A look into the String Subsequence Kernels

Tim Lachmann Arvid Österlund Mathilda von Schantz Mukund Seethamraju

Project Report - DD2434 Advanced Machine Learning
February 3, 2018

Abstract

(Lodhi et al. 2002) proposed String Sub-sequence Kernel (SSK) as a new approach for text categorization. The kernel function calculates the similarity between two documents by finding the character subsequences (either contiguous or non-contiguous) of length k. More weightage is assigned to contiguous strings by penalizing non-contiguous strings by a exponentially decaying factor λ. They have also proposed a dynamic programming based implementation to make the algorithm tractable.

The authors in their paper, evaluated the SSK by classifying documents from Reuters-21578 text categorization dataset (Carnegie Group and Reuters 2004) using Support vector machine classifier. The string subsequence kernel was compared to word-gram kernel and character level N-gram kernel in the paper.

A reimplementation of the string subsequence kernel was made for classifying the documents using support vector machines. We used the Reuters-21578 text categorization collection data for conducting our experiments, which was also used in the original paper. Even though we tried optimizing our code by using (Behnel et al. 2011) and parallel processing, SSK approach would still cost a huge amount of time in building the gram matrix. We infer from our experiments, that the string subsequence kernel methods don't perform well on small datasets.

1 Introduction

Text categorization has many applications in different areas. Some include information retrieval, spam filtering, sentiment analysis, language identification and as well as protein- and gene-identification.

The Support Vector Machine (SVM) is a popular machine learning method in text categorization. SVM as a classifier tries to find a separating hyperplane between the classes that maximizes the margin between itself and its closest data points. When the data can not be separated using a linear classifier, data is projected into a high-dimensional space, where the classifier can find the separating hyperplane. The intractability of working in such high dimensions is solved with "the kernel trick". The kernel is a function, which when given two input data points, returns the inner product between the two data points mapped into the high dimensional space: $K(x, y) = \phi(x) \cdot \phi(y)$. As long as a learning problem can be formulated with higher-dimensional data points only appearing in these inner products, there is no need to explicitly calculate the features, which constitutes a great saving in terms of performance. Some of the reasons why SVMs are helpful in text categorization are given by (Joachims 1998). The arguably most important reason is that text categorization usually involves a large number of features. Therefore, an SVM is well suited for this since one of its statistical properties is that its performance does not depend on the dimensionality of the space where the separating hyperplane is created, due to the kernel trick. There are also few irrelevant features in text. This means that it is hard to reduce the number of features without the loss of performance, which brings us back to the aforementioned point.

1.1 The String Subsequence Kernel (SSK)

Lodhi, Saunders, Shawe-Taylor, Cristianini and Watkins introduces in (Lodhi et al. 2002) a new kernel to be used in text classification. The novel feature in their approach is that it skips the usual step of converting the data from text documents to feature vectors. Instead, they view documents as raw strings, and the similarity of two documents as the number of substrings, contiguous as well as non-contiguous, they have in common. Note that a non-contiguous substring is simply an ordered set of non-adjacent characters from a string. The non-contigous substrings are penalized by a parameter $\lambda \in [0, 1]$, so as to give higher weight to the contiguous substrings, seeing as how they contain more information. The kernel is then given as

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

where Σ^n is the set of all strings of length n and **i** and **j** constitues indices of (contiguous or non-contiguous) substrings in the strings s and t that corresponds to the letters in u. l(**i**) and l(**j**) are lengths of the substrings given as l(**j**)= $j_{|u|} - j_1 + 1$. Thus, l(**j**) is given a higher value for a non-contiguous substring, since the difference between the last index and the first is greater. This means that the non-contiguous substrings are penalized exponentially with the value of λ. Furthermore, the number of occurrences of a substring will weigh in - every time u appears in a string will be accounted for.

This formulation of the kernel is computationally intractable, since it involves all possible strings in Σ^n. Therefore, two supporting functions are formulated to form a recursive definition of the kernel. The recursion builds on the idea of calculating $K_n(s, t)$ for increasing values for the length of u, from 1 up to n, while increasing the number of letters considered in s and t. This idea is realized through two helper functions $K_i'(s, t)$ and $K_i''(s, t)$ where i = 1, ..., n-1.

One letter at a time is removed from one string, and that letter is then seen as the last letter in u. Therefore, u does not have to be taken from Σ^n, or Σ^i when $i < n$, but can rather be inferred from the strings in question. Then each recursion incurs a factor of λ to the final result. The recursive formula can be solved using dynamic programming, to a final algorithm of a complexity O($n|s||t|$), where n is the length of the subsequence considered, and $|s|$ and $|t|$ are the lengths of the two strings.

With a data set of m documents where the average length of a document is t characters, and the length of subsequences considered is n, the calculation of a gram matrix runs in O(m^2nt^2). This becomes intractable for long documents, or big data sets. Therefore, the authors suggests an approximation of the learning algorithm.

