similarities between strings. This is mostly because in so doing the superset conditions (i.e. letter insertions and repetitions that participate to the stability constraint and to Davis’s benchmark) are assigned less than perfect similarities, which restores stability in the scheme and boosts the general correlation to priming data. The second and perhaps most remarkable common point is that both schemes are always outperformed by the 3-Wildcard kernel. In addition to the 3 constraints already satisfied by the Gappy String kernel, switching to a 3-wildcard kernel now allows one to account for edge effects, for two types of transposition effects (distant and compound TL), and it overall raises the correlation with priming data to the 0.8 level.

Detailed explanations on why Open-bigram coding succeeds or fails on different constraints can be found in Hannagan et al. (2011a), and because the mechanisms there described for Open-bigrams are also relevant to the wildcard kernel every time these schemes agree, here we will focus on the constraints where they behave differently: i.e. edge effects, distant TL and compound TL. Edge effects emerge in the 3-Wildcard kernel as a result of taking into account repeated combinations: in the vector that represents word “TIME”, components for TI* and *ME have a value of two —the number of possible wildcard matches— whereas the component for IM* only has a value of one. Consequently, when similarities are computed, the components that involve edge letters tend to contribute more than others to the inner product, and consequently stimuli like “AIME” or “TIMA” are deemed less similar to TIME than TIPE is (the definition of edge effects). Counter-intuitively though, the reason for the superior performance on certain transposition effects also involves edge letters. For instance, both UOB and Gappy String kernels fail on the distant TL constraint because they wrongly imply that a single substitution will be more disruptive than a transposition, since the former condition disturbs more units in the string representation than the latter. That much is also true of the 3-wildcard kernel, but now the difference is that the act of transposing letters C and E in string “ABCDEF” also transposes the values for e.g. A*C and A*E (1 and 3 respectively) in the representation vector. When computing similarities, the contributions for those two components sum up to \(1 \cdot 3 + 3 \cdot 1 = 6\), rather than to \(0 \cdot 1 + 3 \cdot 3 = 9\) as they would in the case of a single substitution. Similar but smaller trade-offs take place for all units involved in representing the modified letters, and aggregate to produce a distant TL effect. The deeper cause for this effect is that introducing a wildcard character also induces a natural count of letter positions in the manner of both-ends coding (Jacobs et al., 1998; Fischer-Baum et al., 2010): in our example with string “ABCDEF”, the vector components for A*C and C*F have values 1 and 2 respectively, i.e. the number of letters that are standing between C and the first letter, and between C and the last.
Discussion

The case for String kernels in the brain

We have shown that String kernels are powerful machine learning algorithms, performing at a state-of-the-art level on problems coming from different fields. But if String kernels provide a performant and versatile machine learning tool, can we conclude that they are part of the solution found by the brain to represent visual words?

A first reason to refrain from making conclusions at this point would be that while the performance of String kernels has been demonstrated on machine learning benchmarks, Overlap or Spatial Coding have so far only been tested in the field of visual word recognition. Short of the correct comparison, how can we conclude that String kernels are more versatile than either of these two schemes? We think that such prudence would be excessive in this case. First, the fact that machine learning researchers did not come up with letter-based and absolute coding alternatives suggests that keeping track of the position of all constituents within the sequence, whether by using probability density functions like in Overlap coding or by differences of positional gradients as in Spatial coding, is not an efficient way to represent sequences. More importantly, we have seen that significant additional assumptions are required in Spatial and Overlap coding in order to compare strings of different lengths or to deal with repeated letters (Gomez et al., 2008; Davis, 2010). These assumptions are already problematic when dealing with short sequences like words, but they would presumably become impractical when dealing with large sequences like proteins or webpages, where the number of letter repetitions grows at least linearly with the string. In any case, what we can safely conclude for now is that Open-bigram coding provides a way to represent sequences that is performant and extends largely outside of cognitive science. These are remarkable properties that are not yet known to be true of Spatial or Overlap coding, but the burden of proof remains on the proponents of these schemes to show that they can outperform String kernels on machine learning benchmarks. Until then, versatility and performance, as measured by the standards of machine learning, are on the side of String kernels.

