String Kernels

Blaž Fortuna
Department of Knowledge Technologies
Jozef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
e-mail: blaz.fortuna@ijs.si

Abstract

This paper provides an overview of string kernels. String kernels compare text documents by the substrings they contain. Because of high computational complexity, methods for approximating string kernels are shown. Several extensions for string kernels are also presented. Finally string kernels are compared to BOW.

1 What Is A Kernel

Standard learning systems (like neural networks or decision trees) operate on input data after they have been transformed into feature vectors living in an m dimensional space. In some cases the data can not be easily described by explicit feature vectors, for example graphs or text. In many cases extracting explicit feature vectors can be almost as hard as solving the whole problem.

Kernel Methods (KM), [4], are an alternative to explicit feature extraction. The main part of KM is a function known as kernel function which returns the inner product between documents mapped into a high dimensional feature space. In many cases the inner product can be calculated without explicitly computing feature vectors.

Several machine learning algorithms can be rewritten in such way that feature vectors only appear inside inner product so there is no need to calculate them explicitly. The best known example is Support Vector Machine (SVM), but there are also others: Kernel PCA, Kernel KCCA, Nearest Neighbour, etc.

2 String Kernel

The most common technique for representing text documents is Bag of Words (BOW) together with TFIDF weighting [1]. In the BOW representation there is a dimension for each word; a document is than encoded as a feature vector with word frequencies as elements. When using this approach, feature vectors are explicitly calculated. Kernel function between documents is the inner product between these feature vectors.

The main idea of string kernels [2] is to compare documents not by words, but by the substrings they contain. These substrings do not need to be contiguous, but they receive different weighting according to degree of contiguity. For example: substring 'c-a-r' is present both in word 'card' and 'custard' but with different weighting. Weight depends on the length of substring and decay factor λ (both defined later). In previous example substring 'car' would receive weight λ^4 as part of 'card' and λ^7 as part of 'custard'.

The advantage of this approach, comparing to BOW, is that it can detect words with different suffixes or prefixes: the words 'microcomputer', 'computers' and 'computerbased' all share common substrings.

Definition (String Subsequence Kernel – SSK) *Let Σ be a finite alphabet. A string is a finite sequence of characters from Σ, including empty sequence. For strings s and t we denote with $|s|$ the length of string $s = s_1 \dots s_{|s|}$, with st the string obtained by concatenating the strings s and t and with $s[i:j]$ substring $s_i \dots s_j$. We say that u is a substring of s if there exists indices $\mathbf{i} = (i_1, \dots, i_{|u|})$ with $1 \le i_1 < \dots < i_{|u|} \le |s|$ such that $u = s[\mathbf{i}]$.*

The length $l(\mathbf{i})$ of the subsequence in s is $i_{|u|} - i_1 + 1$. Feature mapping Φ for string s is given by defining Φ_u for each $u \in \Sigma^n$ as

$$\Phi_u(s) = \sum_{\mathbf{i}:u=s[\mathbf{i}]} \lambda^{l(\mathbf{i})}$$

for some $\lambda \le 1$. These features measure the number of occurrences of subsequences in the string s weighting them according to their lengths. Hence, the inner product of the feature vectors for two strings s and t give a sum over all common subsequences weighted according to their frequency of occurrence and lengths

$$K_n(s,t) = \sum_{u \in \Sigma^n} \Phi_u(s)\Phi_u(t) =$$

$$\sum_{u \in \Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{i})+l(\mathbf{j})}.$$