The idea behind the approximation is to find a compact subset of the data called $\tilde{S}$. If the construction of this set is done in a careful manner, an approximation of the kernel can be done where

$$K(x,z) \approx \sum_{s_i \in \tilde{S}} K(x,s_i)K(z,s_i) \tag{1}$$

The trade-off here is that the smaller $|\tilde{S}|$ is, the faster the calculation of the kernel will be, but the less closely it will resemble the true $K(x,z)$. The authors in their paper, chose the set $\tilde{S}$ by the k most common n-grams, or contiguous substrings of length n, present in the data set.

If we then have a training set of size m, where the average length of the documents are t, and a set $\tilde{S}$ with l entries, each of *n'* characters, the average computation of one kernel value runs in O($lntn'$). Calculating the gram matrix, on the other hand, runs in O($mnn'lt + lm^2$). This because first, each value for $k(x,s_i)$ needs to be computed for all documents x in the training set and for all $s_i \in \tilde{S}$. This computation runs in O($mnn'lt$) - each $k(x,s_i)$ evaluation will run in O($n|s||t|$) = O($nn't$), and there are ml such evaluations to perform. Then, using the stored values, each element in the gram matrix will run in O($|S|$) time (see equation 1) using the stored values of $K(x,s_i)$ and $K(z,s_i)$. Since the original gram matrix computation runs in O(m^2nt^2), the approximation constitutes a saving when $n'l < mt$ and $l < nt^2$.

1.2 Scope and objectives

A subset of the experiments in (Lodhi et al. 2002) will be attempted to be repeated, on a subset of the data. Lodhi et al runs three experiments on the original kernel and two experiments on the approximated version of the kernel. We will run an experiment with the standard SSK on a smaller subset of the data, with two values for n. As in (Lodhi et al. 2002), all experiments are run on a set of documents from Reuter's news agency, called the Reuters-21578 data set. In all experiments, the results of the SSK kernel is compared with two other common kernels: the standard word kernel (WK) which uses the tf-idf weighting scheme, and the n-grams kernel (NGK) which uses the most frequent character n-grams. For the standard SSK, the results that will be presented include the precision, recall and f1-score of the prediction. The precision measures how many of the classified documents are correctly classified and the recall measures how many of the documents that should have had a positive label got a positive label. The f1-score is a score that combines precision and recall, giving equal weighting to both of them. It is given as $2pr/(p+r)$.

The objectives of this project is to re-implement the kernel and the approximated kernel, and to compare the results of running our reimplementation of the SSK with the NGK and WK kernels.

2 Method

2.1 Implementation of the SSK

The string subsequence kernel was implemented according to the description in (Lodhi et al. 2002). First, the recursive version was constructed as the recursive formula mentioned in section 3. Then, a dynamic programming version that uses memoization was implemented. This was done to avoid having to recompute different values of K, K' and K'' for different places in the recursion tree. In the version using memoization, the values for $K'_i(s,t)$, $K''_i(s,t)$ for $i = 1,...,n$, and for $s = s[1:x]$ and $t = t[1:y]$ for $x = 1,...,|s|$ and $y = 1,...,|t|$ were stored in two matrices. Since K' relies on certain values of K'', and vice versa, both matrices had to be computed at the same time. Then, the $K_n(s,t)$ values were computed for $s = s[1:x]$ and $t = t[1:y]$ for $x = 1,...,|s|$ and $y = 1,...,|t|$ using the stored values for $K'_i(s,t)$ and $K''_i(s,t)$. The final result is then found in the last element in the K matrix. The complexity of this evaluation becomes O($n|s||t| + |s||t|$) = O($n|s||t|$), which is the same complexity as the recursive kernel. However, the speedup is still apparent when running the version using memoization, as compared to the recursive one, even for small strings.

2.2 Running experiments

The experiments were all written from scratch in the programming language *python*. Some data-handling modules were written, for fetching and preprocessing data, and some modules for calculating gram matrices and running experiments on an SVM. The SVM was run using the *sklearn* library SVC. All the experiments were run with the one-vs-all strategy, since the Reuters data set contains multi-label documents. The C-parameter used was the default one in *sklearn*s library, C = 1.0. The *sklearn* library was used to compute the results of the SVM classification. All experiments were run with the value $\lambda = 0.5$.

Several measures were taken to improve the performance of the computationally heavy calculation of the gram matrices. These include introducing multi-threading in the computation of the gram matrix, and converting the code performing the kernel evaluations into *cython* code. *Cython* is a language that closely resembles python but uses some lower-level constructs, therefore having a superior performance.