If String kernels are indeed more versatile and better performing than other coding schemes, what does this tell about their possible implementation in the brain? One could argue for instance that the beginning reader faces one neural constraint that has no counterpart in machine learning: his brain must start not from a blank slate but from existing visual processing machinery (Dehaene & Cohen, 2007). However, we have previously mentioned the evidence showing that generic object recognition already involves feature combinations. Starting from this way to represent visual objects, the String kernels alternative would appear much easier to achieve than the other two approaches, because it “only” requires an adaptation of feature combination neurons so that they become letter combination detectors that are sensitive to order. Thus if this adaptation is not too prohibitive in terms of computational cost, the simplest solution for the brain will also turn out to be the best. Despite encouraging sequential complexity results, there is currently no
available result on the cost of computing String kernels in a parallel architecture, let alone on the learnability of the suggested adaptation as opposed to others. However in support of a learnable and cost-efficient adaptation, we have already mentioned how activity in the VWFA has been found to be correlated with Open-bigram frequency (Vinckier et al., 2007).

Finally, granted that String kernels are both more performant and easier to learn by the brain, why should we discard the behavioural evidence that has seemed to favour other schemes lately? To address this question we have presented new simulations on two up-to-date behavioural benchmarks. These simulations show that when Open-bigram coding fully complies to the requirements of String kernels (as to how to handle letter repetitions and how to compute similarities), the performance of this scheme is improved on all benchmarks and is on a par with Spatial Coding on one benchmark, despite requiring less parameters. Perhaps more importantly, in this paper we have introduced a gappy 3-Wildcard kernel inspired from computational biology, and we have shown that it outperforms or matches all the other schemes it was tested against. Although these simulations left some phenomena aside, because they involved experimental manipulations that are beyond the scope of input coding schemes, at the very least they imply that the behavioural evidence should now be reconsidered.

String kernels and the dual-route model of orthographic coding

So far we have been relatively silent as to how, where and when String kernels could be implemented in the brain. We have also left aside the issue that Open-bigram coding appears to be ill-suited for computing grapheme-to-phoneme correspondences, since this arguably requires knowledge of which letters are next to which, such as knowing that the T is just before the H in the complex grapheme "th" (Goswami & Ziegler, 2006). A recent theoretical proposal, illustrated on Figure 4, addresses these issues by suggesting that there are two distinct pathways to word recognition—a chunking and a diagnostic route (Grainger & Ziegler, 2011). We will now briefly describe this proposal and argue that a generic feature space could differentiate during development in order to support the kinds of schemes that are hypothesized in both routes.

After the alphabet has been mastered, the next challenge children usually face in learning to read is to associate letters and short letter sequences to sounds. Part of this process requires extracting frequent letter chunks so as to isolate the relevant complex (i.e., multi-letter) graphemes (such as for instance graphemes SH and OO in SHOOK) to be associated with phonemes. This chunking route would initially be the salient part of the orthographic code, dominant at early stages of reading. The neural correlates of the chunking route could be the mid-anterior fusiform gyrus, given the evidence coming from case studies where lesions in these areas did not affect reading accuracy but generated phonological errors in spelling (Tsapkini & Rapp, 2010). The exact mechanism by which features in the chunking route would be selected and associated to phonemes is yet unknown, although past and present computational models of reading aloud point to a supervised process (Perry et al., 2007; Sibley et al., 2008), and the nature of the association
Figure 4. Grainger and Ziegler’s dual route model of orthographic coding. Starting from an alphabetic array of retinotopic letter detectors activated in parallel, reading acquisition is mediated by the construction of a diagnostic (left) and a chunking (right) route.

with phonology suggests that precise information about letter position would be required, possibly from both ends of the word as proposed by recent spelling studies (Fischer-Baum et al., 2010). It is worth noting here that, as we have seen, kernels with a wildcard character induce (but are not limited to) a natural positional count: it is easy to retrieve the position of any letter in the string by reading out these vector components that are involved in representing the combination of the target letter with either the initial or the final letter.