Computation of features for $n > 4$ is very expensive, hence the explicit use of such would be impossible, but it turns out that the kernel function for these feature vectors can be calculated very efficiently. In order to derive an efficient procedure for computing such kernel we first introduce following function. Let

$$K_i^{'}(s,t) = \sum_{u \in \Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{|s|+|t|-i_1-j_1+2},$$

that is counting the length from the beginning of the particular sequence through the end of strings s and t instead of just $l(\mathbf{i})$ and $l(\mathbf{j})$. We can now define a recursive computation for $K_i^{'}$ and hence compute K_n:

$$\begin{aligned}
K_0^{'}(s,t) &= 1, \text{ for all } s,t, \\
K_i^{'}(s,t) &= 0, \text{ if } \min(|s|,|t|) < i, \\
K_i(s,t) &= 0, \text{ if } \min(|s|,|t|) < i, \\
K_i^{'}(sx,t) &= \lambda K_i^{'}(s,t) + \\
&\quad \textstyle\sum_{j:t_j=x} K_{i-1}^{'}(s,t[1:j-1])\lambda^{|t|-j+2}, \\
&\qquad\qquad i = 1,\ldots,n-1, \\
K_n(sx,t) &= K_n(s,t) + \\
&\quad \textstyle\sum_{j:t_j=x} K_{n-1}^{'}(s,t[1:j-1])\lambda^2.
\end{aligned}$$

To remove any bias introduced by different lengths of document we normalize the feature vectors by creating new embedding $\tilde{\Phi}(s) = \frac{\Phi(s)}{\|\Phi(s)\|}$, which gives rise to the kernel

$$\tilde{K}(s,t) = \frac{K(s,t)}{\sqrt{Ks,sKt,t}}.$$

3 Implementation

The computational complexity for the SSK can be reduced to $O(n|s||t|)$ using the recursive definition from the previous section; space complexity is of order $O(|s||t|)$. While this is much faster than explicit computation of feature vectors, it is still too slow for learning SVM classification on datasets with a thousands of long documents. Approximations of the SSK can be used to tackle down this issue.

3.1 TRIE

So far we were searching for all the substrings in the document giving higher weight to more contiguous ones. Because weight of a substring is dropping exponentially with its length (weight equals $\lambda^{l(s)}$) and hence longer substrings add less to the features, we can limit search only to substrings of length less than m.

TRIE [4] comes from words *retrieval tree.* It defines a tree where all edges have labels from alphabet Σ. One can than view paths between root and leaves as strings composed of labels on edges. The leaves of a tree of depth n, built from substrings of length less than m from document s, are exactly the components of feature vector $\Phi(s)$. Simultaneous building of tree for s and t can be done in $O((|s|+|t|)m^{m-n})$ time because most components can be early detected as non perspective. This method is faster than the one following from recursive equations in all cases where $m - n$ is small.

3.2 Dimension Reduction

The approach here is to find a lower dimensional subspace of the feature space, that captures enough information about them, and to project feature vectors into this subspace [2]. The projected vectors obtained this way can than be explicitly calculated and used as approximation for feature vectors. Since the approximated vectors are explicitly calculated, the linear versions of learning algorithms can be used and this usually adds another speed-up.

The first step of the approach is to choose the orthogonal basis which will define the subspace. Let k be the dimension of the subspace. Than the basis $S = \{s_i | i = 1,\ldots,k\}$ consists of the k most frequent continuous substrings of length n from dataset. The elements of S are orthogonal by the definition of kernel K_n. Feature vectors projected into subspace spanned by basis S are $P(s) = (K_n(s, s_1), \ldots, K_n(s, s_k))$. Evaluation of $K_n(s, s_i)$ is fast since length of s_i is only n.

3.3 Incomplete Cholesky Decomposition

On a fixed set of documents, kernel function can be also given as a matrix with elements being $G_{ij} = K_n(s_i, s_j)$. This matrix is called Gram matrix, note that G is positive-definite matrix. In order to calculate Gram matrix $\frac{1}{2}n(n+1)$ evaluations of kernel function are necessary. To speed up the computation of G we could use only an approximation for it, for which less evaluations would be necessary.

Cholesky decomposition of positive-definite matrix is $G = V^T V$ where V is upper triangular. Let X be a matrix with feature vectors as columns. Gram matrix can than be written as $G = X^T X$. QR decomposition of X can be obtained by doing Gram-Schmidt on the columns of X. Let $X = QR$ where Q is orthogonal ($Q^T Q = I$). Than

$$G = X^T X = (QR)^T (QR) = R^T (Q^T Q) R = R^T R.$$

From V and R being upper triangular and Cholesky decomposition being unique follows $V = R$.