On all the experiments, a cutoff limit for document length was given at 1500 in order to reduce computation time. This value was determined after plotting the number of documents having different document lengths. The document lengths were normally distributed, peaking at a length of around 500 characters. Documents with lengths above 1500 were uncommon. Therefore, it was determined that one can use that cutoff length and still have generate train and test sets that are representative of the data set at large.

One experiment was performed on the standard version of the SSK, with a training set size of 180 and a test set size of 45. The results were evaluated for the categories 'earn', 'acq', 'crude' and 'corn', where the first two are the most common categories in the data set and the two latter are the most uncommon. The number of training and (test) samples in the experiment in each category were 'earn': 45 (15), 'acq': 45 (10), 'crude' 45 (10) and 'corn': 45 (10).

One experiment was performed on the approximated version of the kernel, comparing its results with the results of the WK and NGK on the same data set. In the original report the experiment on the approximative kernel was done on the whole Reuters data set with the ModeApte split, constituting 9603 training samples and 3299 testing samples. To repeat the experiment on a data set of that size was not feasible for this project, due to a lack of time. Therefore, the experiment was instead run with a training set size of 960 documents, and a test set size of 320. The results from the five most common categories were evaluated.

First, an experiment was run on randomly chosen documents, with the only limitation being a minimum amount of documents in each of the categories. The minimum amount was set to reflect each category's number of occurrences in the data set at large. This gave poor results, our theory is that since the data set is multilabeled with a high variance in the amounts of categories per sample and number of times each category is presented where some categories could be classified as a sub-category of a more frequent category, there were too few training samples to get a clear picture of the multilabeled data.

Therefore, a new experiment was run with a test and training set of the same size as before, but this time only documents belonging to a single category were chosen. This should reduce the chances of one category dominating another, making the problem into a clear One-vs-all classification problem. The specific number of training and test documents were, with training (testing) the following: 'earn': 192 (64), 'acq': 401 (106), 'money-fx': 151 (59), 'grain': 32 (7), 'crude': 184 (84).

3 Result

In table 1 the results from the experiment run on the standard SSK is given. The results shows clearly a poor performance of the SSK in comparison to the NGK and the WK. In (Lodhi et al. 2002) the results of the SSK are more similar to those of the WK and NGK. The reason why the results here are worse seem to be the size of the data set, since that is the most distinct difference between this experiment and the one Lodhi, et al performs. Since each category has the same number of documents in the training set, any difference seen between the categories should not be dependent on the number of documents. Therefore, the performance between the different categories are not very dissimilar. In every example, the result of the SSK got better with the higher number of n, except for in the category 'earn'. For most of the categories, the results of n = 2 were poor, indicating that a substring length of 2 is not informative enough to make up a good classification.

The results from the experiment on the approximated version of the kernel are presented in table 2. Firstly, the results show that the Approximated SSK is comparable to the NGK for n = 4 and 5, while performing worse than the two other kernels for n = 3. The result that sticks out is the one for the category 'grain', in which the WK produces an f1 score of 1 while the approximated SSK give an f1 score of 0 for all three values of n. This is due to the fact that the WK with tfidf takes into account when a term only appears in a few documents in the corpus. SSK does no such normalization, and therefore the category with much fewer documents gives abysmal results. The results also show that the F1-score goes up for higher values of n for the approximated kernel.

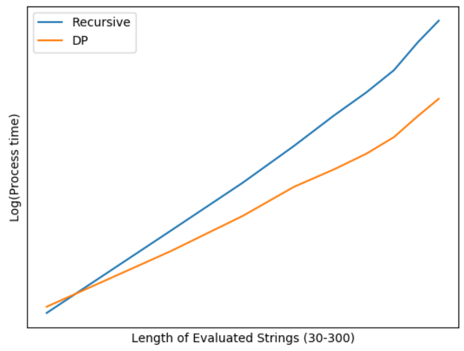
Furthermore, results on performance are presented in figures 1 and **??**. First, the execution time of the recursive version of the SSK is compared to the version using memoization. For different lengths of the input strings, the recursive and the dynamic programming kernels scale in a similar manner, with the memoization kernel growing slightly slower. Observe that the y-scale is logarithmic. For different values of n, the time taken for a kernel evaluation increases more drastically for the recursive kernel, while for the dynamic programming kernel the execution time does not increase as much with increasing values of n. This is the case because for larger values of n, the likelihood of ending up at the same configuration of K, K' and K'' increases, which means more values being recalculated for the recursive kernel.

Table 1: Results for the standard SSK, WK and NGK for a training set size of 180 documents and a test set size of 45. The number of features for the approximative kernel, the NGK and the WK were all set to 500.