Furthermore, the chunking route in this account is thought to be responsible for the development of morpho-orthographic representations, known to play a key role in visual word recognition (Rastle & Davis, 2008). In languages such as English and French, affixes would be an obvious target for the chunking mechanism, given their relatively high frequency of occurrence. However, such chunking could also apply to root morphemes in the case of Semitic languages, and as such would involve contiguous and non-contiguous letter combinations. Indeed, Semitic languages are a perfect example of how natural languages can exploit non-contiguous letter combinations, since the tri-consonantal roots of these languages are formed of non-adjacent letters in the word, and empirical research on reading in Semitic languages underlines the importance of these roots in the word recognition process (Frost, 2006; Velan & Frost, 2011).

Meanwhile, according to the dual-route proposal another type of orthographic coding would slowly be taking shape in the VWFA. There, the increasing receptive fields of hierarchically organised neurons would allow for high-level cells to be tuned to possibly remote combinations of two or more letters (Vinckier et al., 2007), as described for instance
in the 3-Wildcard kernel we have implemented. However the feature space in this kernel would not be static but would evolve during reading acquisition so as to favour those letter combinations that are informative given the lexicon. Learning in the diagnostic route would be slow but automatic, effortless and unsupervised, involving for instance Hebb’s rule or a temporal variant thereof (Wallis & Rolls, 1997). Once set in place, the code would be dominant in silent reading and would allow for direct and fast access to semantics. It is thus proposed that masked priming experiments in the orthographic processing literature have largely been probing this diagnostic orthographic code.

Finally the dual-route model also addresses the hard problem of letter position coding, which is how humans manage to switch from a retinotopic representation to a word centered representation. This can be recast as the problem of learning location invariant word representations despite the fact that the same words will produce very different imprints on the retina at even the slightest variation in fixation point. This concern is not specific to words, and is shared throughout the visual object recognition literature (DiCarlo & Cox, 2007; Fazl et al., 2009), but the problem of invariance is perhaps most clearly perceived when considered in the word domain where the requirement for order is so explicit —e.g. anagrams like “triangle” and “relating” have different meanings. Order is trivially present in the retinal activation pattern or its cortically magnified transform in primary visual areas, but these do not directly allow for recognition: different words seen at the same location would have more overlapping patterns than the same word at different locations. Thus, if one is to recognise say “triangle” at all possible locations, it would appear that somehow location invariant letter detectors must be involved. However fully invariant detectors would not allow one to distinguish between “triangle” and “relating”. So how exactly does the system achieve location invariance without “losing” information about letter order? The answer, according to the dual route model, is that in both routes, location invariance is achieved not by learning location invariant letters, but rather by learning ordered and location invariant letter combinations —or in the language of String kernels, learning a feature map indexed by substrings of a given length. Using ordered letter combinations allows discrimination between anagrams since they will not have the same coordinates in this feature space.

Learning String kernels: the role of invariance

What could be the computational principles at work in the learning of these two orthographic codes? This is still an open question, but one possibility is that both routes of the dual-route model could evolve from the same feature space. Let us imagine a feature space indexed by all possible letter combinations up to a certain k, including a wildcard character. Learning the two orthographic codes could boil down to learning how to discard irrelevant features according to different criteria, or in other words, to learn for instance a 3-Wildcard kernel for the diagnostic route, as was successfully tested in this article, and a String kernel with both contiguous letter combinations and wildcard combinations for the chunking route. The features of the orthographic code in each route would ultimately de-

---

11 As measured for instance by the Frobenius norm or any standard matrix norm.
depend on how frequency of occurrence is used to select relevant entities from the same original feature space, that is, to maximize diagnosticity on the one hand and chunking on the other.

