String Kernels as Composition

Daoud Clarke

February 22, 2013

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

4.1 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.

4.2 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).

4.3 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.

4.4 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.

4.5 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.

There is some research on this problem, mainly in the area of image analysis. Most of this revolves around estimating the probability distribution of vectors within the space (which is very similar to some of Miro's suggestions?). Kondor and Jebara (2003) attempts to do this directly by fitting a Gaussian distribution and then using the Bhattacharyya measure as an inner product between these distributions:

$$K(p,p') = \int \sqrt{p(x)}\sqrt{p'(x)}dx$$

Desobry et al. (2005) attempts to avoid the problem with data sparseness in estimating the probability distribution by computing probability distribution "level sets", subsets of the vector space which correspond to areas where it is estimated that most of the data lies. They then define kernels in terms of how similar these sets are.

Grauman and Darrell (2005) attempt to solve the same problem by estimating the similarity between probability distributions using histograms over the vector space.

Variations on these techniques may be applicable to our situation, but it is possible that high dimensionality, combined with potentially small numbers of vectors (e.g. for sentences) may mean that it is not feasible to attempt to estimate probability distributions.

There are some other potential methods derived from kernels presented in Shawe-Taylor and Cristianini (2004). An obvious one is simply to combine the individual vectors to a single vector using one of the standard "composition" approaches:

- addition: this can be done even if the vectors are defined implicity using kernels
- component-wise multiplication
- component-wise maximum/minimum

The kernel is then defined as the inner product on the composed vectors (though, other than addition using underlying kernels, we are not getting any benefit from kernels in this approach).

The all-subsets kernel allows the efficient computation of a similarity based on all possible subsets of the set of features that make up the vector space. It is defined as

$$\kappa_{\subseteq}(\mathbf{x},\mathbf{z}) = \prod_{i=1}^{n}(1 + x_i z_i)$$

There is potential to adapt this to our situation by considering the "features" to actually be the set of vectors under consideration; this would give us a fast way to compare all possible subsets of the vectors with one another.

Another line of investigation is to use locality sensitive hashing to allow the comparison of sets of vectors. Let $h_n(\mathbf{v})$ be the first n binary digits of an N digit hash function applied to vector $\mathbf{v}$. For a set of vectors $\mathbf{v_i}$, we define a vector

$$\sum_{i,n} 2^{N-n} h_n(\mathbf{v_i})$$

in the space with a different binary string for each dimension.

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...

References

Frédéric Desobry, Manuel Davy, and William J Fitzgerald. A class of kernels for sets of vectors. In *Proceedings of the 13th European Symposium on Artificial Neural Networks*. Citeseer, 2005.

Kristen Grauman and Trevor Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1458–1465. IEEE, 2005.

Risi Kondor and Tony Jebara. A kernel between sets of vectors. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, volume 20, page 361, 2003.

John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.