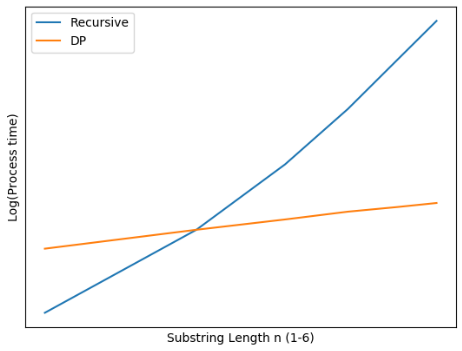
Category	Kernel	Length	F1	Precision	Recall
earn	SSK	2	0.929	1.000	0.867
		3	0.929	1.000	0.867
	NGK	2	**0.968**	0.9375	1.000
		3	0.965	1.000	0.933
	WK	1	0.966	1.000	0.933
acq	SSK	2	0.181	1.000	0.100
		3	0.842	0.889	0.800
	NGK	2	**0.947**	1.000	0.900
		3	0.800	0.800	0.800
	WK	1	0.778	0.875	0.700
crude	SSK	2	0.000	0.000	0.000
		3	0.750	1.000	0.600
	NGK	2	0.778	0.875	0.700
		3	0.824	1.000	0.700
	WK	3	**1.000**	1.000	1.000
corn	SSK	2	0.667	1.000	0.500
		3	**0.947**	1.000	0.900
	NGK	2	0.842	0.889	0.800
		3	0.889	1.000	0.800
	WK	1	**0.947**	1.000	.900

Table 2: F1-scores for the approximative kernel, WK and NGK for a training set size of 960 documents and a test set size of 320. The number of features for the approximative kernel, the NGK and the WK were all set to 500.

Category	**WK**	**NGK**			**Approximated SSK**		
		n			k		
		3	4	5	3	4	5
acq	0.967	0.917	0.939	0.930	0.841	0.934	0.953
crude	0.970	0.899	0.944	0.944	0.112	0.859	0.903
earn	0.992	0.955	0.947	0.969	0.934	0.951	0.976
grain	1.000	0.770	0.714	0.625	0.0	0.0	0.0
money-fx	0.974	0.889	0.973	0.879	0.209	0.788	0.868



(a) Strings with 30-300 characters evaluated. Subsequence length fixed at n=2



(b) Subsequence length 1-6 evaluated, fixed string length of 40 characters

Figure 1: Difference in execution time between the recursive kernel and the dynamic programming kernel for a single kernel evaluation. All values are averaged over 5 iterations.

4 Discussion

The article (Lodhi et al. 2002) is widely cited and widely reimplemented. Some examples of works built on the string subsequence kernel includes (Seewald and Kleedorfer 2007) who uses a version of the SSK based on the recursive calculation of the SSK value with a maximum recursion depth, given as the maximum value for the exponent of the λ parameter. As the exponent of the λ parameter is a degree of contiguity for the two strings being evaluated, their approach essentially disregards substrings that are too spread out. It is therefore an approximation of the SSK, but quite different from the approximation suggested by the original authors of (Lodhi et al. 2002).

Other examples extending the work on the SSK includes (Leslie, Eskin, and Noble 2001) and (Paass et al. 2002), where the first one uses a string kernel based on the SSK for protein classification and the second one is a kernel for identifying phonemes and syllables in spoken language.

We could infer from our experiments that, classifier with N-gram kernel performed better than that of SSK and Approximated SSK on small datasets. Although approximated SSK approach runs faster than that of original SSK, its accuracy is lower than both SSK and N-gram models. Given its time complexity, we would not recommend using string subsequence models for text classification problems. Instead, we suggest developing N-gram models which efficiently handle large data sets.

5 References

[1] S. Behnel et al. “Cython: The Best of Both Worlds”. In: *Computing in Science Engineering* 13.2 (2011), pp. 31–39. ISSN: 1521-9615. DOI: 10.1109/MCSE.2010.118.

[2] Inc. Carnegie Group and Ltd Reuters. “Reuters-21578, Distribution 1.0”. In: (2004).

[3] Thorsten Joachims. “Text categorization with support vector machines: Learning with many relevant features”. In: *Machine learning: ECML-98* (1998), pp. 137–142.

[4] Christina Leslie, Eleazar Eskin, and William Stafford Noble. “The spectrum kernel: A string kernel for SVM protein classification”. In: *Biocomputing 2002*. World Scientific, 2001, pp. 564–575.

[5] Huma Lodhi et al. “Text Classification using String Kernels”. In: *Journal of Machine Learning Research* 2 (2002), pp. 419–444.

[6] Gerhard Paass et al. “SVM classification using sequences of phonemes and syllables”. In: *Principles of Data Mining and Knowledge Discovery* (2002), pp. 373–384.

[7] Alexander K Seewald and Florian Kleedorfer. “Lambda pruning: an approximation of the string subsequence kernel for practical SVM classification and redundancy clustering”. In: *Advances in Data Analysis and Classification* 1.3 (2007), pp. 221–239.