This idea resonates with the way training is currently achieved in HMAX, one successful and in some ways biologically inspired model of visual object recognition and classification (Serre et al., 2005). This network of 26 million units interleaved in simple and complex layers first learns a universal dictionary of features by “passive exposure” to a training base of environmental images. At this stage, simple units are tuned to respond maximally to patterns of activation in the previous complex layer, whereas complex units are hand-wired to respond to the maximally active simple unit in the previous simple layer, from among those tuned to the same feature combination at different locations and sizes. This first training stage results in a large number of simple units tuned to feature combinations in the environment, and invariant to location and scale — a universal dictionary. In a second “expert” learning stage, the network is exposed to a sequence of stimuli that all contain an object from the target domain, and a unit’s activity is now allowed to depend on its previous activity in the sequence. An abstract algorithm then implements a selection process within a randomly chosen pool of units, progressively replacing those units whose activity has fallen below some criterion with new random ones, which eventually produces a pool of units which are best fit to represent objects in the target domain.

The performance of HMAX on classification and identification tasks has been found to be comparable to the best algorithms in machine vision (Serre et al., 2005). What is more, formal results pertaining to invariance and discrimination properties in this type of architecture have been obtained for one-dimensional strings (Bouvrie et al., 2010). In these analyses, one critical mathematical object turned out to be the notion of kernels associated to network layers. Interestingly, these analyses reveal that such networks can discriminate between different strings but that unlike humans, they fail to distinguish a string from its reverse (abcd - dcba) and its checkerboard transformations (ababab-bababa). More generally, at a cognitive and biological level HMAX remains unsatisfactory for a number of reasons: it does not have feedback connections, its invariance properties are still largely handwired, and its expert selection process lacks a neurally plausible substrate. Invariance is more plausibly obtained in other hierarchical models of invariant object recognition like Visnet (Wallis & Rolls, 1997), which has a homogeneous structure and simple learning rules, and pARTSCAN (Fazl et al., 2009) which includes adaptive feedback connections. Both of these models suggest that a neurally plausible candidate mechanism to implement the selection process used in HMAX might be to learn lateral inhibitory connections within layers of the hierarchy, progressively shaping competitive networks that would select the most active units. A model that combines qualities from these three networks (adaptive feedback connections, competitive subnetworks, and simple learning rules acting at multiple stages during training) is currently under investigation (Hannagan & Grainger, 2011).

Finally the question of representational format arises, since at first sight the kind of

---

12We thank one anonymous reviewer for mentioning this study.
localist vectors used in String kernels suggests also a localist coding in the brain, whereby one neuron would be dedicated to each letter combination. There is little hope that, as described, this kind of coding would be of any use in a brain that faces significant amounts of noise, but Bowers (2010) has argued that redundant localist coding would circumvent the noise issue, by having many “copies” of neurons dedicated to represent one unique entity. However, even such a redundant localist view is not consistent with estimates derived from Bayesian reasoning on single cell responses, which point to sparse distributed coding where many neurons code for any letter combination, and each neuron is also active for many letter combinations (Plaut & McClelland, 2010). How then can we accommodate the notion of kernels within a distributed framework? The answer might already be contained in computational models of invariant visual object recognition, since models like HMAX and Visnet produce sparsely distributed units. The question of how representational format impacts on orthographic coding schemes has also been studied in Hannagan et al. (2011a), where a distributed implementation of Open-bigram coding derived from Holographic representations was proposed (Plate, 1995), and shown to outperform the localist version on a benchmark of behavioural effects. The implementation starts by assigning distributed vectors to letters and locations, and from then uses binding and chunking operators to build vectors for arbitrary complex structures. Vectors can be as sparse as required, and correlated letter vectors can be used to involve letter similarity in the computational design and testing of orthographic coding schemes.

Conclusion

In this article we have described the current situation with respect to theoretical proposals for orthographic coding, where several quite different alternatives coexist. We pointed out that unlike other coding schemes, Open-bigrams come very close to being instances of String kernels, a machine learning tool that achieves state-of-the-art performance in the fields of computational biology and natural language processing. This connection stems from the idea to operate comparisons in high-dimensional feature spaces indexed by letter combinations, which not only improves the separability of representations but also provides a common way to address issues encountered across domains—for instance, capturing the various mutations that can relate two proteins, and achieving location invariance in the visual word recognition system.

