String Kernels as Composition

Daoud Clarke

November 6, 2012

We propose that string kernels can be viewed as a general framework for representing vector-based composition.

1 Motivation

The motivation for using string kernels is:

- There is an argument that natural language features such as conjunction, disjunction and negation cannot easily be represented within a framework in which composition is linear. Kernels provide a way to explore a large range of non-linear composition methods, and indeed the original motivation for their use is to make it easy to work with non-linearity.
- There is an argument that the vector space for sentences should be infinite dimensional, since information should not be lost as we compose sentences. Kernels provide a way to implicitly work with very high and infinite dimensional spaces while allowing for efficient computation.
- When doing tasks such as classification, we have a need to be able to evaluate the document as a whole. One way of doing this is to use kernels to compare a two bags or sequences of vectors (associated with individual words, phrases or sentences). Together with tools such as support vector machines, we can then efficiently learn a classifier for documents.
- We can design kernels with desirable properties for vector-based composition, then examine their properties as vector spaces to give us insight into what the nature of vector-based composition should be. For example, if we come up with a good kernel for composing sentences, we can then examine what properties the composition operation has in terms of the implicitly defined vector space.

2 Background

The theory that allows us to consider kernels as vector based composition is the following:

Definition 1 (Shawe-Taylor and Christianini) *A function*

$$\kappa : X \times X \longrightarrow \mathbb{R}$$

satisfies the finitely positive semi-definite property if it is a symmetric function for which the matrices formed by restriction to any finite subset of the space X are positive semi-definite, i.e. their eigenvalues are all non-negative.

Theorem 1 (Shawe-Taylor and Christianini) *A function*

$$\kappa : X \times X \longrightarrow \mathbb{R}$$

which is either continuous or has a countable domain, can be decomposed

$$\kappa(x, y) = \langle \phi(x), \langle \phi(y) \rangle$$

into a feature map ϕ into a Hilbert space F applied to both its arguments followed by the evaluation of the inner product in F if and only if it satisfies the finitely positive semi-definite property.

3 Kernels as Composition

The general form of a model for vector-based composition is

$$A^* \longrightarrow V \longrightarrow \langle \cdot, \cdot \rangle$$

i.e. we map from strings to some vector space V, and from that vector space we can then compute an inner product, for example to get cosine similarities.

The kernel approach reverses the last part of this process:

$$A^* \longrightarrow \langle \cdot, \cdot \rangle \longrightarrow V$$

We map directly from strings to an inner product, and the vector space V is determined implicitly by this inner product. This means that we often do not even need to explicitly represent V, which allows us to deal with all sorts of high-dimensional and infinite-dimensional vector spaces.

4 Examples

Existing Vector Composition

Any existing method of composing vectors can be used to define a kernel on strings, by simply composing the vectors and taking the resulting inner product. For some of these, it may be more computationally efficient to use a kernel directly.

Tensor Product

Assume we have a mapping ψ from A to V where V is some vector space representing symbol meanings. The tensor product approach maps a string $x \in A^*$ to a vector

$$\psi(x_1) \otimes \psi(x_2) \otimes \cdots \otimes \psi(x_n)$$

where x_i are the individual symbols in the string x.

If we define the inner product between strings of different lengths to be zero, then it is easy to see that the kernel function

$$\kappa(x, y) = \begin{cases} \prod_i \langle \psi(x_i), \psi(y_i) \rangle & \text{if } |x| = |y| \\ 0 & \text{otherwise} \end{cases}$$

allows us to compute the inner product between the tensor product vectors, without explicitly computing the vectors.

It also suggests ways in which the requirement that strings have to be the same length to have a non-zero inner product can be relaxed - for example we could look at the value of the inner product for all substrings/subsequences of the longer string with the same length as the shorter string, and take the maximum over these (although we'd need to check that this defines a valid kernel function).

Subsequence Kernels

These map a string to a bag of all subsequences of the string, or a bag of subsequences of strings under a fixed length. It is also possible to weight them according to the number of "gaps" used when forming the subsequence. These have been commonly used in machine learning and are implemented, for example, in the Weka toolkit.

Subsequence Tensor Product

We can combine the above two ideas, and map a string of words to a sum of tensor products of the vector representations for all subsequences of the string. This gives another way to make use of the inner product while still allowing the comparison of strings of different length.

Bag of Vectors

In combining a bag of vectors, we again have the situation where we do not want to lose information, while allowing comparison between bags of different sizes. One way of doing this would be to use something similar to the subsequence tensor product, but allow the vectors to be combined in any order.

5 Questions

If we want to use the context-theoretic framework idea of a vector lattice to compute entailments, then we need to do some extra work to make use of kernels. Although they give us a vector space, they don't give us a particular orthonormal basis on the vector space, so we can't define vector lattice meets and joins. In order to do so, we'd need to either define the vector lattice ordering directly on the kernel space, or define a positive cone. There is then the issue of how to compute values from the vectors if they are defined implicitly...

6 Lattice Orderings and Kernels

If we are going to use kernels to do composition there is the question of how to define entailment between strings. Context-theoretic semantics suggests that the correct way to do this is with a lattice ordering on a vector space. Although most vector spaces come equipped with a natural ordering defined by an associated orthonormal basis, in the case of the vector space defined implicitly by the kernel, there is no such basis and thus no natural ordering. The alternative is to specify an ordering along with the kernel. This allows for a lot of flexibility in expressing how meanings are related.

One simple way of defining a valid vector space ordering is to choose a set of elements that generate a positive cone. For example, if we choose $u - v$ to be positive, then $0 \leq u - v$ or $v \leq u$. Counterintuitively, it is possible to relate two vectors to one another in this way, even if they are orthogonal. This may mean that the actual definition of composition may be less important than the definition of positive cone: we can use a definition of composition that makes many things orthogonal, such as the tensor product, and then relate them using a positive cone.

For example, if we wish to impose the conjunctive nature of "and" on the vector space, we can do this by requiring that $uav \leq u$ and $uav \leq v$ for all u and v where a is the vector for "and". We can do the same for "or" with $u \leq uov$ and $v \leq uov$ where o is the vector for "or".

Similarly, we can make all adjectives restrict the properties of the nouns they operate on by making $au \leq u$ for all adjective vectors a and all noun space vectors u.

In a sense, this is like imposing a logic on top of the vector space structure, where the rules of the logic are encapsulated in the positive cone.