By doing only k steps of Gram-Schmidt we obtain only first k rows of R, $\tilde{G} = R^T R$ can now be used as approximation of G. This approximation can be improved by carefully choosing the vectors for each step of Gram-Schmidt: at each step the vector with the highest residual norm is chosen. It is important to note that all this operation can be done without explicitly using feature vectors. The number of evaluations of kernel function depends on the number of calculated rows of R.

4 Extentions

Syllables or words instead of characters Instead of looking at documents as a succession of characters they can be viewed as a succession as syllables or words. One important advantage of using syllables or words is document length reduction. When using syllables the advantage of detecting similarities between words with different prefixes or suffixes still remains, while using words this property is lost. The difficult part here is in breaking words into syllables.

Convex Combination of Kernels By using the property that convex combination of kernel functions is again a kernel function we can get new kernel functions. In case of string kernels we can combine kernel functions K_i of different lengths i. While calculating K_n (using recursion I mentioned upper), values of K_i for $i = 1, \ldots, n-1$ are obtained as intermediate results. These results can be used to form convex combinations without extra computational cost.

Weights Instead of using same weight λ for all symbols, each symbol (character, syllable, word) x can be assigned different weight λ_x. In this way we can introduce prior knowledge into this model and put accent on certain symbols. This extension can be added with few simple changes of recursive computation of K_n and it does not increase complexity.

Soft-Matching Besides matching only equal symbols, we can also match similar symbols with adding extra factor to ensure that soft matches get lower weight. When words are used as symbols this can be used to incorporate synonyms into the model. For example: with help of WordNet synonym words can be matched. This extension can also be added to recursive computation of K_n without extra computational cost, although finding soft matches alone can be expensive.

5 Experiments

The following results are taken from [3]. String, Syllables and Words kernel were compared to BOW with TF and TFIDF vectors. The task was to correctly classify documents into categories. The experiments were performed on Reuters-21578 dataset and SVM was chosen as a classification method. Gram matrixes were calculated for a subset of 1000 documents with $n = 3$ for all kernels. This took 150 minutes for String, 30 minutes for Syllables and 15 minutes for Word kernel. Two tests were made on this subset. First 300 documents were used for training and the remaining 700 for testing. In second experiment 600 documents were used for training and the remaining 400 for testing.

	CE [%]	F1 [%]	NSV	Time [min]
String Kernel	15	87	184	208
Syllable Kernel	12	89	218	29
Word Kernel	18	85	157	9
BOW (TF only)	18	84	150	1/6
BOW (TFIDF)	8	93	252	1/6

Table 1: Results for text categorisation with svm (CE – Classification error, NSV - Number of Support Vectors)

	CE [%]	F1 [%]	NSV
String Kernel	3	97	305
Syllable Kernel	4	97	379
Word Kernel	3	97	281
BOW (TF only)	3	97	287
BOW (TFIDF)	4	97	443

Table 2: Results for text categorisation with svm (CE – Classification error, NSV - Number of Support Vectors)

The next experiment represents a comparison between two approximation techniques: Dimension Reduction (DR) and Incomplete Cholesky Decomposition (ICR). The speed and performance of trained classifier are compared. The training set consisted of 1000 documents and testing was done on 200.

	Prec [%]	Rec [%]	Time [sec]
TFIDF	95	97	24
DR (1500)	87	90	24
DR (2500)	87	91	48
DR (3500)	89	91	64
ICD (200)	86	92	49
ICD (450)	88	92	114
ICD (750)	90	94	244

Table 3: Comparison of approximation techniques.

References

[1] T. Joachims. Making large-scale svm learning practical. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning.* MIT-Press, 1999.

[2] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and Watkins C. Text classification using string kernels. *Journal of Machine Learning Research*, (2):419–444, 2002.

[3] C. Saunders, H. Tschach, J. and Shawe-Taylor, (2002) Syllables and Other String Kernel extensions. In *Proceedings of Nineteenth International Conference on Machine Learning* (ICML '02)

[4] J. Shawe-Taylor, N. Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, 2004