We have used this new connection to test String kernels on two recent behavioural benchmarks for orthographic coding. Although the modifications that turned Open-bigrams into String kernels were minor, we were able to demonstrate a significant improvement in performance. Furthermore a new coding scheme inspired from computational biology achieved superior scores on our benchmarks. Departing from the traditional use of kernels in Machine Learning we have argued for an explicit computation of feature coordinates in the brain, as evidenced by studies of the VWFA and more generally by brain imaging studies of visual object recognition. On this basis we have concluded that the adaptations required of any child who is learning how to read, that is the adaptations needed to go from generic object recognition to visual word recognition, are likely to involve String kernels rather than any other scheme to encode letter order. Although we leave to future research the proper demonstration of how such a coding scheme could be learned, this
convergence of computational, biological and behavioural arguments concludes our case for String kernels in the brain.

References


Appendix

3-Wildcard coding scheme

Here we provide a definition of the 3-wildcard kernel studied in the simulation section, which was inspired from the wildcard kernel proposed by Leslie & Kuang (2004). Let \( \Sigma \) be the english alphabet augmented by a wildcard character * that matches every letter, and \( \alpha \) a substring of length 3 in string S. We define \( \phi \) as:

\[
\forall \alpha \in S, \forall i \in \Sigma^3, \quad \phi_i(\alpha) = \begin{cases} 1, & \text{if } i \text{ appears in } \alpha \text{ with exactly one wildcard character } * \\ 0, & \text{otherwise.} \end{cases}
\]

The feature map for any string S is the sum of the feature maps for all possible \( \alpha \) in S:

\[
\phi(S) = \sum_{\alpha \in S, |\alpha|=3} \phi(\alpha)
\]

As usual the kernel between strings S and T is defined as the normalised inner product of the feature maps for S and T, that is:

\[
K_{3\text{wildcard}}(S,T) = \frac{\phi(S)^t \cdot \phi(T)}{||\phi(S)|| \cdot ||\phi(T)||}
\]

Tables A1 and A2 below provide detailed similarity measures on the benchmarks of Davis (2010) and Hannagan et al. (2011a), for the wildcard kernel and the other coding schemes studied in our simulation section.
<table>
<thead>
<tr>
<th>prime</th>
<th>target</th>
<th>Humans</th>
<th>Spatial Coding</th>
<th>GvH UOB</th>
<th>Kernel UOB</th>
<th>3-WildCard</th>
</tr>
</thead>
<tbody>
<tr>
<td>xycde</td>
<td>abcde</td>
<td>1</td>
<td>0.57</td>
<td>0.30</td>
<td>0.30</td>
<td>0.26</td>
</tr>
<tr>
<td>axydde</td>
<td>abcde</td>
<td>-3</td>
<td>0.71</td>
<td>0.30</td>
<td>0.30</td>
<td>0.32</td>
</tr>
<tr>
<td>abcxxy</td>
<td>abcde</td>
<td>3</td>
<td>0.57</td>
<td>0.30</td>
<td>0.30</td>
<td>0.28</td>
</tr>
<tr>
<td>axcdedefg</td>
<td>abcdefg</td>
<td>22</td>
<td>0.89</td>
<td>0.71</td>
<td>0.71</td>
<td>0.71</td>
</tr>
<tr>
<td>azyddefg</td>
<td>abcdefg</td>
<td>17</td>
<td>0.78</td>
<td>0.48</td>
<td>0.48</td>
<td>0.52</td>
</tr>
<tr>
<td>azyzedefg</td>
<td>abcdefg</td>
<td>1</td>
<td>0.67</td>
<td>0.29</td>
<td>0.29</td>
<td>0.36</td>
</tr>
<tr>
<td>awxyzfg</td>
<td>abcdefg</td>
<td>4</td>
<td>0.56</td>
<td>0.14</td>
<td>0.14</td>
<td>0.21</td>
</tr>
<tr>
<td>avwxyzg</td>
<td>abcdefg</td>
<td>-7</td>
<td>0.44</td>
<td>0.05</td>
<td>0.05</td>
<td>0.08</td>
</tr>
<tr>
<td>abcdde</td>
<td>abcde</td>
<td>47</td>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>abdxde</td>
<td>abcde</td>
<td>30</td>
<td>0.97</td>
<td>0.90</td>
<td>0.90</td>
<td>0.74</td>
</tr>
<tr>
<td>abcde</td>
<td>abcde</td>
<td>13</td>
<td>0.82</td>
<td>0.90</td>
<td>0.90</td>
<td>0.69</td>
</tr>
<tr>
<td>axcdde</td>
<td>abcde</td>
<td>30</td>
<td>0.86</td>
<td>0.60</td>
<td>0.60</td>
<td>0.58</td>
</tr>
<tr>
<td>abcde</td>
<td>abcde</td>
<td>31</td>
<td>0.86</td>
<td>0.60</td>
<td>0.60</td>
<td>0.59</td>
</tr>
<tr>
<td>acxde</td>
<td>abcde</td>
<td>19</td>
<td>0.84</td>
<td>0.60</td>
<td>0.60</td>
<td>0.53</td>
</tr>
<tr>
<td>abxcde</td>
<td>abcde</td>
<td>18</td>
<td>0.84</td>
<td>0.60</td>
<td>0.60</td>
<td>0.52</td>
</tr>
<tr>
<td>abxdef</td>
<td>abcdedefg</td>
<td>38</td>
<td>0.88</td>
<td>0.67</td>
<td>0.67</td>
<td>0.72</td>
</tr>
<tr>
<td>abxdef</td>
<td>abcdedefg</td>
<td>24</td>
<td>0.9</td>
<td>0.80</td>
<td>0.80</td>
<td>0.68</td>
</tr>
<tr>
<td>abxdef</td>
<td>abcdedefg</td>
<td>7</td>
<td>0.75</td>
<td>0.40</td>
<td>0.48</td>
<td>0.51</td>
</tr>
<tr>
<td>acxbedgfh</td>
<td>abcdedgfh</td>
<td>26</td>
<td>0.93</td>
<td>0.89</td>
<td>0.89</td>
<td>0.82</td>
</tr>
<tr>
<td>badfegh</td>
<td>abcdedgfh</td>
<td>-1</td>
<td>0.71</td>
<td>0.86</td>
<td>0.86</td>
<td>0.73</td>
</tr>
<tr>
<td>badfegh</td>
<td>abcdedgfh</td>
<td>9</td>
<td>0.71</td>
<td>0.86</td>
<td>0.86</td>
<td>0.73</td>
</tr>
<tr>
<td>dcbahfge</td>
<td>abcdedgfh</td>
<td>-1</td>
<td>0.48</td>
<td>0.57</td>
<td>0.57</td>
<td>0.23</td>
</tr>
<tr>
<td>abcdedfg</td>
<td>abcdefg</td>
<td>55</td>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>abcxdedfg</td>
<td>abcdefg</td>
<td>44</td>
<td>0.98</td>
<td>1.00</td>
<td>0.87</td>
<td>0.88</td>
</tr>
<tr>
<td>abxcydefg</td>
<td>abcdefg</td>
<td>33</td>
<td>0.91</td>
<td>1.00</td>
<td>0.76</td>
<td>0.79</td>
</tr>
<tr>
<td>abcxyzedfg</td>
<td>abcdefg</td>
<td>22</td>
<td>0.82</td>
<td>1.00</td>
<td>0.68</td>
<td>0.71</td>
</tr>
<tr>
<td>xabcdefg</td>
<td>abcdefg</td>
<td>50</td>
<td>0.89</td>
<td>1.00</td>
<td>0.87</td>
<td>0.81</td>
</tr>
<tr>
<td>abcdeefg</td>
<td>abcdefg</td>
<td>41</td>
<td>1</td>
<td>1.00</td>
<td>0.93</td>
<td>0.94</td>
</tr>
<tr>
<td>abcdedfg</td>
<td>abcdefg</td>
<td>39</td>
<td>1</td>
<td>1.00</td>
<td>0.91</td>
<td>0.90</td>
</tr>
<tr>
<td>abcdedfg</td>
<td>abcdefg</td>
<td>39</td>
<td>0.98</td>
<td>1.00</td>
<td>0.87</td>
<td>0.87</td>
</tr>
<tr>
<td>abcdedefg</td>
<td>abcdefg</td>
<td>42</td>
<td>0.87</td>
<td>0.88</td>
<td>0.91</td>
<td>0.91</td>
</tr>
<tr>
<td>abcdeefg</td>
<td>abcdefg</td>
<td>35</td>
<td>0.87</td>
<td>0.71</td>
<td>0.85</td>
<td>0.86</td>
</tr>
<tr>
<td>abcdgf</td>
<td>abcdefg</td>
<td>37</td>
<td>0.87</td>
<td>0.71</td>
<td>0.85</td>
<td>0.86</td>
</tr>
<tr>
<td>abcdg</td>
<td>abcdefg</td>
<td>36</td>
<td>0.87</td>
<td>0.71</td>
<td>0.85</td>
<td>0.86</td>
</tr>
<tr>
<td>acdf</td>
<td>abcdef</td>
<td>27</td>
<td>0.72</td>
<td>0.40</td>
<td>0.63</td>
<td>0.62</td>
</tr>
<tr>
<td>axcdef</td>
<td>abcdef</td>
<td>12</td>
<td>0.75</td>
<td>0.40</td>
<td>0.40</td>
<td>0.40</td>
</tr>
<tr>
<td>acdf</td>
<td>abcdef</td>
<td>20</td>
<td>0.72</td>
<td>0.40</td>
<td>0.63</td>
<td>0.62</td>
</tr>
<tr>
<td>acdf</td>
<td>abcdef</td>
<td>5</td>
<td>0.69</td>
<td>0.33</td>
<td>0.53</td>
<td>0.46</td>
</tr>
<tr>
<td>fcda</td>
<td>abcdef</td>
<td>7</td>
<td>0.28</td>
<td>0.07</td>
<td>0.11</td>
<td>0.08</td>
</tr>
<tr>
<td>abcde</td>
<td>abcdefg</td>
<td>45</td>
<td>0.67</td>
<td>0.48</td>
<td>0.69</td>
<td>0.54</td>
</tr>
<tr>
<td>cdefg</td>
<td>abcdefg</td>
<td>37</td>
<td>0.67</td>
<td>0.48</td>
<td>0.69</td>
<td>0.52</td>
</tr>
<tr>
<td>acdeg</td>
<td>abcdefg</td>
<td>29</td>
<td>0.75</td>
<td>0.48</td>
<td>0.69</td>
<td>0.59</td>
</tr>
<tr>
<td>abcd</td>
<td>abcdefg</td>
<td>23</td>
<td>0.56</td>
<td>0.29</td>
<td>0.53</td>
<td>0.40</td>
</tr>
<tr>
<td>defg</td>
<td>abcdefg</td>
<td>12</td>
<td>0.56</td>
<td>0.29</td>
<td>0.53</td>
<td>0.40</td>
</tr>
<tr>
<td>aceg</td>
<td>abcdefg</td>
<td>8</td>
<td>0.6</td>
<td>0.29</td>
<td>0.53</td>
<td>0.53</td>
</tr>
</tbody>
</table>

| Corr. :   | 0.73 | 0.67 | 0.75 | 0.80 |

Table A1: Human priming (ms) and Similarities for the four coding schemes on selected conditions from the Davis (2010) benchmark studied in our simulation section. Adapted, with permission, from Davis (2010).
Table A2: Similarities for the four coding schemes on the Hannagan et al. (2011a) benchmark studied in our simulation section. Adapted, with permission, from Hannagan et al. (2011a).

<table>
<thead>
<tr>
<th>prime</th>
<th>target</th>
<th>Spatial Coding</th>
<th>GvH UOB</th>
<th>Kernel UOB</th>
<th>3-WildCard</th>
</tr>
</thead>
<tbody>
<tr>
<td>abcde</td>
<td>abcde</td>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>abde</td>
<td>abcde</td>
<td>0.79</td>
<td>0.60</td>
<td>0.77</td>
<td>0.70</td>
</tr>
<tr>
<td>abccde</td>
<td>abcde</td>
<td>1</td>
<td>1.00</td>
<td>0.92</td>
<td>0.82</td>
</tr>
<tr>
<td>abcxde</td>
<td>abcde</td>
<td>0.92</td>
<td>1.00</td>
<td>0.82</td>
<td>0.74</td>
</tr>
<tr>
<td>abxye</td>
<td>abcde</td>
<td>0.71</td>
<td>0.30</td>
<td>0.30</td>
<td>0.31</td>
</tr>
<tr>
<td>axcde</td>
<td>abcde</td>
<td>0.86</td>
<td>0.60</td>
<td>0.60</td>
<td>0.58</td>
</tr>
<tr>
<td>abxdef</td>
<td>abcdef</td>
<td>0.88</td>
<td>0.67</td>
<td>0.67</td>
<td>0.72</td>
</tr>
<tr>
<td>abxdyf</td>
<td>abcdef</td>
<td>0.75</td>
<td>0.40</td>
<td>0.40</td>
<td>0.44</td>
</tr>
<tr>
<td>xbcde</td>
<td>abcde</td>
<td>0.71</td>
<td>0.60</td>
<td>0.60</td>
<td>0.51</td>
</tr>
<tr>
<td>abxde</td>
<td>abcde</td>
<td>0.86</td>
<td>0.60</td>
<td>0.60</td>
<td>0.60</td>
</tr>
<tr>
<td>abcdx</td>
<td>abcde</td>
<td>0.71</td>
<td>0.60</td>
<td>0.60</td>
<td>0.51</td>
</tr>
<tr>
<td>abdce</td>
<td>abcde</td>
<td>0.89</td>
<td>0.90</td>
<td>0.90</td>
<td>0.74</td>
</tr>
<tr>
<td>bacfefh</td>
<td>abcdefgh</td>
<td>0.48</td>
<td>0.86</td>
<td>0.86</td>
<td>0.73</td>
</tr>
<tr>
<td>abedcf</td>
<td>abcdef</td>
<td>0.78</td>
<td>0.80</td>
<td>0.80</td>
<td>0.68</td>
</tr>
<tr>
<td>acxde</td>
<td>abcde</td>
<td>0.81</td>
<td>0.60</td>
<td>0.60</td>
<td>0.53</td>
</tr>
<tr>
<td>abcdfe</td>
<td>abcdefg</td>
<td>0.67</td>
<td>0.48</td>
<td>0.69</td>
<td>0.54</td>
</tr>
<tr>
<td>cdefg</td>
<td>abcdefg</td>
<td>0.67</td>
<td>0.48</td>
<td>0.69</td>
<td>0.52</td>
</tr>
<tr>
<td>acdeg</td>
<td>abcdefg</td>
<td>0.69</td>
<td>0.48</td>
<td>0.69</td>
<td>0.59</td>
</tr>
<tr>
<td>abccdf</td>
<td>abcbbdef</td>
<td>0.82</td>
<td>0.71</td>
<td>0.85</td>
<td>0.86</td>
</tr>
<tr>
<td>abccdef</td>
<td>abcbbdef</td>
<td>0.81</td>
<td>0.88</td>
<td>0.91</td>
<td>0.91</td>
</tr>
</tbody>
</table>